aws

User Guide

Amazon Inspector



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Inspector: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon Inspector? ... 1
Features ... 1
Accessing Amazon Inspector ... 3
Getting started ... 5
Before activating Amazon Inspector ... 5
Getting started tutorial: Activating Amazon Inspector ... 6
Automated scans ... 12
Overview of Amazon Inspector scan types ... 12
Activating a scan type ... 14
Activating scans ... 16
Amazon EC2 instance scanning ... 17
Agent-based scanning ... 17
Agentless scanning ... 23
Managing scan mode ... 25
Excluding instances from Amazon Inspector scans ... 25
Supported operating systems ... 26
Deep inspection for Linux instances ... 26
Scanning Windows EC2 instance ... 32
Machine image scanning ... 34
Eligible machine images ... 36
Scan behaviors for machine image scanning ... 37
Configuring machine image scanning ... 39
On-demand machine image scanning ... 46
Amazon ECR container image scanning ... 56
Scan behaviors for Amazon ECR scanning ... 57
Mapping container images to running containers ... 59
Supported operating systems and media types ... 60
Configuring the Amazon ECR re-scan duration ... 61
Lambda function scanning ... 63
Scan behaviors for Lambda function scanning ... 64
Supported runtimes and functions ... 65
Amazon Inspector Lambda standard scanning ... 65
Amazon Inspector Lambda code scanning ... 67
Deactivating a scan type ... 68

Deactivating scans ... 70
CIS scans ... 71
Amazon EC2 instance requirements for Amazon Inspector CIS scans ... 72
Amazon Virtual Private Cloud endpoint requirements for running CIS scans on private Amazon EC2 instances ... 73
Running CIS scans ... 73
Considerations for managing Amazon Inspector CIS scans with AWS Organizations ... 74
Amazon Inspector owned Amazon S3 buckets used for Amazon Inspector CIS scans ... 76
Creating a CIS scan configuration ... 77
Viewing CIS scan results ... 78
Editing a CIS scan configuration ... 80
Downloading a CIS scan results ... 80
Amazon Inspector Code Security ... 82
Prerequisites ... 82
Activating Code Security ... 82
Creating a customer managed key to access AWS KMS ... 82
Creating an integration ... 85
Creating an integration for GitHub ... 86
Creating an integration for GitLab Self Managed ... 88
Viewing integrations ... 90
Viewing code repositories ... 90
Deleting an integration ... 91
Creating a scan configuration ... 92
Viewing scan configurations ... 95
Editing a scan configuration ... 96
Deleting a scan configuration ... 96
Performing an on-demand scan ... 97
Supported languages ... 97
Deactivating Code Security ... 98
Understanding findings ... 99
Finding types ... 100
Package vulnerability ... 100
Code vulnerability ... 101
Network reachability ... 101
Viewing findings ... 102
Viewing finding details ... 104

Viewing the Amazon Inspector score ... 107
Amazon Inspector score ... 107
Vulnerability Intelligence ... 109
Understanding severity levels for findings ... 110
Software package vulnerability severity ... 110
Code vulnerability severity ... 111
Network reachability severity ... 110
Managing findings ... 114
Filtering findings ... 114
Creating filters in the Amazon Inspector console ... 114
Suppressing findings ... 115
Creating a suppression rule ... 116
Viewing suppressed findings ... 116
Editing a suppression rule ... 117
Deleting a suppression rule ... 117
Exporting findings reports ... 117
Step 1: Verify your permissions ... 118
Step 2: Configure an S3 bucket ... 120
Step 3: Configure an AWS KMS key ... 124
Step 4: Configure and export a findings report ... 127
Troubleshoot errors ... 129
Automating responses to findings with EventBridge ... 130
Event schema ... 131
Creating an EventBridge rule to notify you of Amazon Inspector findings ... 133
EventBridge for Amazon Inspector multi-account environments ... 137
Dashboard ... 138
Viewing the dashboard ... 138
Understanding dashboard components ... 139
Searching the vulnerability database ... 142
Searching the vulnerability database ... 142
Understanding CVE details ... 143
CVE details ... 143
Vulnerability intelligence ... 143
References ... 143
Exporting SBOMs ... 144
Amazon Inspector formats ... 144

Filters for SBOMs ... 149
Configure and export SBOMs ... 150
EventBridge schema ... 153
Amazon EventBridge base schema for Amazon Inspector ... 153
Amazon Inspector finding event schema example ... 154
Amazon Inspector initial scan complete event schema example ... 176
Amazon Inspector on-demand machine image scan event schema examples ... 179
Amazon Inspector coverage event schema example ... 183
Amazon Inspector auto enable schema example ... 185
Amazon Inspector VM Scanner ... 186
Overview ... 186
Benefits ... 186
Deep inspection ... 187
Enabling Inspector VM Scanner ... 187
Requirements ... 188
Enabling Enhanced EC2 Scanning ... 188
Amazon VPC endpoint requirements for Enhanced EC2 Scanning on private Amazon EC2 instances ... 189
Frequently asked questions ... 190
Third-party software attribution ... 191
Manual installation and configuration ... 191
Manually installing and uninstalling Inspector VM Scanner ... 191
Installation paths ... 195
Uninstalling Inspector VM Scanner ... 196
Running Inspector VM Scanner ... 196
Viewing output ... 196
Viewing logs ... 196
Viewing logs with systemd ... 197
Advanced configuration ... 197
Configuring local outputs ... 197
Configuring resource usage ... 199
Configuring scan targets ... 199
Managing periodic execution ... 200
SSM plugin ... 203
The Amazon Inspector SSM plugin for Linux ... 203
The Amazon Inspector SSM plugin for Windows ... 204

Azure integration 206
How it works 206
How AWS Config is used 206
Connector types 206
Scanning methods 207
Prerequisites 207
Configuring Azure 210
Configuring Amazon Inspector 215
Deleting a connector 226
Amazon Inspector SBOM Generator 228
Supported packages types 228
Supported container image configuration checks 229
Installing Sbomgen 229
Using Sbomgen 230
Generate an SBOM for a container image and output the result 231
Generate an SBOM from directories and archives 232
Generate an SBOM from Go or Rust compiled binaries 232
Generate an SBOM from mounted volumes 232
Send an SBOM to Amazon Inspector for vulnerability identification 234
Use additional scanners to enhance detection capabilities 236
Optimize container scans by adjusting maximum file size to scan 237
Limit container image pull size 237
Detect scratch images with filesystem heuristics 239
Collect running process metadata 240
Evaluate CIS Docker Benchmark checks 241
Record FIPS configuration indicators 242
Disable progress indicator 242
Authenticating to private registries with Sbomgen 243
Authenticate using cached credentials (recommended) 243
Authenticate using the interactive method 243
Authenticate using the non-interactive method 244
Example outputs from Sbomgen 244
Previous versions 247
Operating system collection 276
Supported operating system artifacts 277
APK-based OS package collection 278

DPKG-based OS package collection ... 279
RPM-based OS package collection ... 280
Windows OS version collection ... 281
Chainguard image package collection ... 282
Distroless image package collection ... 283
MinimOS package collection ... 284
Docker Hardened Images package collection ... 285
Red Hat Hardened Images package collection ... 286
CleanStart hardened images package collection ... 288
Dependency collection ... 289
Go dependency scanning ... 289
Java dependency scanning ... 292
JavaScript dependency scanning ... 297
.NET dependency scanning ... 303
PHP dependency scanning ... 308
Python dependency scanning ... 311
Ruby dependency scanning ... 318
Rust dependency scanning ... 321
Swift dependency scanning ... 324
Unsupported artifacts ... 327
Ecosystem collection ... 329
Supported ecosystems ... 329
7-Zip ecosystem collection ... 334
AI agent skills ecosystem collection ... 335
AI IDE extensions ecosystem collection ... 336
Amazon Q Developer ecosystem collection ... 338
Apache ecosystem collection ... 339
Atlassian ecosystem collection ... 345
Claude Code ecosystem collection ... 347
Conda ecosystem collection ... 348
Curl ecosystem collection ... 350
Cursor ecosystem collection ... 352
Docker binaries ecosystem collection ... 352
Elasticsearch ecosystem collection ... 354
Generic AI/ML Models ecosystem collection ... 355
GitHub Copilot ecosystem collection ... 356

Google ecosystem collection ... 357
Homebrew ecosystem collection ... 359
HuggingFace ecosystem collection ... 360
Java ecosystem collection ... 360
Jenkins ecosystem collection ... 363
Kiro CLI ecosystem collection ... 364
MariaDB and MySQL ecosystem collection ... 365
Microsoft applications ecosystem collection ... 368
Microsoft SQL Server ecosystem collection ... 372
MongoDB ecosystem collection ... 373
Mozilla ecosystem collection ... 374
Nginx ecosystem collection ... 377
Node.js runtime collection ... 378
Ollama ecosystem collection ... 379
Ollama Model Collector ecosystem collection ... 381
OpenSSH ecosystem collection ... 381
OpenSSL ecosystem collection ... 382
Oracle Database Server collection ... 384
PHP ecosystem collection ... 385
Redis ecosystem collection ... 386
Standalone AI applications ecosystem collection ... 387
Windsurf ecosystem collection ... 388
WordPress ecosystem collection ... 389
Zed ecosystem collection ... 392
SSL/TLS certificate scans ... 393
Using Sbomgen certificate scans ... 393
License collection ... 396
Collect license information ... 396
Supported packages ... 397
Package URLs ... 404
PURL structure ... 404
Version references ... 407
Recommendations ... 407
Java ... 407
JavaScript ... 407
Python ... 408

Using CycloneDX namespaces ... 408
amazon:inspector:sbom_scanner namespace taxonomy ... 409
amazon:inspector:sbom_generator namespace taxonomy ... 410
Plugin developer guide ... 417
Overview ... 417
Quick Start: Creating New Plugins ... 418
IDE Setup ... 420
Plugin Directory Structure ... 420
Discovery Plugins ... 422
Collection Plugins ... 426
Running Your Plugins ... 430
Listing Available Scanners ... 431
Scanner Selection ... 433
Plugin Collision Rules ... 436
Debugging ... 437
API Reference ... 439
Error Handling ... 440
Sandbox Restrictions ... 440
Sharing Code Between Plugins ... 441
Plugin testing guide ... 442
Quick Start ... 443
Directory Layout ... 444
The testing.* API ... 445
Running Tests ... 446
Testing Plugin Helpers ... 447
Behavior and Invariants ... 448
Limitations ... 449
Developer Responsibilities ... 449
Plugin API reference ... 450
Overview ... 450
Plugin Lifecycle Globals ... 452
File I/O ... 456
FileHandle Methods ... 463
Binary Utilities ... 463
Package Output ... 466
Package Provenance ... 470

Component Hashes ... 472
Component Licenses ... 474
CycloneDX Properties ... 475
Property Constants ... 477
Scanner Groups ... 480
Component Type Constants ... 481
PURL Type Constants ... 482
Platform Constants ... 482
Hash Algorithm Constants ... 483
Artifact Info ... 484
System Info ... 486
Regular Expressions ... 487
Structured Parsing ... 488
Windows Registry ... 489
Logging ... 491
Debugging Functions ... 492
Testing API ... 492
Error Handling ... 495
CI/CD integration ... 496
Plugin integration ... 496
Supported CI/CD solutions ... 497
Custom integration ... 497
Set up an account for CI/CD integration ... 498
Sign up for an AWS account ... 498
Configure an IAM role for CI/CD integration ... 499
Amazon Inspector Dockerfile checks ... 500
Using Sbomgen Dockerfile checks ... 500
Supported Dockerfile checks ... 502
Creating a custom CI/CD integration ... 508
Step 1. Configuring AWS account ... 508
Step 2. Installing Sbomgen binary ... 508
Step 3. Using Sbomgen ... 508
Step 4. Calling the Amazon Inspector Scan API ... 509
(Optional) Step 5. Generate and scan SBOM in a single command ... 509
API output formats ... 510
GitHub actions ... 522

Jenkins plugin ... 522
Step 1. Set up an AWS account ... 523
Step 2. Install the Amazon Inspector Jenkins plugin ... 523
(Optional) Step 3. Add Docker credentials to Jenkins ... 523
(Optional) Step 4. Add AWS credentials to Jenkins ... 523
Step 5. Allow Jenkins to display the HTML report ... 526
Step 6. Add Amazon Inspector Scan to your build ... 526
Step 7. View your Amazon Inspector vulnerability report ... 532
Troubleshooting ... 533
CodeCatalyst actions ... 536
CodePipeline actions ... 536
TeamCity plugin ... 537
GitLab components ... 537
Assessing coverage ... 539
Assessing account-level coverage ... 540
Assessing coverage of Amazon EC2 instances ... 540
Amazon EC2 instances status values ... 541
Assessing coverage of Amazon ECR repositories ... 543
Amazon ECR repository scan status values ... 544
Assessing coverage of Amazon ECR container images ... 545
Amazon ECR container image scan status values ... 546
Assessing coverage of AWS Lambda functions ... 547
Lambda functions scan status values ... 548
Assessing coverage of machine images ... 548
Machine image scan status values ... 549
Managing multiple accounts ... 551
Understanding the delegated administrator account and member account ... 551
Organization policy governance model ... 552
Delegated administrator actions ... 552
Member account actions ... 554
Designating an administrator account ... 555
Considerations ... 555
Permissions required to designate a delegated administrator ... 556
Designating a delegated administrator ... 556
Activating Amazon Inspector scans for member accounts ... 558
Disassociating member accounts ... 562

Removing the delegated administrator ... 563
Tagging resources ... 565
Tagging fundamentals ... 565
Adding tags ... 566
Adding tags to Amazon Inspector resources ... 566
Removing tags ... 567
Removing tags from Amazon Inspector resources ... 567
Usage ... 569
Using the usage console ... 569
Understanding how Amazon Inspector calculates usage costs ... 571
About the Amazon Inspector free trial ... 572
Security ... 574
Data protection ... 575
Encryption at rest ... 576
Encryption in transit ... 581
Identity and Access Management ... 581
Audience ... 581
Authenticating with identities ... 582
Managing access using policies ... 583
How Amazon Inspector works with IAM ... 585
Identity-based policy examples ... 590
AWS managed policies ... 594
Using service-linked roles ... 612
Troubleshooting ... 624
Monitoring Amazon Inspector ... 626
CloudTrail logs ... 627
Compliance validation ... 630
Resilience ... 630
Infrastructure security ... 631
Incident response ... 631
Security Research ... 632
Research methodology ... 632
Detection summary ... 633
Recent malicious package reports ... 634
Related resources ... 635
AWS PrivateLink ... 635

Considerations ... 635
Create an interface endpoint ... 636
Integrations ... 637
Using Amazon Inspector with AWS Organizations ... 637
Integrating Amazon Inspector with Amazon ECR ... 637
Amazon Inspector integration with Security Hub CSPM ... 638
Amazon ECR integration ... 638
Activating the integration ... 638
Using the integration with a multi-account environment ... 638
Security Hub CSPM integration ... 639
Viewing Amazon Inspector findings in AWS Security Hub CSPM ... 639
Activating and configuring the Amazon Inspector integration with Security Hub CSPM ... 643
Activating Amazon Inspector from Security Hub CSPM using organization policy ... 643
Disabling the flow of findings from an integration ... 644
Viewing security controls for Amazon Inspector in Security Hub CSPM ... 644
Supported operating systems and programming languages ... 645
Supported operating systems ... 647
Supported operating systems: Amazon EC2 scanning ... 647
Supported operating systems: machine image scanning ... 651
Supported operating systems: Amazon ECR scanning with Amazon Inspector ... 654
Supported operating systems: CIS scanning ... 657
Supported operating systems: Amazon Inspector Scan API ... 658
Discontinued operating systems ... 661
Supported programming languages ... 665
Supported programming languages: Amazon EC2 agentless scanning ... 665
Supported programming languages: Amazon Inspector VM Scanner ... 666
Supported programming languages: Amazon Inspector SSM plugin ... 666
Supported programming languages: Amazon ECR scanning ... 667
Supported runtimes ... 667
Supported runtimes: Amazon Inspector Lambda standard scanning ... 668
Supported runtimes: Amazon Inspector Lambda code scanning ... 669
Deactivating Amazon Inspector ... 671
Deactivating Amazon Inspector managed by organization policies ... 672
Deactivate Amazon Inspector ... 673
Quotas ... 674
Regions and endpoints ... 676

Service endpoints for Amazon Inspector 676
Endpoints for Amazon Inspector Scan API 676
Region-specific feature availability 684
Document history 689
Amazon Inspector Product Updates 689
AWS Glossary 735

What is Amazon Inspector?

Amazon Inspector is a vulnerability management service that automatically discovers workloads and continually scans them for software vulnerabilities and unintended network exposure. Amazon Inspector discovers and scans Amazon EC2 instances, container images in Amazon ECR, and Lambda functions. When Amazon Inspector detects a software vulnerability or unintended network exposure, it creates a finding, which is a detailed report about the issue. You can manage findings in the Amazon Inspector console or API.

> **Note**
>
> When submitting a support request, Amazon Inspector might access and process relevant findings in the AWS Region where they are stored (but within the same geography) to address the issue.

Topics

- Features of Amazon Inspector
- Accessing Amazon Inspector

Features of Amazon Inspector

Centrally manage multiple Amazon Inspector accounts

If your AWS environment has multiple accounts, you can centrally manage your environment through a single account by using AWS Organizations. Using this approach, you can designate an account as the delegated administrator account for Amazon Inspector.

Amazon Inspector can be activated for your entire organization with a single click. Additionally, you can automate activating the service for future members whenever they join your organization. The Amazon Inspector delegated administrator account can manage findings data and certain settings for members of the organization. This includes viewing aggregated findings details for all member accounts, activating or deactivating scans for member accounts, and reviewing scanned resources within the AWS organization.

Continuously scan your environment for vulnerabilities and network exposure

With Amazon Inspector, you don't need to manually schedule or configure assessment scans. Amazon Inspector automatically discovers and begins scanning your eligible resources. Amazon Inspector continues to assess your environment throughout the lifecycle of your resources by automatically rescanning resources in response to changes that could introduce a new vulnerability, such as: installing a new package in an EC2 instance, installing a patch, and when a new common vulnerabilities and exposures (CVE) that impacts the resource is published. Unlike traditional security scanning software, Amazon Inspector has minimal impact on the performance of your fleet.

When vulnerabilities or open network paths are identified, Amazon Inspector produces a finding that you can investigate. The finding includes comprehensive details about the vulnerability, the affected resource, and remediation recommendations. If you appropriately remediate a finding, Amazon Inspector automatically detects the remediation and closes the finding.

Assess vulnerabilities accurately with the Amazon Inspector Risk score

As Amazon Inspector collects information about your environment through scans, it provides severity scores specifically tailored to your environment. Amazon Inspector examines the security metrics that compose the National Vulnerability Database (NVD) base score for a vulnerability and adjusts them according to your compute environment. For example, the service may lower the Amazon Inspector score of a finding for an Amazon EC2 instance if the vulnerability is exploitable over the network but no open network path to the internet is available from the instance. This score is in CVSS format and is a modification of the base Common Vulnerability Scoring System (CVSS) score provided by NVD.

Identify high-impact findings with the Amazon Inspector dashboard

The Amazon Inspector dashboard offers a high-level view of findings from across your environment. From the dashboard, you can access the granular details of a finding. The dashboard contains streamlined information about scan coverage in your environment, your most critical findings, and which resources have the most findings. The risk-based remediation panel in the Amazon Inspector dashboard presents the findings that affect the largest number of instances and images. This panel makes it easier to identify the findings with the greatest impact on your environment, review finding details, and review suggested solutions.

Manage your findings using customizable views

In addition to the dashboard, the Amazon Inspector console offers a **Findings** view. This page lists all findings for your environment and provides the details of individual findings. You can view

findings grouped by category or vulnerability type. In each view, you can further customize your results using filters. You can also use filters to create suppression rules that hide unwanted findings from your views.

You can use filters and suppression rules to generate finding reports that show all findings or a customized selection of findings. Reports can be generated in CSV or JSON formats.

Monitor and process findings with other services and systems

To support integration with other services and systems, Amazon Inspector publishes findings to Amazon EventBridge as finding events. EventBridge is a serverless event bus service that can route findings data to targets such as AWS Lambda functions and Amazon Simple Notification Service (Amazon SNS) topics. With EventBridge, you can monitor and process findings in near-real time as part of your existing security and compliance workflows.

If you have activated AWS Security Hub CSPM, then Amazon Inspector will also publish findings to Security Hub CSPM. Security Hub CSPM is a service that provides a comprehensive view of your security posture across your AWS environment and helps you check your environment against security industry standards and best practices. With Security Hub CSPM, you can more easily monitor and process your findings as part of a broader analysis of your organization's security posture in AWS.

Accessing Amazon Inspector

Amazon Inspector is available in most AWS Regions. For a list of Regions where Amazon Inspector is currently available, see Amazon Inspector endpoints and quotas in the *Amazon Web Services General Reference*. To learn more about AWS Regions, see Managing AWS Regions in the *Amazon Web Services General Reference*. In each Region, you can work with Amazon Inspector in the following ways.

AWS Management Console

The AWS Management Console is a browser-based interface that you can use to create and manage AWS resources. As part of that console, the Amazon Inspector console provides access to your Amazon Inspector account and resources. You can perform Amazon Inspector tasks from the Amazon Inspector console.

AWS command line tools

With AWS command line tools, you can issue commands at your system's command line to perform Amazon Inspector tasks. Using the command line can be faster and more convenient than using the console. The command line tools are also useful if you want to build scripts that perform tasks.

AWS provides two sets of command line tools: the AWS Command Line Interface (AWS CLI) and the AWS Tools for PowerShell. For information about installing and using the AWS CLI, see the AWS Command Line Interface User Guide. For information about installing and using the Tools for PowerShell, see the AWS Tools for PowerShell User Guide.

AWS SDKs

AWS provides SDKs that consist of libraries and sample code for various programming languages and platforms, including Java, Go, Python, C++, and .NET. The SDKs provide convenient, programmatic access to Amazon Inspector and other AWS services. They also handle tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For information about installing and using the AWS SDKs, see Tools to Build on AWS.

Amazon Inspector REST API

The Amazon Inspector REST API gives you comprehensive, programmatic access to your Amazon Inspector account and resources. With this API, you can send HTTPS requests directly to Amazon Inspector. However, unlike the AWS command line tools and SDKs, use of this API requires your application to handle low-level details such as generating a hash to sign a request.

Getting started with Amazon Inspector

This section provides information to consider before activating Amazon Inspector and a getting started tutorial describing how to activate Amazon Inspector and view your findings in the Amazon Inspector console and with the Amazon Inspector API.

Topics

- Before activating Amazon Inspector
- Getting started tutorial: Activating Amazon Inspector

Before activating Amazon Inspector

Before activating Amazon Inspector, consider the following:

Amazon Inspector is a Regional service

Your data is stored in the AWS Region where you activate Amazon Inspector. Repeat the steps in the first part of the getting started tutorial for all AWS Regions where you plan to use Amazon Inspector.

Amazon Inspector creates the service-linked roles AWSServiceRoleForAmazonInspector2 and AWSServiceRoleForAmazonInspector2Agentless

A service-linked role is a role in AWS Identity and Access Management (IAM) that's linked to an AWS servce. AWSServiceRoleForAmazonInspector2 and AWSServiceRoleForAmazonInspector2Agentless allow Amazon Inspector to access AWS services required to perform security assessments.

IAM identities with administrator permissions can enable Amazon Inspector

Protect your credentials by creating users with IAM or AWS IAM Identity Center. This helps you make sure users only have the permissions required to manage Amazon Inspector. For more information, see AWS managed policy: AmazonInspectorFullAccess.

Hybrid scanning is automatically enabled

Hybrid scanning includes agent-based scanning and agentless scanning. By default, Amazon Inspector uses these scan methods on all eligible Amazon EC2 instances. For more information, see Scanning Amazon EC2 instances with Amazon Inspector.

Amazon ECR scanning and Lambda function scanning doesn't require the SSM agent

Agent-based scanning uses the SSM agent to collect software inventory. Agentless scanning uses Amazon EBS snapshots to collect software inverntory.

> **Note**
>
> By default, the SSM agent is already installed in Amazon EC2 instances based on Amazon Machine Images. However, you might need to activate the SSM agent manually in some cases. For more information, see Working with the SSM agent in the *AWS Systems Manager User Guide*.

Monthly costs are based on workloads scanned

For more information, see Amazon Inspector pricing.

Multi-account enablement with AWS Organizations

For organizations using AWS Organizations, Amazon Inspector supports both delegated administrator management and organization policy-based enablement. Organization policies provide centralized governance with automatic enablement for new accounts. For detailed instructions on both approaches, see Getting started tutorial: Activating Amazon Inspector.

Getting started tutorial: Activating Amazon Inspector

This topic describes how to activate Amazon Inspector for a standalone account environment (member account) and multi-account environment (delegated administrator account). When you activate Amazon Inspector, it automatically begins discovering workloads and scanning them for software vulnerabilities and unintended network exposure.

Standalone account environment

The following procedure describes how to activate Amazon Inspector in the console for a member account. To programatically activate Amazon Inspector, inspector2-enablement-with-cli.

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.

2. Choose **Get Started**.
3. Choose **Activate Amazon Inspector**.

When you activate Amazon Inspector for a standalone account, all scan types are activated by default. For information about member accounts, see Understanding the delegated administrator account and member accounts in Amazon Inspector.

Multi-account (with AWS Organizations policy)

AWS Organizations policies provide centralized governance for enabling Amazon Inspector across your organization. When you use an organization policy, Amazon Inspector enablement is automatically managed for all accounts covered by the policy, and member accounts cannot modify policy-managed scanning using Amazon Inspector API.

Prerequisites

- Your account must be part of an AWS Organizations organization.
- You must have permissions to create and manage organization policies in AWS Organizations.
- Trusted access for Amazon Inspector must be enabled in AWS Organizations. For instructions, see Enabling trusted access for Amazon Inspector in the *AWS Organizations User Guide*.
- The Amazon Inspector service-linked roles should exist in the management account. To create them, enable Amazon Inspector in the management account or run the following commands from the management account:
  - `aws iam create-service-linked-role --aws-service-name inspector2.amazonaws.com`
  - `aws iam create-service-linked-role --aws-service-name agentless.inspector2.amazonaws.com`
- An Amazon Inspector delegated administrator should be designated.

> **Note**
>
> Without the service-linked Amazon Inspector roles of management account and delegated administrator, organization policies will enforce Amazon Inspector enablement, but member accounts will not be associated with the Amazon Inspector organization for centralized findings and account management.

To enable Amazon Inspector using AWS Organizations policies

1. Designate a delegated administrator for Amazon Inspector before creating organization policies to ensure member accounts are associated with the Amazon Inspector organization for centralized findings visibility. Sign in to the AWS Organizations management account, open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home, and follow the steps in Designating a delegated administrator for your AWS organization.

   > **Note**
   >
   > We strongly recommend keeping your AWS Organizations Amazon Inspector delegated administrator account ID and Amazon Inspector designated delegated administrator account ID the same. If the AWS Organizations delegated administrator account ID differs from the Amazon Inspector delegated administrator account ID, Amazon Inspector prioritizes the Inspector-designated account ID. When the Amazon Inspector delegated administrator is not set but the AWS Organizations delegated administrator is set and the management account has the Amazon Inspector service-linked roles, Amazon Inspector automatically assigns the AWS Organizations delegated administrator account ID as the Amazon Inspector delegated administrator.

2. In the Amazon Inspector console, navigate to **General settings** from the management account. Under **Delegation policy**, choose **Attach statement**. In the **Attach policy statement** dialog, review the policy, select **I acknowledge that I have reviewed the policy and understand the permissions it grants**, and then choose **Attach statement**.

   > **Important**
   >
   > The management account must have the following permissions to attach the delegation policy statement:
   >
   > - Amazon Inspector permissions from the AmazonInspector2FullAccess_v2 managed policy
   > - AWS Organizations `organizations:PutResourcePolicy` permission from the AWSOrganizationsFullAccess managed policy

If the `organizations:PutResourcePolicy` permission is missing, the operation fails with the error: `Failed to attach statement to the delegation policy`.

3. Next, create an Amazon Inspector AWS Organizations policy. From the navigation pane, choose **Management**, and then choose **Configurations**.
4. Configure vulnerability management policy. Provide **Details** with name and description (optional) for the policy.
5. On the Configure Inspector page, in the **Details** section, enter a name and a description for the policy. In the **Capability Selection**, do one of the following:
   - Choose **Configure and enable all capabilities (Recommended)**. This turns on all of the Inspector capabilities including EC2, ECR, Lambda standard, Lambda code scan, and Code Security.
   - Choose **Select subset of capabilities**. Select any scan type capabilities that should be turned on.
6. In the **Account selection** section, select one of the following options:
   - Choose **All organizational units and accounts** if you want to apply the configuration to all organizational units and accounts.
   - Choose **Specific organizational units and accounts** if you want to apply the configuration to specific organizational units and accounts. If you choose this option, use the search bar or organizational structure tree to specify the organizational units and accounts where the policy will be applied.
   - Choose **No organizational units or accounts** if you do not want to apply the configuration to any organizational unit or account.
7. In the **Regions** section, choose **Enable all Regions**, **Disable all Regions**, or **Specify Regions**.
   - If you choose **Enable all Regions**, you can determine whether to automatically enable new Regions.
   - If you choose **Disable all Regions**, you can determine whether to automatically disable new Regions.
   - If you choose **Specify Regions**, you must choose which Regions you want to enable and disable.

(Optional) For **Advanced settings**, refer to the guidance from AWS Organizations.

(Optional) For **Resource tags**, add tags as key-value pairs to help you easily identify the configuration.

8. Choose **Next**, review your changes, and then choose **Apply**. Your target accounts are configured based on the policy. The configuration status of your policy displays at the top of the Policies page. Each capability provides a status on whether it was configured or where there are deployment failures. For any failures, choose the link for the failure message to see more details. To view the effective policy at the account level, you can review the Organization tab on the Configurations page where you can choose an account.

When Amazon Inspector is enabled through organization policies, accounts covered by the policy cannot disable the policy-managed scan types through the Amazon Inspector API or console. For detailed information about what delegated administrators and member accounts can and cannot do under organization policies, see Managing multiple accounts in Amazon Inspector with AWS Organizations.

Multi-account (without AWS Organizations policy)

> **Note**
>
> You must use the AWS Organizations management account to complete this procedure. Only the AWS Organizations management account can designate a delegated administrator. Permissions might be required to designate a delegated administrator. For more information, see Permissions required to designate a delegated administrator.

When you activate Amazon Inspector for the first time, Amazon Inspector creates the service linked role `AWSServiceRoleForAmazonInspector` for the account. For information about how Amazon Inspector uses service-linked roles, see Using service-linked roles for Amazon Inspector.

To designate a delegated administrator for Amazon Inspector

1. Sign in to the AWS Organizations management account, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Choose **Get started**.

3. Under **Delegated administrator**, enter the 12-digit ID of the AWS account you want to designate as the delegated administrator.
4. Choose **Delegate**, and then choose **Delegate** again.
5. (Optional) If you want to activate Amazon Inspector for the AWS Organizations management account, choose **Activate Amazon Inspector** under **Service permissions**.

When you designate a delegated administrator, all scan types are activated for the account by default. For information about the delegated administrator account,see Understanding the delegated administrator account and member accounts in Amazon Inspector.

Automated scan types in Amazon Inspector

Amazon Inspector uses a purpose-built scanning engine that monitors your resources for actionable software vulnerabilities and unintended network exposure. When Amazon Inspector detects a software vulnerability or unintended network exposure, it creates a finding. When you activate Amazon Inspector for the first time, your account is automatically enrolled in all scan types, which include Amazon Amazon EC2 scanning, Amazon ECR Scanning, and Lambda standard scanning.

> **Note**
>
> Lambda code scanning is an optional layer of Lambda function scanning that you can activate at any time.

> **Note**
>
> Machine image scanning and Code Security for Amazon Inspector are optional scan types that you activate separately. Your account isn't automatically enrolled in them when you activate Amazon Inspector for the first time.

Topics

- Overview of Amazon Inspector scan types
- Activating a scan type
- Scanning Amazon EC2 instances with Amazon Inspector
- Scanning machine images with Amazon Inspector
- Scanning Amazon Elastic Container Registry container images with Amazon Inspector
- Scanning AWS Lambda functions with Amazon Inspector
- Deactivating a scan type in Amazon Inspector

Overview of Amazon Inspector scan types

Amazon Inspector provides different scan types, which focus on specific resource types in your AWS environment.

Amazon EC2 scanning

When you activate Amazon EC2 scanning, Amazon Inspector scans your EC2 instances for common vulnerabilities and exposures (CVEs), network exposure issues, network reachability issues, operating system and programming language package vulnerabilities. Amazon Inspector performs scans through the use of the SSM agent installed on your instance or through Amazon EBS snapshots of instances. For more information, see Scanning Amazon EC2 instances with Amazon Inspector. By default, when you activate Amazon EC2 scanning, you automatically enable hybrid scanning mode. For more information, see Agentless scanning.

Machine image scanning

When you activate machine image scanning, Amazon Inspector discovers the machine images (AMIs) that your account owns and scans them for operating system and programming language package vulnerabilities. Amazon Inspector collects a software inventory from the Amazon EBS snapshots that back each machine image, so you don't need to launch an instance from an image for Amazon Inspector to scan it. By default, machine image scanning is continuous: Amazon Inspector scans new images as you create, copy, register, or enable them, and rescans in-scope images when new CVEs are published. You control which images are in scope with scan eligibility windows, a scan mode, and exclusion options. For more information, see Scanning machine images with Amazon Inspector.

You can also scan a single machine image on demand, without activating Amazon Inspector. For more information, see Running on-demand machine image scans with Amazon Inspector.

> **Note**
>
> Findings from on-demand machine image scans are an exception. Amazon Inspector delivers them only through Amazon EventBridge, so they don't appear in the Amazon Inspector console, aren't returned by `ListFindings`, and aren't sent to Security Hub CSPM. For more information, see Running on-demand machine image scans with Amazon Inspector.

Amazon ECR scanning

When you activate Amazon ECR scanning, Amazon Inspector converts all of the repositories in your private registry from basic scanning container repositories to enhanced scanning repositories. You can configure this setting with inclusion rules to scan on-push only or

to scan select repositories. Amazon Inspector only scans ECR container images which are active (`imageStatus` field is `ACTIVE`) in ECR. Amazon Inspector scans all images pushed or transitioned to active (`lastActivatedAt`) in ECR, or pulled, within the configured re-scan duration. The default re-scan duration for new accounts, including new accounts added to an organization, is 7 days. For accounts that activated Amazon ECR scanning from May 16, 2024, to October 14, 2026, the default re-scan duration is 14 days. For Amazon Inspector accounts created prior to May 16, 2024, the default configuration monitors images that were pushed or pulled within the last 90 days. You can change this setting at any time. For more information, see Scanning Amazon Elastic Container Registry container images with Amazon Inspector.

Lambda standard scanning

When you activate Lambda standard scanning, Amazon Inspector discovers all of the Lambda functions in your account and immediately scans them for vulnerabilities. Amazon Inspector scans new Lambda functions and layers when they're deployed. Amazon Inspector rescans them when they're updated or when new CVEs are published. For more information, scanning, see Scanning AWS Lambda functions with Amazon Inspector.

Lambda standard scanning + Lambda code scanning

When you activate Lambda code scanning, Amazon Inspector discovers the Lambda functions and layers in your account and scans them for code vulnerabilities. This type of scanning evaluates application package dependencies used in a Lambda function for CVEs. When you activate this scan type, you also activate Lambda standard scanning. For more information, see Scanning AWS Lambda functions with Amazon Inspector.

Code Security for Amazon Inspector

This scan type leverages the Amazon Q Developer scanning engine to scan first-party application code, third-party application dependencies, and Infrastructure as Code for vulnerabilities For more information, see Code Security for Amazon Inspector.

Activating a scan type

You can activate a scan type at any time. When you activate a scan type, Amazon Inspector begins scanning eligible resources for the scan type.

Amazon EC2 scanning

This scan type extracts metadata from an Amazon EC2 instance before comparing the metadata against rules collected from security advisories. When you activate this scan type, Amazon

Inspector scans all eligible Amazon EC2 instances in your account for package vulnerabilities and network reachability issues. After you activate this scan type, you can view how many instances are being scanned in the **Instances** tab.

Machine image scanning

This scan type extracts a software inventory from the Amazon EBS snapshots that back a machine image (AMI) before comparing that inventory against rules collected from security advisories. When you activate this scan type, Amazon Inspector scans all eligible machine images that your account owns for package vulnerabilities and continues to monitor them. After you activate this scan type, you can view how many machine images are being scanned in the **Machine images** tab.

> **Note**
>
> After you activate machine image scanning, review your scan configuration to control which machine images are in scope. For more information, see Configuring machine image scanning.

Amazon ECR scanning

This scan type scans container images and container repositories in Amazon ECR. When you activate this scan type, you change the scanning configuration setting for your private registry from basic scanning to enhanced scanning. After you activate Amazon ECR scanning, you can view how many images and repositories are being scanned in the **Container images** and **Container repositories** tabs.

Lambda standard scanning + Lambda code scanning

Lambda standard scanning is the default Lambda scan type. When you activate Lambda standard scanning, all of your Lambda functions are scanned for software vulnerabilities, as long as they were invoked or updated in the last 90 days. After you activate Lambda standard scanning, you view how many Lambda functions are being scanned in the **Lambda functions** tab.

Lambda code scanning scans custom application code in a Lambda function. When you activate Lambda code scanning, all of your Lambda functions will be scanned for code vulnerabilities, as long as they were invoked or updated in the last 90 days. After you activate Lambda standard scanning, you can view how many Lambda functions are being scanned for code vulnerabilities in the **Lambda functions** tab.

> **Note**
>
> If you want to activate Lambda code scanning, you must activate Lambda standard scanning first.

Amazon Inspector Code Security

This scan type scans first-party application code, third-party application dependencies, and Infrastructure as Code for vulnerabilities. When you activate Code Security, Amazon Inspector begins scanning your code repositories for code vulnerabilities based on your scan configurations. After you activate Amazon Inspector Code Security, you can view how many code repositories are being scanned in the **Code repositories** tab.

Activating scans

The following procedure describes how to activate a scan type in Amazon Inspector.

> **Note**
>
> If you're the delegated administrator for an AWS organization, you can enable Amazon Inspector scan types for multiple accounts in multiple Regions using a shell script. For more information, see inspector2-enablement-with-cli on GitHub. Otherwise, complete the following steps while signed in as the Amazon Inspector delegated administrator.

Console

To activate scans

1. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Using the AWS Region selector in the upper-right corner of the page, select the Region where you want to activate a new scan type.
3. In the navigation pane, choose **Account management**.
4. On the **Account management** page, select the accounts for which you would like to activate a scan type.
5. Choose **Activate** and select the type of scanning you would like to activate.

6. (Recommended) Repeat these steps in each AWS Region for which you want to activate that scan type.

API

Run the Enable API operation. In the request, provide the account IDs you are activating scans for, and idempotency token, and one or more of `EC2`, `MACHINE_IMAGE`, `ECR`, `LAMBDA`, or `LAMBDA_CODE` for `resourceTypes` to activate scans of that type.

Scanning Amazon EC2 instances with Amazon Inspector

Amazon Inspector Amazon EC2 scanning extracts metadata from your EC2 instance before comparing the metadata against rules collected from security advisories. Amazon Inspector scans instances for package vulnerabilities and network reachability issues to produce findings. Amazon Inspector performs network reachability scans once every 12 hours and package vulnerability scans on a variable cadence that depends on the scan method associated with the EC2 instance.

Package vulnerability scans can be performed using an agent-based or agentless scan method. Both of these scan methods determine how and when Amazon Inspector collects the software inventory from an EC2 instance for package vulnerability scans. Agent-based scanning collects software inventory from your instances using the Amazon EC2 Systems Manager (SSM) agent, and agentless scanning collects software inventory using Amazon EBS snapshots.

Amazon Inspector uses the scan methods that you activate for your account. When you activate Amazon Inspector for the first time, your account is automatically enrolled in hybrid scanning, which uses both scan methods. However, you can change this setting at any time. For information about how to activate a scan type, see Activating a scan type. This section provides information about Amazon EC2 scanning.

Agent-based scanning

Enhanced EC2 Scanning

Enhanced EC2 Scanning is performed using the Amazon Inspector VM Scanner. This scanner is installed and updated using SSM associations. Customers can opt in by going to their Amazon Inspector console and visiting the **Settings** > **Scan Settings** page. Choose **Start Upgrade** to begin Enhanced EC2 Scanning.

Recommendation

We recommend that you upgrade to Enhanced EC2 Scanning. The Amazon Inspector VM Scanner uses the same scanning mechanism across operating systems and across other Amazon Inspector supported resources, which produces more consistent findings than the Amazon Inspector SSM plugin. On Windows in particular, it avoids the per-query timeouts that can cause the Amazon Inspector SSM plugin to report findings inconsistently.

1. Amazon Inspector creates SSM associations in your account to collect inventory from your instances. These associations install plugins on individual instances to collect inventory.
2. Using system tools like Systemd and Scheduled Tasks, Inspector VM Scanner extracts package inventory from an instance and communicates that information to Amazon Inspector.
3. Amazon Inspector evaluates the extracted inventory and generates findings for any detected vulnerabilities.

Standard scanning

Agent-based scans are performed continuously using the Amazon Inspector SSM plugin on all eligible instances. For agent-based scans, Amazon Inspector uses SSM associations, and plugins installed through these associations, to collect software inventory from your instances. In addition to package vulnerability scans for operating system packages, Amazon Inspector agent-based scanning can also detect package vulnerabilities for application programming language packages through Amazon Inspector deep inspection.

The following process explains how Amazon Inspector uses SSM to collect inventory and perform agent-based scans:

1. Amazon Inspector creates SSM associations in your account to collect inventory from your instances. These associations install plugins on individual instances to collect inventory.
2. Using SSM, Amazon Inspector extracts package inventory from an instance.
3. Amazon Inspector evaluates the extracted inventory and generates findings for any detected vulnerabilities.

> **Note**
>
> For agent-based scanning, the Amazon EC2 instance must be managed by SSM in same AWS account.

Amazon VPC endpoint requirements for Enhanced EC2 Scanning on private Amazon EC2 instances

You can run Enhanced EC2 Scanning on Amazon EC2 instances over an Amazon network. However, if you want to run Enhanced EC2 Scanning on private Amazon EC2 instances, you must create Amazon VPC endpoints. The following endpoints are required:

- `com.amazonaws.region.inspector2-telemetry`
- `com.amazonaws.region.s3`
- `com.amazonaws.region.ssm`
- `com.amazonaws.region.ssmmessages`

Where *`region`* is the Region code for the applicable AWS Region.

For more information, see Improve the security of Amazon EC2 instances by using Amazon VPC endpoints for Systems Manager in the *AWS Systems Manager User Guide*.

> **Note**
>
> Beginning with SSM Agent version 3.3.40.0, SSM Agent uses the `ssmmessages` endpoint instead of the `ec2messages` endpoint whenever it's available. Earlier versions of SSM Agent use the `ec2messages` endpoint. This endpoint is supported only in AWS Regions launched before 2024. For more information, see Reference: ec2messages, ssmmessages, and other API operations in the *AWS Systems Manager User Guide*.

> **Note**
>
> Currently, some AWS Regions don't support the `com.amazonaws.region.inspector2-telemetry` endpoint.

Eligible instances

Amazon Inspector will use the agent-based method to scan an instance if it meets the following conditions:

- The instance has a supported OS. For a list of supported OS see the **Agent-based scan support** column of the section called "Supported operating systems: Amazon EC2 scanning".
- The instance is not excluded from scans by Amazon Inspector EC2 exclusion tags.

Agent-based scan behaviors

When using the agent-based scan method, Amazon Inspector initiates new vulnerability scans of EC2 instances in the following situations:

- When you launch a new EC2 instance.
- When you install new software on an existing EC2 instance (Linux and Mac).
- When Amazon Inspector adds a new common vulnerabilities and exposures (CVE) item to its database, and that CVE is relevant to your EC2 instance (Linux and Mac).

Amazon Inspector updates the **Last scanned** field for an EC2 instance when an initial scan is completed. After this, the **Last scanned** field is updated when Amazon Inspector evaluates SSM inventory (every 30 minutes by default), or when an instance is re-scanned because a new CVE impacting that instance was added to the Amazon Inspector database.

You can check when an EC2 instance was last scanned for vulnerabilities from the Instances tab on the **Account management** page or by using the ListCoverage command.

Configuring the SSM Agent

In order for Amazon Inspector to detect software vulnerabilities for an Amazon EC2 instance using the agent-based scan method, the instance must be a managed instance in Amazon EC2 Systems Manager (SSM). An SSM managed instance has the SSM Agent installed and running, and SSM has permission to manage the instance. If you are already using SSM to manage your instances, no other steps are needed for agent-based scans.

The SSM Agent is installed by default on EC2 instances created from some Amazon Machine Images (AMIs). For more information, see About SSM Agent in the *AWS Systems Manager User*

Guide. However, even if it's installed, you may need to activate the SSM Agent manually, and grant SSM permission to manage your instance.

The following procedure describes how to configure an Amazon EC2 instance as a managed instance using an IAM instance profile. The procedure also provides links to more detailed information in the *AWS Systems Manager User Guide*.

AmazonSSMManagedInstanceCore is the recommended policy to use when you attach an instance profile. This policy has all the permissions needed for Amazon Inspector EC2 scanning.

> **Note**
>
> You can also automate SSM management of all your EC2 instances, without the use of IAM instance profiles, by using SSM Default Host Management Configuration. For more information, see Default Host Management Configuration.

To configure SSM for an Amazon EC2 instance

1. If it's not already installed by your operating system vendor, install the SSM Agent. For more information, see Working with SSM Agent.
2. Use the AWS CLI to verify that the SSM Agent is running. For more information, see Checking SSM Agent status and starting the agent.
3. Grant permission for SSM to manage your instance. You can grant permission by creating an IAM instance profile and attaching it to your instance. We recommend using the AmazonSSMManagedInstanceCore policy, because this policy has the permissions for SSM Distributor, SSM Inventory and SSM State manager, that Amazon Inspector needs for scans. For instructions on creating an instance profile with these permissions and attaching it to an instance, see Configure instance permissions for Systems Manager.
4. (Optional) Activate automatic updates for the SSM Agent. For more information, see Automating updates to SSM Agent.
5. (Optional) Configure Systems Manager to use an Amazon Virtual Private Cloud (Amazon VPC) endpoint. For more information, see Create Amazon VPC endpoints.

SSM resources created for scanning

Amazon Inspector requires a number of SSM resources in your account to run Amazon EC2 scans. The following resources are created when you first activate Amazon Inspector EC2 scanning:

> **Note**
>
> If any of these SSM resources are deleted while Amazon Inspector Amazon EC2 scanning is activated for your account, Amazon Inspector will attempt to recreate them at the next scan interval.

`InspectorInventoryCollection-do-not-delete`

This is a Systems Manager State Manager (SSM) association that Amazon Inspector uses to collect software application inventory from your Amazon EC2 instances. If your account already has an SSM association for collecting inventory from `InstanceIds*`, Amazon Inspector will use that instead of creating its own.

`InspectorResourceDataSync-do-not-delete`

This is a resource data sync that Amazon Inspector uses to send collected inventory data from your Amazon EC2 instances to an Amazon S3 bucket owned by Amazon Inspector. For more information, see Configuring resource data sync for Inventory in the *AWS Systems Manager User Guide*.

`InspectorVmScannerDistributor-do-not-delete`

This is an SSM association that Amazon Inspector uses to install and update the Amazon Inspector VM Scanner on your Amazon EC2 instances.

`InspectorDistributor-do-not-delete`

This is an SSM association Amazon Inspector uses for scanning Windows instances. This association installs the Amazon Inspector SSM plugin on your Windows instances. If the plugin file is inadvertently deleted this association will reinstall it at the next association interval.

`InvokeInspectorSsmPlugin-do-not-delete`

This is an SSM association Amazon Inspector uses for scanning Windows instances. This association allows Amazon Inspector to initiate scans using the plugin, you can also use it to set custom intervals for scans of Windows instances. For more information, see Setting custom schedules for Windows instance scans.

`InspectorLinuxDistributor-do-not-delete`

This is an SSM association that Amazon Inspector uses for Amazon EC2 Linux deep inspection. This association installs the Amazon Inspector SSM plugin on your Linux instances.

`InvokeInspectorLinuxSsmPlugin-do-not-delete`

This is an SSM association Amazon Inspector uses for Amazon EC2 Linux deep inspection. This association allows Amazon Inspector to initiate scans using the plugin.

Note

When you deactivate Amazon Inspector Amazon EC2 scanning or deep inspection, the SSM resource `InvokeInspectorLinuxSsmPlugin-do-not-delete` is no longer invoked.

Agentless scanning

Amazon Inspector uses the agentless scanning method on eligible instances when your account is in hybrid scanning mode. Hybrid scanning mode includes agent-based and agentless scans and is automatically enabled when you activate Amazon EC2 scanning.

For agentless scans, Amazon Inspector uses EBS snapshots to collect a software inventory from your instances. Agentless scanning scans instances for operating system and application programming language package vulnerabilities..

Note

When scanning Linux instances for application programming language package vulnerabilities, the agentless method scans all available paths, whereas agent-based scanning only scans the default paths and additional paths you specify as part of Amazon Inspector deep inspection for Linux-based Amazon EC2 instances. This may result in the same instance having different findings depending on whether it is scanned using the agent-based method or agentless method.

The following process explains how Amazon Inspector uses EBS snapshots to collect inventory and perform agentless scans:

1. Amazon Inspector creates an EBS snapshot of all volumes attached to the instance. While Amazon Inspector is using it, the snapshot is stored in your account and tagged with `InspectorScan` as a tag key, and a unique scan ID as the tag value.

2. Amazon Inspector retrieves data from the snapshots using EBS direct APIs and evaluates them for vulnerabilities. Findings are generated for any detected vulnerabilities.
3. Amazon Inspector deletes the EBS snapshots it created in your account.

Eligible instances

Amazon Inspector will use the agentless method to scan an instance if it meets the following conditions:

- The instance has a supported OS. For more information, see the >Agent-based scan support column of the section called "Supported operating systems: Amazon EC2 scanning".
- The instance has a status of `Unmanaged EC2 instance`, `Stale inventory`, or `No inventory`.
- The instance is backed by Amazon EBS and has one of the following file system formats:
  - `ext3`
  - `ext4`
  - `xfs`
- The instance isn't excluded from scans through Amazon EC2 exclusion tags.
- The number of volumes attached to the instance is less than 8 and have a combined size that's less than or equal to 1200 GB.

Agentless scan behaviors

When your account is configured for **Hybrid scanning**, Amazon Inspector performs agentless scans on eligible instances every 24 hours. Amazon Inspector detects and scans newly eligible instances every hour, which includes new instances without SSM agents, or pre-existing instances with statuses that have changed to `SSM_UNMANAGED`.

Amazon Inspector updates the **Last scanned** field for an Amazon EC2 instance whenever it scans extracted snapshots from an instance after an agentless scan.

You can check when an EC2 instance was last scanned for vulnerabilities from the Instances tab on the Account management page or by using the ListCoverage command.

Managing scan mode

Your EC2 scan mode determines which scan methods Amazon Inspector will use when performing EC2 scans in your account. You can view the scan mode for your account from the EC2 scanning settings page under **General settings**. Standalone accounts or Amazon Inspector delegated administrators can change the scan mode. When you set the scan mode as the Amazon Inspector delegated administrator that scan mode is set for all member accounts in your organization. Amazon Inspector has the following scan modes:

Agent-based scanning – In this scan mode, Amazon Inspector will exclusively use the agent-based scan method when scanning for package vulnerabilities. This scan mode only scans SSM managed instances in your account, but has the benefit of providing continuous scans in response to new CVE's or changes to the instances. Agent-based scanning also provides Amazon Inspector deep Inspection for eligible instances. This is the default scan mode for newly activated accounts.

Hybrid scanning – In this scan mode, Amazon Inspector uses a combination of both agent-based and agentless methods to scan for package vulnerabilities. For eligible EC2 instances that have the SSM agent installed and configured, Amazon Inspector uses the agent-based method. For eligible instances that aren't SSM managed, Amazon Inspector will use the agentless method for eligible EBS-backed instances.

To change the scan mode

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Using the AWS Region selector in the upper-right corner of the page, select the Region where you want to change your EC2 scan mode.
3. From the side navigation panel, under **General settings**, select **EC2 scanning settings**.
4. Under **Scan Mode**, select **Edit**.
5. Choose a scan mode and then select **Save changes**.

Excluding instances from Amazon Inspector scans

You can exclude Linux and Windows instances from Amazon Inspector scans by tagging these instances with the `InspectorEc2Exclusion` key. Tag key is case-insensitive. Including a tag value is optional. For information about adding tags, see Tag your Amazon EC2 resources.

When you tag an instance for exclusion from Amazon Inspector scans, Amazon Inspector marks the instance as excluded and won't create findings for it. However, the Amazon Inspector SSM plugin will continue to be invoked. To prevent the plugin from being invoked, you must allow access to tags in instance metadata.

> **Note**
>
> You're not charged for excluded instances.

Additionally, you can exclude an instance from agentless scans by tagging the AWS KMS key used to encrypt that volume with the `InspectorEc2Exclusion` tag. For more information, see Tagging keys.

Supported operating systems

Amazon Inspector scans supported Mac, Windows, and Linux instances for vulnerabilities in operating system packages. For Linux instances, Amazon Inspector can produce findings for application programming language packages using Amazon Inspector deep inspection for Linux-based Amazon EC2 instances. For Mac and Windows instances only operating system packages are scanned.

For information about supported operating systems, including which operating systems can be scanned without an SSM agent, see Amazon EC2 instances status values.

Amazon Inspector deep inspection for Linux-based Amazon EC2 instances

> **Note**
>
> This page applies to customers that have not opted in to Enhanced EC2 Scanning. Customers opted into Enhanced EC2 Scanning do not need any additional policy configuration.

Amazon Inspector expands Amazon EC2 scanning coverage to include deep inspection. With deep inspection, Amazon Inspector detects package vulnerabilities for application programming

language packages in your Linux-based Amazon EC2 instances. Amazon Inspector scans default paths for programming language package libraries. However, you can configure custom paths in addition to the paths that Amazon Inspector scans by default.

> **Note**
>
> Deep inspection requires `ssm:PutInventory` and `ssm:GetParameter` permissions. If an IAM instance profile is configured on the instance, Amazon Inspector uses that profile and ignores the DHMC role. The instance profile must include these permissions. If no instance profile is set, Amazon Inspector uses the configured Default Host Management Configuration role, which must include these permissions.

To perform deep inspection scans for your Linux-based Amazon EC2 instances, Amazon Inspector uses data collected with the Amazon Inspector SSM plugin. To manage the Amazon Inspector SSM plugin and perform deep inspection for Linux, Amazon Inspector automatically creates the SSM association `InvokeInspectorLinuxSsmPlugin-do-not-delete` in your account. Amazon Inspector collects updated application inventory from your Linux-based Amazon EC2 instances every 6 hours.

> **Deep inspection on Windows and macOS**
>
> Deep inspection is not supported for Windows or Mac instances unless you use Enhanced EC2 Scanning. With Enhanced EC2 Scanning, deep inspection is performed by the Amazon Inspector VM Scanner, which supports Linux, Windows, and macOS instances.

This section describes how to manage Amazon Inspector deep inspection for Amazon EC2 instances, including how to set custom paths for Amazon Inspector to scan.

Topics

- Accessing or deactivating deep inspection
- Custom paths for Amazon Inspector deep inspection
- Custom schedules for Amazon Inspector deep inspection
- Supported programming languages

Accessing or deactivating deep inspection

> **Note**
>
> For accounts that activate Amazon Inspector after April 17, 2023, deep inspection is automatically activated as part of Amazon EC2 scanning.

To manage deep inspection

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home
2. From the navigation pane, choose **General settings**, and then choose Amazon EC2 scanning settings.
3. Under **Deep inspection of Amazon EC2 instance**, you can set custom paths for your organization or for your own account.

You can check the activation status programmatically for a single account with the GetEc2DeepInspectionConfiguration API. You can check the activation status programmatically for multiple accounts with the BatchGetMemberEc2DeepInspectionStatus API.

If you activated Amazon Inspector before April 17, 2023, you can activate deep inspection through the console banner or the UpdateEc2DeepInspectionConfiguration API. If you're the delegated administrator for an organization in Amazon Inspector, you can use the BatchUpdateMemberEc2DeepInspectionStatus API to activate deep inspection for yourself and your member accounts.

You can deactivate deep inspection through the UpdateEc2DeepInspectionConfiguration API. Member accounts in an organization can't deactivate deep inspection. Instead, the member account must be deactivated by their delegated administrator using the BatchUpdateMemberEc2DeepInspectionStatus API.

Custom paths for Amazon Inspector deep inspection

You can set custom paths for Amazon Inspector to scan during deep inspection of your Amazon EC2 instances. When you set a custom path, Amazon Inspector scans packages in that directory and all of the sub-directories in it, in addition to the locations Amazon Inspector scans by default.

> **Important**
>
> Custom paths are supported for Linux, Windows, and macOS instances when you use Enhanced EC2 Scanning (the Amazon Inspector VM Scanner). If you use the legacy Amazon Inspector SSM plugin instead of Enhanced EC2 Scanning, deep inspection and custom paths are available only for Linux instances.

All accounts can define up to 5 custom paths. The delegated administrator for an organization can define 10 custom paths.

Amazon Inspector scans all custom paths in addition to the locations that Amazon Inspector scans by default for all accounts. On Linux instances, the default locations include the following:

- `/usr/lib`
- `/usr/lib64`
- `/usr/local/lib`
- `/usr/local/lib64`

On Windows instances, Amazon Inspector scans standard operating system and programming language package locations on the system drive by default. Software installed in a custom or non-standard location – for example, a database or application server on a non-system drive such as `D:\` – is scanned only if you add it as a custom path.

> **Note**
>
> Custom paths must be local paths. Amazon Inspector doesn't scan mapped network paths, such as Network File System mounts or Amazon S3 file system mounts.

Formatting custom paths

A custom path cannot be longer than 256 characters. The following is an example of how a custom path might look:

Example path

```
/home/usr1/project01
```

> **Note**
>
> The package limit per instance is 5,000. The maximum package inventory collection time is 15 minutes. Amazon Inspector recommends that you choose custom paths to avoid these limits.

Setting a custom path in the Amazon Inspector console and with the Amazon Inspector API

The following procedures describe how to set a custom path for Amazon Inspector deep inspection in the Amazon Inspector console and with the Amazon Inspector API. After you set a custom path, Amazon Inspector includes the path in the next deep inspection.

Console

1. Sign in to the AWS Management Console as the delegated administrator, and open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home
2. Use the AWS Region selector to choose the Region where you want to activate Lambda standard scanning.
3. From the navigation pane, choose **General settings**, and then choose **EC2 scanning settings**.
4. Under **Custom paths for your own account**, choose **Edit**.
5. In the path text boxes, enter your custom paths.
6. Choose **Save**.

API

Run the UpdateEc2DeepInspectionConfiguration command. For `packagePaths` specify an array of paths to scan.

Custom schedules for Amazon Inspector deep inspection

By default, Amazon Inspector collects an application inventory from Amazon EC2 instances every 6 hours. However, you can run the following commands to control how often Amazon Inspector does this.

Example command 1: List associations to view association ID and current interval

The following command shows the association ID for the association `InvokeInspectorLinuxSsmPlugin-do-not-delete`.

```
aws ssm list-associations \
--association-filter-list "key=AssociationName,value=InvokeInspectorLinuxSsmPlugin-do-not-delete" \
--region your-Region
```

Example command 2: Update association to include new interval

The following command uses the association ID for the association `InvokeInspectorLinuxSsmPlugin-do-not-delete`. You can set the rate for `schedule-expression` from 6 hours to a new interval, such as 12 hours.

```
aws ssm update-association \
--association-id "your-association-ID" \
--association-name "InvokeInspectorLinuxSsmPlugin-do-not-delete" \
--schedule-expression "rate(6 hours)" \
--region your-Region
```

> **Note**
>
> Depending on your use case, if you set the rate for `schedule-expression` from 6 hours to an interval like 30 minutes, you can exceed the daily ssm inventory limit. This causes results to be delayed, and you might encounter Amazon EC2 instances with partial error statuses.

Supported programming languages

For Linux instances, Amazon Inspector deep inspection can produce findings for application programming language packages and operating system packages.

For Mac and Windows instances, Amazon Inspector deep inspection can produce findings only for operating system packages.

For more information about supported programming languages, see Supported programming languages: Amazon EC2 deep inspection.

Scanning Windows EC2 instances with Amazon Inspector

> **Note**
>
> This page applies to customers that have not opted in to Enhanced EC2 Scanning.

Amazon Inspector automatically discovers all supported Windows instances and includes them in continuous scanning without any extra actions. For information about which instances are supported, see Operating systems and programming languages supported by Amazon Inspector. Amazon Inspector runs Windows scans at regular intervals. Windows instances are scanned at discovery and then every 6 hours. However, you can adjust the default scan interval after the first scan.

When Amazon EC2 scanning is activated, Amazon Inspector creates the following SSM associations for your Windows resources: `InspectorDistributor-do-not-delete`, `InspectorInventoryCollection-do-not-delete`, and `InvokeInspectorSsmPlugin-do-not-delete`. To install the Amazon Inspector SSM plugin on your Windows instances, the `InspectorDistributor-do-not-delete` SSM association uses the `AWS-ConfigureAWSPackage` SSM document and the `AmazonInspector2-InspectorSsmPlugin` SSM Distributor package. For more information, see The Amazon Inspector SSM plugin for Windows. To collect instance data and generate Amazon Inspector findings, the `InvokeInspectorSsmPlugin-do-not-delete` SSM association runs the Amazon Inspector SSM plugin at 6-hour intervals. However, you can customize this setting using a cron or rate expression.

> **Note**
>
> Amazon Inspector stages updated Open Vulnerability and Assessment Language (OVAL) definition files to the S3 bucket `inspector2-oval-prod-`*`your-AWS-Region`*. The Amazon S3 bucket contains OVAL definitions used in scans. These OVAL definitions shouldn't be modified. Otherwise, Amazon Inspector won't scan for new CVEs when they release.

Amazon Inspector scan requirements for Windows instances

To scan a Windows instance, Amazon Inspector requires the instance to meet the following criteria:

- The instance is an SSM managed instance. For instructions about setting up your instance for scanning, see Configuring the SSM Agent.
- The instance operating system is one of the supported Windows operating systems. For a complete list of supported operating systems, see Amazon EC2 instances status values.
- The instance has the Amazon Inspector SSM plugin installed. Amazon Inspector automatically installs the Amazon Inspector SSM plugin for managed instances upon discovery. See the next topic for details about the plugin.

> **Note**
>
> If your host is running in an Amazon VPC without outgoing internet access, Windows scanning requires your host to be able to access Regional Amazon S3 endpoints. To learn how to configure an Amazon S3 Amazon VPC endpoint, see Create a gateway endpoint in the *Amazon Virtual Private Cloud User Guide*. If your Amazon VPC endpoint policy is restricting access to external S3 buckets, you must specifically allow access to the bucket maintained by Amazon Inspector in your AWS Region that stores the OVAL definitions used to evaluate your instance. This bucket has the following the format: `inspector2-oval-prod-REGION`.

Setting custom schedules for Windows instance scans

You can customize the time between your Windows Amazon EC2 instance scans by setting a cron expression or rate expression for the `InvokeInspectorSsmPlugin-do-not-delete` association using SSM. For more information, see Reference: Cron and rate expressions for Systems Manager in the *AWS Systems Manager User Guide* or use the following instructions.

Select from the following code examples to change the scan cadence for Windows instances from the default 6 hours to 12 hours using either a rate expression or a cron expression.

The following examples require you to use the **AssociationId** for the association named `InvokeInspectorSsmPlugin-do-not-delete`. You can retrieve your **AssociationId** by running the following AWS CLI command:

```
$ aws ssm list-associations --association-filter-list
 "key=AssociationName,value=InvokeInspectorSsmPlugin-do-not-delete" --region us-east-1
```

> **Note**
>
> The **AssociationId** is Regional, so you need to first retrieve a unique ID for each AWS Region. You can then run the command to change the scan cadence in each Region where you want to set a custom scan schedule for Windows instances.

Example rate expression

```
$ aws ssm update-association \
--association-id "YourAssociationId" \
--association-name "InvokeInspectorSsmPlugin-do-not-delete" \
--schedule-expression "rate(12 hours)"
```

Example cron expression

```
$ aws ssm update-association \
--association-id "YourAssociationId" \
--association-name "InvokeInspectorSsmPlugin-do-not-delete" \
--schedule-expression "cron(0 0/12 * * ? *)"
```

Scanning machine images with Amazon Inspector

Amazon Inspector scans the machine images that your account owns for operating system and programming language package vulnerabilities. A machine image is an Amazon Machine Image (AMI) that you use to launch EC2 instances. When Amazon Inspector detects a vulnerability in a machine image, it produces a **Package Vulnerability** type finding with a **Resource type** of `MACHINE_IMAGE`. Scanning your machine images lets you find and remediate vulnerabilities before you use an image to launch instances.

For each machine image, Amazon Inspector collects a software inventory from all of the Amazon EBS snapshots that back the image, and then evaluates that inventory against vulnerability data that Amazon Inspector sources from more than 50 data feeds. Because Amazon Inspector reads the snapshots that back an image, you don't need to launch an instance from a machine image for Amazon Inspector to scan it.

By default, machine image scanning is continuous. When you activate machine image scanning for an account, Amazon Inspector discovers the machine images that your account owns, scans all of

the eligible ones, and then keeps monitoring them for as long as they remain in scope. Amazon Inspector scans new machine images as you create, copy, register, or enable them, and rescans in-scope images whenever Amazon Inspector adds a new common vulnerabilities and exposures (CVE) item to its database that's relevant to that image. You don't need to request a rescan.

Exclusion mode, which scans all eligible machine images except the ones you tag for exclusion, is the default scan mode. If you only want Amazon Inspector to scan a targeted subset of your machine images, you can change the scan mode to inclusion mode and tag the images you want scanned. Amazon Inspector scans continuously in either mode; the scan mode controls which images are in scope, not how often Amazon Inspector scans them. For more information, see Managing machine image scan mode.

If you don't want continuous scanning, you can scan a single machine image on demand instead. An on-demand scan is a one-time scan that you request for a specific machine image, and it doesn't require you to activate Amazon Inspector. On-demand scans ignore your machine image scan configuration, and Amazon Inspector delivers the results only through Amazon EventBridge. For more information, see Running on-demand machine image scans with Amazon Inspector.

For information about how to activate machine image scanning, see Activating a scan type. For information about how to control which machine images Amazon Inspector scans, see Configuring machine image scanning. For information about how to view findings, see Viewing Amazon Inspector findings.

Note

Amazon Inspector only scans machine images that your account owns. Amazon Inspector doesn't scan a machine image that is not owned by you. To scan a shared or public machine image, copy it into your own account first. Copying an image creates new snapshots in your account, and Amazon Inspector scans the copy.

Note

Machine image scanning is available in the commercial AWS Regions and the AWS GovCloud (US) Regions. Continuous machine image scanning and on-demand machine image scans aren't available in the China Regions. For more information, see Region-specific feature availability.

Topics

- Eligible machine images
- Scan behaviors for machine image scanning
- Configuring machine image scanning
- Running on-demand machine image scans with Amazon Inspector

Eligible machine images

Amazon Inspector scans a machine image if the image meets all of the following conditions:

- Your account owns the machine image and the Amazon EBS snapshots that back it.
- The machine image is backed by Amazon EBS and has a state of `available`.
- The machine image runs a supported operating system. For more information, see Supported operating systems: machine image scanning.
- The machine image runs a single operating system. Amazon Inspector doesn't support scanning machine images that contain multiple operating systems.
- The snapshots that back the machine image use one of the following file system formats:
  - `ext3`
  - `ext4`
  - `ntfs`
  - `xfs`

  > **Note**
  >
  > Amazon Inspector doesn't support Logical Volume Manager (LVM). If any snapshot that backs the machine image uses an LVM volume, Amazon Inspector can't scan the image and reports a scan status of `UNSUPPORTED_OS`.

- The machine image is within your scan eligibility windows, and it's in scope based on your scan mode and exclusion options. For more information, see Configuring machine image scanning.

> **Note**
>
> If Amazon Inspector can't scan a machine image, Amazon Inspector reports a terminal scan status for the image, such as `UNSUPPORTED_OS` or `STORAGE_LIMIT_EXCEEDED`. For more information, see Scan status values for machine images.

Scan behaviors for machine image scanning

When you first activate machine image scanning, Amazon Inspector discovers the machine images that your account owns and scans the images that are within your scan eligibility windows. Amazon Inspector then continues to monitor each in-scope image.

Amazon Inspector initiates a new scan of a machine image in the following situations:

- Whenever you create, copy, or register a machine image.
- Whenever you enable a machine image that Amazon Inspector hasn't previously scanned.
- Whenever Amazon Inspector adds a new CVE item to its database, and that CVE is relevant to a machine image that's in scope.
- Whenever you change your scan configuration in a way that brings a machine image into scope. For more information, see How Amazon Inspector applies a configuration change.

Amazon Inspector keeps your scanning scope current by monitoring Amazon EC2 machine image lifecycle events. The following table describes how Amazon Inspector responds to each event.

How Amazon Inspector responds to Amazon EC2 machine image lifecycle events

Event	Amazon Inspector behavior
`CreateImage`	Amazon Inspector scans the new machine image and all of the snapshots that back it.
`CopyImage`	Amazon Inspector treats the copy as a new machine image and scans it, because copying an image creates new snapshots in your account. You can exclude copied images from scanning. For more information, see Excluding copied and backup machine images.

Event	Amazon Inspector behavior
`RegisterImage`	Amazon Inspector treats the registered machine image as a new image and scans it, along with all of the snapshots that back it.
`EnableImage`	If Amazon Inspector previously scanned the machine image, Amazon Inspector doesn't rescan it. Otherwise, Amazon Inspector treats the image as a new image and scans it.
`DisableImage`	Amazon Inspector takes no action. Disabling a machine image only prevents Amazon EC2 from launching instances from it, and the image still exists in your account.
`DeregisterImage`	Amazon Inspector removes the machine image from your scanning scope, because the image no longer exists in your account.
Tag change events	Amazon Inspector re-evaluates the machine image against your scan mode. Depending on your scan mode, adding or removing the configured tag either brings the image into scope or removes it. For more information, see Managing machine image scan mode.

You can check when Amazon Inspector last scanned a machine image for vulnerabilities from the **Machine images** tab on the **Account management** page, or by using the ListCoverage API operation.

Scan status values for machine images

The following table lists the scan status values that Amazon Inspector reports for a machine image.

Machine image scan status values

Status	Description
`PENDING_INITIAL_SCAN`	Amazon Inspector discovered the machine image and is collecting a software inventory from its snapshots.

Status	Description
ACTIVE	Amazon Inspector scanned the machine image and continues to monitor it for new CVEs while the image remains in scope.
ACCESS_DENIED	Amazon Inspector couldn't access the machine image or the snapshots that back it. This can occur if a resource policy or a AWS KMS key policy prevents Amazon Inspector from reading the snapshot data.
UNSUPPORTED_OS	The operating system of the machine image isn't supported for scanning, or one or more of the snapshots that back the image uses a configuration that Amazon Inspector can't read, such as Logical Volume Manager (LVM). For more information, see Supported operating systems: machine image scanning.
COLLECTION_TIME_LIMIT_EXCEEDED	Amazon Inspector timed out while collecting the software inventory from the snapshots that back the machine image.
STORAGE_LIMIT_EXCEEDED	The combined size of the snapshots that back the machine image exceeds the limit that Amazon Inspector supports for scanning.
INTERNAL_ERROR	Amazon Inspector encountered an internal error while scanning the machine image. Amazon Inspector retries the scan automatically.

Configuring machine image scanning

Your scan configuration determines which of the machine images that your account owns are in scope. It has three parts, which Amazon Inspector applies in this order:

1. **Scan eligibility windows** – How recently a machine image must have been created, or had an instance launched from it, to be in scope. For more information, see Setting scan eligibility windows.
2. **Exclusion options** – Whether to exclude copied machine images and backup machine images. To exclude other individual machine images, use an exclusion tag in exclusion mode. For more information, see Excluding copied and backup machine images.

3. **Scan mode** – Whether Amazon Inspector scans every eligible image except the ones you tag (exclusion mode), or only the ones you tag (inclusion mode). For more information, see Managing machine image scan mode.

A machine image is in scope only if it satisfies all three parts. Amazon Inspector excludes a machine image if any exclusion condition applies.

You configure machine image scanning by using the UpdateConfiguration API operation, or from the machine image scanning settings under **General settings** in the Amazon Inspector console. To view your current configuration, use the GetConfiguration API operation. Amazon Inspector applies default values for every setting when you activate machine image scanning for an account. The default configuration is the following:

- **Created within** and **Last launched within** – 14 days each.
- Exclusion options – Copied machine images and backup machine images are excluded.
- Scan mode – Exclusion mode (`SCAN_ALL`), with no exclusion tag.

> **Note**
>
> If you're the delegated administrator for an organization, your machine image scan configuration propagates to all member accounts in your organization that have machine image scanning activated. Member accounts can't configure machine image scanning for themselves. If a member account calls UpdateConfiguration, Amazon Inspector returns an `AccessDeniedException`. For more information, see Managing multiple accounts in Amazon Inspector with AWS Organizations.

Topics

- Setting scan eligibility windows
- Excluding copied and backup machine images
- Managing machine image scan mode
- How Amazon Inspector applies a configuration change
- Configuring machine image scan settings

Setting scan eligibility windows

Scan eligibility windows determine how long Amazon Inspector continues to monitor a machine image. Amazon Inspector provides two independent windows:

Created within

A machine image is eligible if you created it within this window.

Last launched within

A machine image is eligible if you launched an EC2 instance from it within this window.

A machine image is eligible if *either* window is satisfied. This means an older machine image stays in scope as long as you're still launching instances from it, and a newly created image is in scope even if you haven't launched an instance from it yet.

The following durations are available for each window:

- 7 days
- 14 days (default)
- 30 days
- 60 days
- 90 days
- 180 days

When a machine image falls outside both windows, Amazon Inspector stops monitoring it and closes its findings. A deprecated image stays eligible for rescans as long as it satisfies either window. Amazon Inspector checks your images against the windows once a day. As a result, a change in scope can take up to 24 hours to take effect.

> **Important**
>
> Lengthening a scan eligibility window increases the number of machine images that Amazon Inspector scans, which increases your Amazon Inspector costs. Amazon Inspector meters machine image scanning per scan. For more information, see Monitoring Usage and Cost in Amazon Inspector and Amazon Inspector pricing.

Excluding copied and backup machine images

You can exclude two categories of machine image from scanning. These exclusions apply in both scan modes. Amazon Inspector applies them before your scan mode tag filter. To exclude other individual images, add an exclusion tag to them and use exclusion (SCAN_ALL) mode. For more information, see Managing machine image scan mode.

Copied machine images

Excludes machine images that are copies of other images. Amazon Inspector identifies a copy from the description that Amazon EC2 adds to the copied image and its snapshots. Use this option when you already scan the image in the source account and don't want to pay to scan the copy too. It's turned on by default.

Backup machine images

Excludes machine images that Amazon Data Lifecycle Manager or AWS Backup created. Amazon Inspector identifies these images by the tags that AWS applies to them: `aws:dlm:lifecycle-policy-id` or `aws:backup:source-resource`. Use this option when you don't want automated backup images to increase your scan volume. It's turned on by default. For more information about Data Lifecycle Manager, see Amazon Data Lifecycle Manager in the *Amazon Elastic Compute Cloud User Guide*.

> **Note**
>
> You're not charged for excluded machine images.

Managing machine image scan mode

The scan mode determines which of your eligible machine images Amazon Inspector keeps in scope. It also sets whether the tag you configure adds images to that scope or removes them. Amazon Inspector scans continuously in both modes, and supports the following modes.

Exclusion mode (SCAN_ALL) – default

Amazon Inspector continuously scans every eligible machine image except the ones that carry the tag you configure. In this mode, the tag is an *exclusion* tag. Use this mode when you want broad coverage and only need to opt specific images out.

Inclusion mode (`TARGET_SCANNING`)

Amazon Inspector continuously scans only the eligible machine images that carry the tag you configure, and excludes all others. In this mode, the tag is an *inclusion* tag. Use this mode when you want to limit scanning to a targeted subset of your images, such as production images only.

You choose the tag that Amazon Inspector matches on, so you can reuse a tag that's already part of your tagging strategy. Note the following about the tag:

- A tag is required for inclusion (`TARGET_SCANNING`) mode. If you set this mode without a tag, Amazon Inspector returns a `ValidationException`. In exclusion (`SCAN_ALL`) mode, a tag is optional. If you don't provide one, Amazon Inspector scans every eligible machine image.
- You can configure one tag. If you provide more than one, Amazon Inspector returns a `ValidationException`.
- A tag value is optional. If you specify a key without a value, Amazon Inspector matches any machine image that has that key, regardless of its value. If you specify both a key and a value, Amazon Inspector matches only machine images that have that exact key-value pair.

For information about adding tags to a machine image, see Tag your Amazon EC2 resources in the *Amazon Elastic Compute Cloud User Guide*.

When you add or remove the configured tag on a machine image, Amazon Inspector re-evaluates that image. In exclusion (`SCAN_ALL`) mode, adding the tag removes the image from your scanning scope, and removing the tag brings it back in. In inclusion (`TARGET_SCANNING`) mode, adding the tag brings the image into scope, and removing the tag removes it. In both modes, the exclusion options still apply, so an image that you tag for inclusion stays out of scope if it's also a copied or backup image that you chose to exclude.

Important

Before you switch from exclusion mode to inclusion mode, tag the machine images that you want Amazon Inspector to keep scanning. When you switch modes, Amazon Inspector immediately removes every untagged image from your scanning scope and closes the findings for those images.

How Amazon Inspector applies a configuration change

When you change your scan configuration, Amazon Inspector re-evaluates every machine image in the account right away. Amazon Inspector then does the following:

- Starts scanning machine images that the change brought into scope.
- Stops monitoring machine images that the change removed from scope, and closes the findings for those images.

Amazon Inspector processes one configuration change for an account at a time. If you call UpdateConfiguration while Amazon Inspector is still applying a previous change, Amazon Inspector returns a `ValidationException`. Retry the request after the previous change finishes. If you submit a configuration that matches your current configuration, Amazon Inspector treats the request as a no-op.

Separately from configuration changes, Amazon Inspector re-evaluates your machine images against your scan eligibility windows once a day. For more information, see Setting scan eligibility windows.

> **Note**
>
> If you're the delegated administrator for an organization, Amazon Inspector applies your change to your own account and then propagates it to each member account that has machine image scanning activated. Each member account is re-evaluated independently, so the change can take effect at different times across your organization.

Configuring machine image scan settings

The following procedure describes how to configure machine image scanning. To complete this procedure for a multi-account environment, follow these steps while signed in as the Amazon Inspector delegated administrator.

Console

To configure machine image scanning

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.

2. Select the AWS Region where you want to configure machine image scanning.
3. From the navigation pane, choose **General settings**, and then choose **Machine image scanning settings**.
4. Under **Scan eligibility**, choose a duration for **Created within** and a duration for **Last launched within**.
5. Under **Scan mode**, choose **Exclusion mode** or **Inclusion mode**, and then enter the tag key and optional tag value that Amazon Inspector matches on. A tag is required for **Inclusion mode**. (Optional) Select or clear **Exclude copied machine images** and **Exclude backup machine images**; both are selected by default.
6. Choose **Save**.
7. (Recommended) Repeat these steps in each AWS Region where you activated machine image scanning.

API

Run the UpdateConfiguration API operation. In the request, provide an `amiConfiguration` object. Any field that you omit keeps its current value.

The following example continuously scans all eligible machine images except the ones tagged with `InspectorExclude=true`, and excludes copied machine images.

```
{
    "amiConfiguration": {
        "createdWithin": "180_DAYS",
        "lastLaunchedWithin": "90_DAYS",
        "scanModeConfiguration": {
            "scanMode": "SCAN_ALL",
            "tags": [
                {
                    "key": "InspectorExclude",
                    "value": "true"
                }
            ],
            "excludeCopiedAmis": true,
            "excludeBackupDlmAmis": false
        }
    }
}
```

The following example continuously scans only the eligible machine images that have the `InspectorScan` tag key, regardless of the tag value, and excludes backup machine images.

```
{
    "amiConfiguration": {
        "scanModeConfiguration": {
            "scanMode": "TARGET_SCANNING",
            "tags": [
                {
                    "key": "InspectorScan"
                }
            ],
            "excludeCopiedAmis": false,
            "excludeBackupDlmAmis": true
        }
    }
}
```

Running on-demand machine image scans with Amazon Inspector

An on-demand machine image scan is a one-time scan of a single machine image that you request. You don't need to activate Amazon Inspector to run one. Use an on-demand scan to scan a single image at a point in time, such as an image a build pipeline just produced. The image doesn't enter a continuous scanning scope.

You start an on-demand scan with the StartOnDemandScan API operation. Amazon Inspector returns a `scanId` that identifies the scan, collects a software inventory from the Amazon EBS snapshots that back the machine image, evaluates that inventory against Amazon Inspector vulnerability data, and then publishes the results to Amazon EventBridge. You can also start an on-demand scan from the Amazon EC2 console, which calls StartOnDemandScan on your behalf.

Important

Amazon Inspector delivers the findings from an on-demand scan only through EventBridge. On-demand scan findings don't appear in the Amazon Inspector console, aren't returned by the ListFindings API operation, and aren't sent to AWS Security Hub CSPM. To view

the findings for a machine image in a console, use the Amazon EC2 console. If you also activated continuous machine image scanning and the image is in scope, its continuous-scan findings still appear in these places. For more information, see Receiving on-demand scan results with Amazon EventBridge.

The following table compares on-demand scanning with continuous machine image scanning.

On-demand compared with continuous machine image scanning

	On-demand scanning	Continuous scanning
How a scan starts	You call StartOnDemandScan for a specific machine image.	Amazon Inspector scans machine images automatically when you create, copy, register, or enable an image, and when a relevant new CVE is published, as long as the image is within your scan eligibility windows. For more information, see Setting scan eligibility windows.
Activation required	No. You don't need to activate Amazon Inspector.	Yes. You activate machine image scanning for the account.
Scan configuration	Not applied. On-demand scans ignore scan eligibility windows, scan mode, and exclusion options.	Applied. For more information, see Configuring machine image scanning.
Rescans	None. Each scan is a one-time evaluation. Call StartOnDemandScan again to rescan.	Amazon Inspector rescans in-scope images automatically. For more information about which images are in scope, see Eligible machine images.
Where results go	EventBridge only.	The same places as your other Amazon Inspector findings. For more information, see Viewing Amazon Inspector findings.

	On-demand scanning	Continuous scanning
Free trial	Not eligible.	Eligible. For more information, see About the Amazon Inspector free trial.

> **Note**
>
> On-demand scanning is independent of continuous machine image scanning. If you activated machine image scanning for your account, you can still run on-demand scans, including for machine images that your scan configuration excludes. Amazon Inspector meters an on-demand scan separately from any continuous scan of the same image.

Topics

- Requirements for on-demand machine image scans
- Starting an on-demand machine image scan
- Scan status values for on-demand machine image scans
- Receiving on-demand scan results with Amazon EventBridge
- Pricing for on-demand machine image scans

Requirements for on-demand machine image scans

Amazon Inspector runs an on-demand scan only if the following conditions are met:

- The IAM identity that starts the scan has permission for both the `inspector2:StartOnDemandScan` action and the `iam:CreateServiceLinkedRole` action. Amazon Inspector needs `iam:CreateServiceLinkedRole` to create the service-linked roles that it uses to read your snapshot data. For more information, see Service-linked role permissions in the *IAM User Guide*.
- Your account owns the machine image and the Amazon EBS snapshots that back it. If the image belongs to another account, Amazon Inspector returns an `AccessDeniedException`. To scan an image that another account shared with you, or a public image, first copy it into your own account. Then scan the copy.

- No other on-demand scan is already in progress for the same machine image. Amazon Inspector runs one on-demand scan per image at a time, but you can scan different images at the same time. If a scan is already in progress, Amazon Inspector returns a `ConflictException`. Use GetOnDemandScanStatus to check whether the in-progress scan finished before you retry.
- The machine image meets the same eligibility requirements that apply to continuous scanning, including a supported operating system and a supported file system format on every snapshot that backs the image. For more information, see Eligible machine images and Supported operating systems: machine image scanning.

When you start your first on-demand scan, Amazon Inspector creates the service-linked roles that it needs to read your snapshot data. While an on-demand scan is in progress for your account, Amazon Inspector prevents you from deleting those service-linked roles so that the scan can finish. After all of your on-demand scans finish, you can delete the roles. The roles remain in your account until you delete them. For more information, see Using service-linked roles for Amazon Inspector.

Starting an on-demand machine image scan

The following procedure describes how to start an on-demand scan of a machine image.

API

Run the StartOnDemandScan API operation. In the request, provide the ID of the machine image as `resourceId`, and set `resourceType` to `AMI`.

```
{
    "resourceId": "ami-0abcdef1234567890",
    "resourceType": "AMI"
}
```

Amazon Inspector returns a `scanId` that uniquely identifies the scan. Store the `scanId`. You use it to check the status of the scan, and Amazon Inspector includes it in every EventBridge notification for the scan. If you don't have the `scanId`, you can find it by using the ListOnDemandScans API operation.

```
{
```

```
    "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d"
}
```

Amazon EC2 console

To start an on-demand machine image scan

1. Open the Amazon EC2 console.
2. Select the AWS Region that contains the machine image that you want to scan.
3. From the navigation pane, choose **AMIs**, and then select the machine image that you want to scan.
4. Choose **Actions**, and then choose **Scan for vulnerabilities**.

To check the status of a scan, use the GetOnDemandScanStatus API operation with the `scanId` and the machine image ID. To list the on-demand scans that Amazon Inspector still retains for a machine image, use the ListOnDemandScans API operation. Amazon Inspector returns the scans for the machine image with the most recent scan first, and paginates the results with `maxResults` and `nextToken`.

> **Note**
>
> Amazon Inspector retains the record of an on-demand scan, including its status and the manifest that Amazon Inspector collected, for 30 days after the scan completes. After 30 days, GetOnDemandScanStatus returns a `ResourceNotFoundException` for that `scanId`, and the scan no longer appears in ListOnDemandScans results.

Scan status values for on-demand machine image scans

GetOnDemandScanStatus returns a `scanStatus` for the scan, and a `scanStatusReason` when the scan failed. These values are specific to on-demand scans and differ from the scan status values that Amazon Inspector reports for continuously scanned machine images. For the continuous scanning values, see Scan status values for machine images.

On-demand machine image scan status values

Scan status	Scan status reason	Description
IN_PROGRESS	–	Amazon Inspector started the scan and is collecting the software inventory or evaluating it for vulnerabilities.
SUCCESSFUL	–	Amazon Inspector finished the scan and published the findings to EventBridge.
FAILED	ACCESS_DENIED	Amazon Inspector couldn't access the machine image or the snapshots that back it. This can occur if a resource policy or a AWS KMS key policy prevents Amazon Inspector from reading the snapshot data.
FAILED	UNSUPPORTED_OS	The operating system of the machine image isn't supported for scanning, or one or more of the snapshots that back the image uses a configuration that Amazon Inspector can't read, such as Logical Volume Manager (LVM). For more informati on, see Supported operating systems: machine image scanning.
FAILED	STORAGE_LIMIT_EXCE EDED	The combined size of the snapshots that back the machine image exceeds the limit that Amazon Inspector supports for scanning. Amazon Inspector can't scan this machine image.
FAILED	EXTRACTION_COLLECT ION_TIME_LIMIT_EXC EEDED	Amazon Inspector timed out while collecting the software inventory from the snapshots that back the machine image.
FAILED	RESOURCE_TERMINATE D	You deregistered the machine image, or deleted a snapshot that backs it, while the scan was in progress.

Scan status	Scan status reason	Description
FAILED	INTERNAL_ERROR	Amazon Inspector encountered an internal error while scanning the machine image. Call StartOnDemandScan again to retry the scan.

Receiving on-demand scan results with Amazon EventBridge

Amazon Inspector publishes two kinds of EventBridge events for an on-demand machine image scan:

Findings events

Amazon Inspector publishes one event for each vulnerability that it detects in the machine image. The `detail-type` field is set to `Inspector2 OnDemand Finding`. These events identify the machine image by its ID and don't include other machine image metadata.

Scan complete events

Amazon Inspector publishes one event when the scan reaches a terminal state. The `detail-type` field is set to `Inspector2 OnDemand Scan Complete`. A successful event includes a `findingSeverityCounts` object. A failed event includes a `scanStatusReason` field instead.

Every event includes the `scanId` of the scan, so you can correlate findings events with the scan complete event for the same scan. To receive these events, create an EventBridge rule in the same AWS Region and account as the scan. For more information, see Amazon EventBridge event schema for Amazon Inspector events and Creating rules that react to events in the *Amazon EventBridge User Guide*.

The following example shows the scan complete event for a successful on-demand scan.

```
{
    "version": "0",
    "id": "7bf7f4b1-9f6c-4d3e-8a2b-5c1d0e9f3a44",
    "detail-type": "Inspector2 OnDemand Scan Complete",
    "source": "aws.inspector2",
    "account": "111122223333",
```

```
    "time": "2026-06-12T01:30:00Z",
    "region": "us-east-1",
    "resources": [
        "ami-0abcdef1234567890"
    ],
    "detail": {
        "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d",
        "awsAccountId": "111122223333",
        "resourceId": "ami-0abcdef1234567890",
        "resourceType": "MACHINE_IMAGE",
        "scanStatus": "SUCCESSFUL",
        "startedAt": "2026-06-12T00:00:00.000Z",
        "completedAt": "2026-06-12T01:30:00.000Z",
        "findingSeverityCounts": {
            "CRITICAL": 0,
            "HIGH": 2,
            "MEDIUM": 5,
            "LOW": 1,
            "INFORMATIONAL": 0,
            "UNTRIAGED": 0,
            "TOTAL": 8
        },
        "version": "1.0"
    }
}
```


The following example shows the scan complete event for a failed on-demand scan.


```
{
    "version": "0",
    "id": "3c2e9a17-8b4d-4f61-9a0c-2d7e6f1b8c33",
    "detail-type": "Inspector2 OnDemand Scan Complete",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2026-06-12T01:30:00Z",
    "region": "us-east-1",
    "resources": [
        "ami-0abcdef1234567890"
    ],
    "detail": {
        "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d",
```

```
        "awsAccountId": "111122223333",
        "resourceId": "ami-0abcdef1234567890",
        "resourceType": "MACHINE_IMAGE",
        "scanStatus": "FAILED",
        "scanStatusReason": "UNSUPPORTED_OS",
        "startedAt": "2026-06-12T00:00:00.000Z",
        "completedAt": "2026-06-12T01:30:00.000Z",
        "version": "1.0"
    }
}
```

The following example shows a findings event for a package vulnerability that Amazon Inspector detected in a machine image.

```
{
    "source": "aws.inspector2",
    "detail-type": "Inspector2 OnDemand Finding",
    "detail": {
        "findingArn": "arn:aws:inspector2:us-east-1:111122223333:finding/
ondemand/9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d/cea7c73a3d57d9c142992e845644d317",
        "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d",
        "awsAccountId": "111122223333",
        "type": "PACKAGE_VULNERABILITY",
        "title": "CVE-2022-3171 - com.google.protobuf:protobuf-java",
        "description": "A parsing issue with binary data in protobuf-java...",
        "severity": "HIGH",
        "status": "ACTIVE",
        "firstObservedAt": "2026-06-12T01:30:00.000Z",
        "lastObservedAt": "2026-06-12T01:30:00.000Z",
        "updatedAt": "2026-06-12T01:30:00.000Z",
        "resources": [
            {
                "type": "MACHINE_IMAGE",
                "id": "ami-0abcdef1234567890",
                "partition": "aws",
                "region": "us-east-1"
            }
        ],
        "inspectorScore": 7.5,
        "inspectorScoreDetails": {
            "adjustedCvss": {
```

```
                "scoreSource": "NVD",
                "cvssSource": "NVD",
                "version": "3.1",
                "score": 7.5,
                "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H",
                "adjustments": []
            }
        },
        "packageVulnerabilityDetails": {
            "vulnerabilityId": "CVE-2022-3171",
            "vulnerablePackages": [
                {
                    "name": "com.google.protobuf:protobuf-java",
                    "version": "2.5.0",
                    "epoch": 0,
                    "packageManager": "JAVA",
                    "filePath": "/usr/lib/jars/protobuf-java-2.5.0.jar",
                    "fixedInVersion": "3.16.3"
                }
            ],
            "source": "NVD",
            "sourceUrl": "https://nvd.nist.gov/vuln/detail/CVE-2022-3171",
            "vendorSeverity": "high"
        },
        "remediation": {
            "recommendation": {
                "text": "Update com.google.protobuf:protobuf-java to 3.16.3 or later"
            }
        },
        "fixAvailable": "YES",
        "exploitAvailable": "NO"
    }
}
```


Note

EventBridge delivers events at least once, so your rule can receive the same event more than once. Use the `findingArn` field to deduplicate findings events. If a rule is misconfigured, a target is throttled, or a target lacks the permissions that EventBridge needs, an event can also be dropped. Confirm that you received a scan complete event for

every `scanId` that you started, and use GetOnDemandScanStatus to reconcile any scan that you don't have an event for.

Pricing for on-demand machine image scans

Amazon Inspector meters each on-demand machine image scan individually. Amazon Inspector meters a scan only after it reaches a SUCCESSFUL status, so you're not charged for a scan that fails. Each call to StartOnDemandScan that completes successfully is a separate metered scan, even if you scan the same machine image more than once.

Important

On-demand machine image scans aren't included in the Amazon Inspector free trial. Amazon Inspector meters on-demand scans from your first scan, whether or not you activated Amazon Inspector for the account. For more information, see Monitoring Usage and Cost in Amazon Inspector and Amazon Inspector pricing.

Scanning Amazon Elastic Container Registry container images with Amazon Inspector

Amazon Inspector scans container images stored in Amazon Elastic Container Registry for software vulnerabilities to generate package vulnerability findings. When you activate Amazon ECR scanning, you set Amazon Inspector as the preferred scanning service for your private registry.

Note

Amazon ECR uses a registry policy to grant permissions to an AWS principal. This principal has the required permissions to call Amazon Inspector APIs for scanning. When setting the scope of your registry policy, you must not add the `ecr:*` action or `PutRegistryScanningConfiguration` in deny. This results in errors at the registry level when enabling and disabling scanning for Amazon ECR.

With basic scanning, you can configure your repositories to scan on push or perform manual scans. With enhanced scanning, you scan for operating system and programming language package

vulnerabilities at the registry level. For a side-by-side comparison of the differences between basic and enhanced scanning, see the Amazon Inspector FAQ.

> **Note**
>
> Basic scanning is provided and billed through Amazon ECR. For more information, see Amazon Elastic Container Registry pricing. Enhanced scanning is provided and billed through Amazon Inspector. For more information, see Amazon Inspector pricing.

For information about how to activate Amazon ECR scanning, see Activating a scan type. For information about how to view findings, see Viewing Amazon Inspector findings. For information about how to view findings within Amazon ECR at the image level, see Image scanning in the *Amazon Elastic Container Registry User Guide*. You can manage findings using AWS services not available for basic scanning, like AWS Security Hub CSPM and Amazon EventBridge.

You can view the scan configuration for each repository in Amazon Inspector through coverage pages and APIs. However, the configuration settings for basic scanning versus continuous scanning can only be modified in Amazon ECR. Amazon Inspector provides visibility into these settings but does not offer direct modification capabilities. For more information, see Scan images for software vulnerabilities in Amazon ECR in the *Amazon ECR User Guide*.

This section provides information about Amazon ECR scanning and describes how to configure enhanced scanning for Amazon ECR repositories.

Scan behaviors for Amazon ECR scanning

When you first activate Amazon ECR scanning, Amazon Inspector detects images pushed within the last 7 days. Amazon Inspector then scans the images and sets the scan statuses to `ACTIVE`. Amazon Inspector only scans images active in ECR (`imageStatus` field is `ACTIVE`). Images with Archived status in ECR (`imageStatus` field is `ARCHIVED`) are not scanned by Amazon Inspector.

If you enable continuous scanning, Amazon Inspector monitors your images as long as you push them within 7 days (by default), their last-in-use date is within 7 days (by default), or Amazon Inspector scans them within the configured re-scan duration. The default re-scan durations depend on when you activated Amazon ECR scanning for your account.

When Amazon ECR scanning was activated	Image push date duration	Image re-scan duration	Re-scan based on
On or after October 15, 2026	7 days	7 days	Last in use date
May 16, 2024–October 14, 2026	14 days	14 days	Last in use date
Before May 16, 2024	90 days	90 days	Last pull date

For more information, see Configuring the Amazon ECR re-scan duration.

For continuous scanning, Amazon Inspector initiates new vulnerability scans of container images in the following situations:

- Whenever a new container image is pushed.
- Whenever Amazon Inspector adds a new common vulnerabilities and exposures (CVE) item to its database, and that CVE is relevant to that container image (continuous scanning only).
- Whenever a container image is transitioned from archived to active in ECR.

If you configure your repository for on push scanning, images are only scanned when you push them.

You can check when a container image was last checked for vulnerabilities from the **Container images** tab on the **Account management** page or by using the ListCoverage API. Amazon Inspector updates the **Last scanned at** field of an Amazon ECR image in response to the following events:

- When Amazon Inspector completes an initial scan of a container image.
- When Amazon Inspector re-scans a container image because a new common vulnerabilities and exposures (CVE) item that impacts that container image was added to the Amazon Inspector database.

Archived ECR container images

Amazon Inspector does not scan container images archived in ECR (`imageStatus` is `ARCHIVED`). When an active image in ECR is transitioned to archived, Amazon Inspector automatically closes

findings and then deletes the findings after 3 days. If an archived container image is transitioned to active in ECR, Amazon Inspector triggers a new scan.

Mapping container images to running containers

Amazon Inspector provides comprehensive container security management by mapping container images to running containers across Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS). These mappings provide insights into vulnerabilities for images on running containers.

> **Note**
>
> The managed policy `AWSReadOnlyAccess` alone does not provide sufficient permissions to view the mapping between Amazon ECR images and running containers. You need both the `AWSReadOnlyAccess` and `AWSInspector2ReadOnlyAccess` managed policies to view container image mapping information.

You can prioritize remediation efforts based on operational risks and maintain security coverage across the entire container ecosystem. You can view how many container images are currently in use and which container images were last used on an Amazon ECS or Amazon EKS cluster in the past 24 hours. You can also view how many Amazon ECS tasks and Amazon EKS pods are deployed. This information can be found in the Amazon Inspector console on the details screen for container image findings and with the `ecrImageInUseCount` and `ecrImageLastInUseAt` filters for the `FilterCriteria` data type. For new container images or accounts, it can take up to 36 hours for data to be available. Afterwards, this data is updated once every 24 hours. For more information, see Viewing Amazon Inspector findings and Viewing details for Amazon Inspector findings.

> **Note**
>
> This data is automatically sent to Amazon ECR findings when you activate Amazon ECR scanning and configure your repository for continuous scanning. Continuous scanning must be configured at the Amazon ECR repository level. For more information, see Enhanced scanning in the *Amazon Elastic Container Registry User Guide*.

You can also re-scan container images from clusters based on their last-in-use date.

This feature is also supported on Fargate with Amazon ECS and Amazon EKS.

Supported operating systems and media types

For information about supported operating systems, see Supported operating systems: Amazon ECR scanning with Amazon Inspector.

Amazon Inspector scans of Amazon ECR repositories cover the following supported media types:

Image manifest

- `"application/vnd.oci.image.manifest.v1+json"`
- `"application/vnd.docker.distribution.manifest.v2+json"`

Image configuration

- `"application/vnd.docker.container.image.v1+json"`
- `"application/vnd.oci.image.config.v1+json"`

Image layers

- `"application/vnd.docker.image.rootfs.diff.tar"`
- `"application/vnd.docker.image.rootfs.diff.tar.gzip"`
- `"application/vnd.docker.image.rootfs.foreign.diff.tar.gzip"`
- `"application/vnd.oci.image.layer.v1.tar"`
- `"application/vnd.oci.image.layer.v1.tar+gzip"`
- `"application/vnd.oci.image.layer.v1.tar+zstd"`
- `"application/vnd.oci.image.layer.nondistributable.v1.tar"`
- `"application/vnd.oci.image.layer.nondistributable.v1.tar+gzip"`

> **Note**
>
> Amazon Inspector does not support the `"application/vnd.docker.distribution.manifest.list.v2+json"` media type for the scanning of Amazon ECR repositories.

Configuring the Amazon ECR re-scan duration

The Amazon ECR re-scan duration setting determines how long Amazon Inspector continuously monitors container images in repositories. You configure the re-scan duration for the image last-in-use date, last pull date, and push date. As a best practice, configure the re-scan duration to best suit your environment.

If you build images often, choose a shorter scan duration. For images used over long periods of time, choose a longer scan duration. The default scan duration for new accounts, including new accounts added to an organization, is 7 days.

Amazon Inspector continues to monitor and re-scan an image as long as it was last in use on a cluster or pushed within 7 days (by default). If an image hasn't been pushed or used on a running container within the configured push and last-in-use dates, Amazon Inspector stops monitoring it. There is an option to change the setting to monitor images by last pull date instead of the last-in-use date, if required. When Amazon Inspector stops monitoring an image, it sets the image scan status code to inactive and reason code to expired. Amazon Inspector then schedules all associated image findings to be closed.

If you increase the push date duration, Amazon Inspector applies the change to all actively scanned images in repositories configured for continual scanning. However, inactive images remain inactive, even if you push them within the new duration.

When you configure the re-scan duration from a delegated administrator account, Amazon Inspector applies the setting to all member accounts in the organization. If the delegated administrator account does not enable Amazon ECR scanning, it cannot view clusters for an API image.

For multi-architecture images, the last-in-use date tracking is not supported. When using multi-architecture images, we recommend that you configure scanning based on image pull or push events instead of the last-in-use date to ensure proper re-scanning behavior.

> **Note**
>
> As of October 15, 2026, the default re-scan duration when you activate Amazon ECR scanning is 7 days. Accounts that already have Amazon ECR scanning activated keep their current re-scan duration settings.

Image re-scan duration

The image re-scan duration determines how long Amazon Inspector monitors images. The image re-scan duration includes two modes: **Last in use date** (default) or **Last pull date**. Choose **Last in use date** (default) if you want to use the last-in-use date from your Amazon ECS/Amazon EKS cluster activity. Choose **Last pull date** if you want to use the last pull date from your Amazon ECR images to re-scan images. The following options are available as re-scan durations:

- 3 days
- 7 days (default)
- 14 days
- 30 days
- 60 days
- 90 days
- 180 days

Image push date duration

The image push date duration determines how long Amazon Inspector will continuously monitor images after being pushed to repositories. The following options are available as re-scan durations:

- 3 days
- 7 days (default)
- 14 days
- 30 days
- 60 days
- 90 days
- 180 days
- Lifetime

To configure the Amazon ECR re-scan duration

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Select the AWS Region where you want to configure the Amazon ECR re-scan duration.
3. From the navigation pane, choose **General settings**, and then choose **ECR scanning settings**.

4. Under **ECR re-scan duration**, choose the image re-scan mode, and then choose the corresponding duration.
5. Under **Image push date**, choose the image push date.
6. Choose **Save**.

Understanding ECR container image states

Inspector only scans `ACTIVE` images in ECR container images. ECR container images in an `ARCHIVED` status are not scanned. To learn more about scanning behaviors, see Scan behaviors for Amazon ECR scanning.

When an ECR container image's image status in ECR transitions to `ACTIVE`, Inspector uses the `lastActivatedAt` field to monitor re-scan duration.

Scanning AWS Lambda functions with Amazon Inspector

Amazon Inspector support for AWS Lambda functions and layers provides continuous automated security vulnerability assessments. Amazon Inspector offers two types of Lambda function scanning:

Amazon Inspector Lambda standard scanning

This scan type is the default Lambda scan type. It scans application dependencies in Lambda functions and layers for package vulnerabilities.

Amazon Inspector Lambda code scanning

This scan type scans custom application code in your Lambda functions and layers for code vulnerabilities. You can activate Lambda standard scanning or Lambda standard scanning with Lambda code scanning.

If you want to activate Lambda code scanning, you must activate Lambda standard scanning first. For more information, see Activating a scan type.

When you activate Lambda function scanning, Amazon Inspector creates the following service-linked channels in your account: `cloudtrail:CreateServiceLinkedChannel` and `cloudtrail:DeleteServiceLinkedChannel`. Amazon Inspector manages these channels and uses them to monitor CloudTrail events for scans. The channels allow you to view CloudTrail

events in your account as if you had a trail in CloudTrail. We recommend creating your own trail in CloudTrail to manage events in your account. For information about how to view these channels, see Viewing service-linked channels in the *AWS CloudTrail User Guide*.

> **Note**
>
> Amazon Inspector does not support scanning Lambda functions encrypted with customer managed keys. This applies to Lambda standard scanning and Lambda code scanning.

Scan behaviors for Lambda function scanning

Upon activation, Amazon Inspector scans all Lambda functions invoked or updated in the last 90 days in your account. Amazon Inspector initiates vulnerability scans of Lambda functions in the following situations:

- As soon as Amazon Inspector discovers an existing Lambda function.
- When you deploy a new Lambda function to the Lambda service.
- When you deploy an update to the application code or dependencies of an existing Lambda function or its layers.
- Whenever Amazon Inspector adds a new common vulnerabilities and exposures (CVE) item to its database, and that CVE is relevant to your function.

Amazon Inspector monitors each Lambda function throughout its lifetime until it's either deleted or excluded from scanning.

You can check when a Lambda function was last checked for vulnerabilities from the **Lambda functions** tab on the **Account management** page or by using the ListCoverage API. Amazon Inspector updates the **Last scanned at** field for a Lambda function in response to the following events:

- When Amazon Inspector completes an initial scan of a Lambda function.
- When a Lambda function is updated.
- When Amazon Inspector re-scans a Lambda function because a new CVE item impacting that function was added to the Amazon Inspector database.

Supported runtimes and eligible functions

Amazon Inspector supports different runtimes for Lambda standard scanning and Lambda code scanning. For a list of supported runtimes for each scan type, see Supported runtimes: Amazon Inspector Lambda standard scanning and Supported runtimes: Amazon Inspector Lambda code scanning.

In addition to having a supported runtime, a Lambda function needs to meet the following criteria to be eligible for Amazon Inspector scans:

- The function has been invoked or updated in the last 90 days.
- The function is marked `$LATEST`.
- The function isn't excluded from scans by tags.

> **Note**
>
> Lambda functions that haven't been invoked or modified in the last 90 days are automatically excluded from scans. Amazon Inspector will resume scanning an automatically excluded function if it is invoked again or if changes are made to the Lambda function code.

Amazon Inspector Lambda standard scanning

Amazon Inspector Lambda standard scanning identifies software vulnerabilities in the application package dependencies you add to your Lambda function code and layers. For example, if your Lambda function uses a version of the `python-jwt` package with a known vulnerability, Lambda standard scanning will generate a finding for that function.

If Amazon Inspector detects a vulnerability in your Lambda function application package dependencies, Amazon Inspector produces a detailed **Package Vulnerability** type finding.

For instructions on activating a scan type see Activating a scan type.

Note

Lambda standard scanning doesn't scan the AWS SDK dependency installed by default in the Lambda runtime environment. Amazon Inspector only scans dependencies uploaded with the function code or inherited from a layer.

Note

Deactivating Amazon Inspector Lambda standard scanning will also deactivate Amazon Inspector Lambda code scanning.

Excluding functions from Lambda standard scanning

You can add tags to Lambda functions, so you can exclude them from Amazon Inspector Lambda standard scans. Excluding functions from scans can prevent unactionable alerts. When you tag a function for exclusion, the tag must have the following key-value pair.

- Key:`InspectorExclusion`
- Value:`LambdaStandardScanning`

This topic describes how to tag a function for exclusion from scans. For more information about adding tags in Lambda, see Using tags on Lambda functions.

To exclude a function from scans

1. Sign in using your credentials, and then open the Lambda console at https://console.aws.amazon.com/lambda/.
2. From the navigation pane, choose **Functions**.
3. Choose the name of the function you would want to exclude from Amazon Inspector Lambda standard scans.
4. Choose **Configuration**, and then choose **Tags**.
5. Choose **Manage tags**, and then **Add new tag**.
   a. For **Key**, enter `InspectorExclusion`.
   b. For **Value**, enter `LambdaStandardScanning`

6. Choose **Save**.

Amazon Inspector Lambda code scanning

> **Important**
>
> This feature captures snippets of Lambda functions to highlight detected vulnerabilities. These snippets can show hardcoded credentials and other sensitive materials.

With this feature, Amazon Inspector scans application code in a Lambda function for code vulnerabilities based on AWS security best practices to detect data leaks, injection flaws, missing encryption, and weak cryptography. Amazon Inspector uses automated reasoning and machine learning to evaluate your Lambda function application code. It also uses internal detectors that are developed in collaboration with Amazon Q to identify policy violations and vulnerabilities.

Amazon Inspector generates a code vulnerability when it detects a vulnerability in your Lambda function application code. This finding type includes a code snippet showing the issue and where you can find the issue in your code. It also suggests how to remediate the issue. The suggestion includes plug-and-play code blocks that you can use to replace vulnerable lines of code. These code fixes are provided in addition to general code remediation guidance for this finding type.

Code remediation suggestions is powered by automated reasoning. Some code remediation suggestions might not work as intended. You are responsible for the code remediation suggestions you adopt. Always review code remediation suggestions before adopting them. You might need to edit them to make sure your code performs as intended. For more information, see the Responsible AI Policy.

If you want to activate Lambda code scanning, you must activate Lambda standard scanning first. For more information, see Activating a scan type. For information about which AWS Regions support this feature, see Region-specific feature availability.

Encrypting your code in code vulnerability findings

Amazon Q stores code snippets that are detected to be in connection with a code vulnerability finding using Lambda code scanning. By default, Amazon Q controls the AWS owned key used to encrypt your code. However, you can use your own customer managed key for encryption through the Amazon Inspector API. For more information, see Encryption at rest for code in your findings.

Excluding functions from Lambda code scanning

You can add tags to Lambda functions, so you can exclude them from Amazon Inspector Lambda code scans. Excluding functions from scans can prevent unactionable alerts. When you tag a function for exclusion, the tag must have the following key-value pair.

- Key – `InspectorCodeExclusion`
- Value – `LambdaCodeScanning`

This topic describes how to tag a function for exclusion from code scans. For more information about adding tags in Lambda, see Using tags on Lambda functions.

To exclude a function from code scans

1. Sign in using your credentials, and then open the Lambda console at https://console.aws.amazon.com/lambda/.
2. From the navigation pane, choose **Functions**.
3. Choose the name of the function you would want to exclude from Amazon Inspector Lambda code scans.
4. Choose **Configuration**, and then choose **Tags**.
5. Choose **Manage tags**, and then **Add new tag**.
   a. For **Key**, enter `InspectorCodeExclusion`.
   b. For **Value**, enter `LambdaCodeScanning`
6. Choose **Save**.

Deactivating a scan type in Amazon Inspector

When you deactivate a scan type, you lose access to any findings the scan type produced. If you reactivate the scan type, Amazon Inspector scans all eligible resources to generate new findings. If you want to keep a record of your findings, you can export them to an Amazon Simple Storage Service (Amazon S3) bucket as a findings report. For more information, see Exporting Amazon Inspector findings reports. When you deactivate a scan type, you might encounter the following changes in the AWS account where you deactivated the scan type:

Amazon EC2 scanning

When you deactivate Amazon Inspector Amazon EC2 scanning for an account, the following SSM associations are deleted:

- `InspectorDistributor-do-not-delete`
- `InspectorInventoryCollection-do-not-delete`
- `InspectorLinuxDistributor-do-not-delete`
- `InvokeInspectorLinuxSsmPlugin-do-not-delete`
- `InvokeInspectorSsmPlugin-do-not-delete`.

Additionally, the Amazon Inspector SSM plugin is removed from all Windows hosts. For more information, see Scanning Windows EC2 instance.

Machine image scanning

When you deactivate machine image scanning for an account, Amazon Inspector stops monitoring the machine images in that account, removes them from your coverage, and deletes the machine image scan configuration for the account. If you reactivate machine image scanning later, Amazon Inspector applies the default scan configuration.

Amazon ECR scanning

When you deactivate Amazon ECR scanning for an account, the Amazon ECR scan type account changes from **Enhanced scanning** with Amazon Inspector to **Basic scanning** with Amazon ECR.

Lambda standard scanning

When you deactivate Lambda standard scanning for an account, you deactivate Lambda code scanning if the scan type was actived. You also delete the CloudTrail service-linked channel that Amazon Inspector create when you activate Lambda standard scanning.

Amazon Inspector Code Security

When you deactivate Code Security for your account, you delete all integrations, projects, and scan configurations associated with it. If your account is the delegated administrator for an organization, you only deactivate Code Security for your account, and memeber accounts become standalone accounts.

Deactivating scans

Deactivating all scan types for an account deactivates Amazon Inspector for that account in that AWS Region. For more information, see Deactivating Amazon Inspector.

To complete this procedure for a multi-account environment, follow these steps while signed in as the Amazon Inspector delegated administrator.

Console

To deactivate scans

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. By using the AWS Region selector in the upper-right corner of the page, select the Region where you want to deactivate scans.
3. In the navigation pane, choose **Account management**.
4. Choose the **Accounts** tab to show the scanning status of an account.
5. Select the check box of each account for which you want to deactivate scans.
6. Choose **Actions**, and, from the **Deactivate** options, select the scan type you wish to deactivate.
7. (Recommended) Repeat these steps in each AWS Region for which you want to deactivate that scan type.

API

Run the Disable API operation. In the request, provide the account IDs you are deactivating scans for, and for `resourceTypes` provide one or more of `EC2`, `MACHINE_IMAGE`, `ECR`, `LAMBDA`, or `LAMBDA_CODE` to deactivate scans.

Center for Internet Security (CIS) scans for Amazon EC2 instance operating systems

Amazon Inspector CIS scans (CIS scans) benchmark your Amazon EC2 instance operating systems to make sure you configured them according to best practice recommendations established by the Center for Internet Security. CIS Security Benchmarks provides industry standard configuration baselines and best practices for configuring a system securely. You can perform or schedule CIS scans after you enable Amazon Inspector EC2 scanning for an account. For information about how to activate Amazon EC2 scanning, see Activating a scan type.

> **Note**
>
> CIS standards are intended for x86_64 operating systems. Some checks may not be evaluated or return invalid remediation instructions on ARM-based resources.

Amazon Inspector performs CIS scans on target Amazon EC2 instances based on instance tags and your defined scanning schedule. Amazon Inspector performs a series of instance checks on each targeted instance. Each check evaluates whether your system configuration meets specific CIS Benchmark recommendations. Each check has a CIS check ID and title, which correspond with a CIS Benchmark recommendation for that platform. When a CIS scan completes, you can view the results to see which instance checks passed, skipped, or failed for that system.

> **Note**
>
> To perform or schedule CIS scans, you must have a secure internet connection. However, if you want to run CIS scans on private instances, you must use a VPC endpoint.

Topics

- Amazon EC2 instance requirements for Amazon Inspector CIS scans
- Running CIS scans
- Considerations for managing Amazon Inspector CIS scans with AWS Organizations
- Amazon Inspector owned Amazon S3 buckets used for Amazon Inspector CIS scans
- Creating a CIS scan configuration

- Viewing CIS scan results
- Editing a CIS scan configuration
- Downloading a CIS scan results

Amazon EC2 instance requirements for Amazon Inspector CIS scans

To run a CIS scan on your Amazon EC2 instance, the Amazon EC2 instance must meet the following criteria:

- The instance operating system is one of the supported operating systems for CIS scans. For more information, see Operating systems and programming languages supported by Amazon Inspector.
- The instance is an Amazon EC2 Systems Manager instance. For more information, see Working with the SSM Agent in the *AWS Systems Manager User Guide*.
- The Amazon Inspector SSM plugin is installed on the instance. Amazon Inspector automatically installs this plugin on manged instances.
- The instance has an instance profile that grants permissions for SSM to manage the instance and Amazon Inspector to run CIS scans for that instance. To grant these permissions, attach the AmazonSSMManagedInstanceCore and AmazonInspector2ManagedCisPolicy policies to an IAM role. Then attach the IAM role to your instance as an instance profile. For instructions on creating and attaching an instance profile, see Work with IAM roles in the *Amazon EC2 User Guide*.

> **Note**
>
> You're not required to enable Amazon Inspector deep inspection before running a CIS scan on your Amazon EC2 instance. If you disable Amazon Inspector deep inspection, Amazon Inspector automatically installs the SSM Agent, but the SSM Agent won't be invoked to run deep inspection anymore. However, as a result, the `InspectorLinuxDistributor-do-not-delete` association is present in your account.

Amazon Virtual Private Cloud endpoint requirements for running CIS scans on private Amazon EC2 instances

You can run CIS scans on Amazon EC2 instances over an Amazon network. However, if you want to run CIS scans on private Amazon EC2 instances, you must create Amazon VPC endpoints. The following endpoints are required when you create Amazon VPC endpoints for Systems Manager:

- `com.amazonaws.region.inspector2`
- `com.amazonaws.region.s3`
- `com.amazonaws.region.ssm`
- `com.amazonaws.region.ssmmessages`

For more information, see Creating Amazon VPC endpoints for Systems Manager in the *AWS Systems Manager User Guide*.

> **Note**
>
> Beginning with SSM Agent version 3.3.40.0, SSM Agent uses the `ssmmessages` endpoint instead of the `ec2messages` endpoint whenever it's available. Earlier versions of SSM Agent use the `ec2messages` endpoint. This endpoint is supported only in AWS Regions launched before 2024. For more information, see Reference: ec2messages, ssmmessages, and other API operations in the *AWS Systems Manager User Guide*.

> **Note**
>
> Currently, some AWS Regions don't support the `amazonaws.com.region.inspector2` endpoint.

Running CIS scans

You can either run a CIS scan once on-demand or as a scheduled recurring scan. To run a scan, you first create a scan configuration.

When you create a scan configuration, you specify tag key-value pairs to use to target instances. If you are the Amazon Inspector delegated administrator for an organization, you can specify

multiple accounts in the scan configuration, and Amazon Inspector will look for instances with the specified tags in each of those accounts. You choose the CIS Benchmark level for the scan. For each benchmark, CIS supports a level 1 and level 2 profile designed to provide baselines for different levels of security that different environments may require.

- **Level 1** – recommends essential basic security settings that can be configured on any system. Implementing these settings should cause little or no interruption of service. The goal of these recommendations is to reduce the number of entry points into your systems, reducing your overall cybersecurity risks.
- **Level 2** – recommends more advanced security settings for high-security environments. Implementing these settings requires planning and coordination to minimize the risk of business impact. The goal of these recommendations is to help you achieve regulatory compliance.

Level 2 extends level 1. When you choose Level 2, Amazon Inspector checks for all configurations recommended for level 1 and level 2.

After defining the parameters for your scan, you can choose whether to run it as a one time scan, which runs after you complete the configuration, or a recurring scan. Recurring scans can run daily, weekly, or monthly, at a time of your choice.

> **Tip**
>
> We recommend choosing a day and time that's least likely to impact your system while the scan is running.

Considerations for managing Amazon Inspector CIS scans with AWS Organizations

When you run CIS scans in an organization, Amazon Inspector delegated administrators and member accounts interact with CIS scan configurations and scan results differently.

How Amazon Inspector delegated administrators can interact with CIS scan configurations and scan results

When the delegated administrator creates a scan configuration, either for all accounts or a specific member accounts, the organization owns the configuration. Scan configurations that an organization owns have an ARN specifying the organization ID as the owner:

`arn:aws:inspector2:Region:111122223333:owner/OrganizationId/cis-configuration/scanId`

The delegated administrator can manage scan configurations that an organization owns, even if another account created them.

The delegated administrator can view scan results for any account in its organization.

If the delegated administrator creates a scan configuration and specifies SELF as the target account, the delegated administrator owns scan configuration, even if they leave the organization. However, the delegated administrator cannot change the target of a scan configuration with SELF as the target.

> **Note**
>
> The delegated adminstrator cannot add tags to CIS scan configurations the organization owns.

How Amazon Inspector member accounts can interact with CIS scan configurations and scan results

When a member account creates a CIS scan configuration, it owns the configuration. However, the delegated administrator can view the configuration. If a member account leaves the organization, the delegated administrator won't be able to view the configuration.

> **Note**
>
> The delegated administrator cannot edit a scan configuration the member account creates.

Member accounts, delegated administrators with SELF as the target, and standalone accounts all own scan configurations they create. These scan configurations have an ARN that shows the account ID as the owner:

`arn:aws:inspector2:Region:111122223333:owner/111122223333/cis-configuration/scanId`

A member account can view scan results in their account, including scan results from CIS scans the delegated administrator scheduled.

Amazon Inspector owned Amazon S3 buckets used for Amazon Inspector CIS scans

Open Vulnerability and Assessment Language (OVAL) is an information security effort that standardizes how to assess and report the machine state of computer systems. The following table lists all of the Amazon Inspector owned Amazon S3 buckets with OVAL defintions that are used for CIS scans. Amazon Inspector stages OVAL definition files that are required for CIS scans. The Amazon Inspector owned Amazon S3 buckets should be allowlisted in VPCs if necessary.

> **Note**
>
> The details for each of the following Amazon Inspector owned Amazon S3 buckets aren't subject to change. However, the table might be updated to reflect newly supported AWS Regions. You can't use Amazon Inspector ownerd Amazon S3 buckets for other Amazon S3 operations or in your own Amazon S3 buckets.

CIS bucket	AWS Region
`cis-datasets-prod-arn-5908f6f`	Europe (Stockholm)
`cis-datasets-prod-bah-8f88801`	Middle East (Bahrain)
`cis-datasets-prod-bjs-0f40506`	China (Beijing)
`cis-datasets-prod-bom-435a167`	Asia Pacific (Mumbai)
`cis-datasets-prod-cdg-f3a9c58`	Europe (Paris)
`cis-datasets-prod-cgk-09eb12f`	Asia Pacific (Jakarta)
`cis-datasets-prod-cmh-63030b9`	US East (Ohio)
`cis-datasets-prod-cpt-02c5c6f`	Africa (Cape Town)
`cis-datasets-prod-dub-984936f`	Europe (Ireland)
`cis-datasets-prod-fra-6eb96eb`	Europe (Frankfurt)

CIS bucket	AWS Region
cis-datasets-prod-gru-de69f99	South America (São Paulo)
cis-datasets-prod-hkg-8e30800	Asia Pacific (Hong Kong)
cis-datasets-prod-iad-8438411	US East (N. Virginia)
cis-datasets-prod-icn-f4eff1c	Asia Pacific (Seoul)
cis-datasets-prod-kix-5743b21	Asia Pacific (Osaka)
cis-datasets-prod-lhr-8b1fbd0	Europe (London)
cis-datasets-prod-mxp-7b1bbce	Europe (Milan)
cis-datasets-prod-nrt-464f684	Asia Pacific (Tokyo)
cis-datasets-prod-osu-5bead6f	AWS GovCloud (US-East)
cis-datasets-prod-pdt-adadf9c	AWS GovCloud (US-West)
cis-datasets-prod-pdx-acfb052	US West (Oregon)
cis-datasets-prod-sfo-1515ba8	US West (N. California)
cis-datasets-prod-sin-309725b	Asia Pacific (Singapore)
cis-datasets-prod-syd-f349107	Asia Pacific (Sydney)
cis-datasets-prod-yul-5e0c95e	Canada (Central)
cis-datasets-prod-zhy-5a8eacb	China (Ningxia)
cis-datasets-prod-zrh-67e0e3d	Europe (Zurich)

Creating a CIS scan configuration

This topic describes how to create a CIS scan configuration.

To run a CIS scan

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the AWS Region dropdown to select the AWS Region where you want to run a CIS scan.
3. From the navigation pane, choose **On-demand scans**, and then choose **CIS scans**.
4. Choose **Create new scan**.
5. For **Scan configuration name**, enter a **Scan configuration name**.
6. For **Target resource tags**, enter a **Key** and corresponding **Value** for the instances you want to scan. You can specify up to five different values for each key and a total of 25 tags to include in the scan.
7. For **CIS Benchmark level**, you can select **Level 1** for basic security configurations or **Level 2** for advanced security configurations.
8. For **Target accounts**, specify which accounts to include in the CIS scan. For more information, see Considerations for managing Amazon Inspector CIS scans with AWS Organizations.

   If your account is the delegated administrator account, you can select **All accounts** or **Specify accounts**. The **All accounts** option targets all accounts in your organization. The **Specify accounts** only targets individual accounts in your organization. If you choose this option, you can specify more than one account by separating the account numbers with a comma. You can also enter SELF instead of an account ID to create a scan configuration for your account

   If your account is a standalone account or member account in an organization, you can select **Self** to create a scan configuration for your account.
9. For **Schedule**, choose **One time scan**, which runs as soon as you finish creating your scan configuration, or **Recurring scans**, which runs at the time you specify.
10. Confirm your choices, and then choose **Create**.

Viewing CIS scan results

Amazon Inspector creates a scan job for every scan configuration that runs and collects results of a scan with a unique scan ID. CIS scan results are available for 90 days. You can view CIS scan results by its checks or scanned resources:

- **Scan results aggregated by checks** – Groups the results of a scan by each individual check performed during the scan. For each check, you get a report of how many resources failed, skipped, or passed.
- **Scan results aggregated by scanned resources** – Groups the results of a scan by each scanned resource the scan targets during the scan. For each resource, you get a report of how many checks that a resource failed, skipped, or passed.

This topic describes how to view results for a CIS scan.

To view scan results

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the AWS Region dropdown to select the AWS Region where you created your CIS scan configuration.
3. From the navigation pane, choose **On-demand scans**, and then choose **CIS scans**.
4. Choose the **Scan results** tab.
5. Under the **Scheduled by** column, choose the scan schedule ID you want to view. Or select the row with the scan schedule ID you want to view, and then choose **View details**.
6. Choose **Checks** to view each check that was performed performed or **Scanned resources** to view each scanned resource that was targeted during the scan.

You can also view details for scheduled CIS scans.

To view details for scheduled CIS scans

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the AWS Region dropdown to select the AWS Region where you created your CIS scan configuration.
3. From the navigation pane, choose **On-demand scans**, and then choose **CIS scans**.
4. Choose the **Scheduled** tab.
5. Under the **Scan configuration name** column, choose the name of the scan configuration you want to view. Or select the row with the scan configuration you want to view, and then choose **View details**.

Editing a CIS scan configuration

This topic describes how to edit a CIS scan configuration.

To edit a CIS scan configuration

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the AWS Region dropdown to select the AWS Region where you created your CIS scan configuration.
3. From the navigation pane, choose **On-demand scans**, and then choose **CIS scans**.
4. Choose the **Scheduled** tab.
5. Select the row with the scan configuration you want to edit, and then choose **Edit**.

Downloading a CIS scan results

You can download a PDF or CSV of a CIS scan using the Amazon Inspector console or API.

> **Note**
>
> You can only download a CSV file of your CIS scan results for CIS scans collected after 05/03/2024.

This topic describes how to download a CIS scan using the Amazon Inspector console.

To download CIS scan results from the console

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the AWS Region dropdown to select the AWS Region where you created your CIS scan configuration.
3. From the navigation pane, choose **On-demand scans**, and then choose **CIS scans**.
4. Choose the **Scan results** tab.
5. Under the **Scheduled By** column, choose the scan schedule ID that you want to view. Or select the row with the scan schedule ID that you want to view, and then choose **View details**.

6. Choose **Download**, and then choose **PDF** or **CSV**. If your account is the delegated administrator account, you can choose **Select account** to download results for a specific member account.

Amazon Inspector Code Security

Amazon Inspector is a vulnerability management service that automatically discovers workloads and continually scans them for software vulnerabilities and unintended network exposure. With Code Security, Amazon Inspector scans first-party application source code, third-party application dependencies, and Infrastructure as Code for vulnerabilities. You can activate Code Security in the Amazon Inspector console or with the Amazon Inspector API. Once you activate Code Security, you can create and apply a scan configuration to your code repository to determine how often and when it will be scanned. You can view, edit, and delete your scan configuration at any time. For information about the AWS Regions where Code Security is available, see Regions and endpoints. For information about pricing, see Amazon Inspector pricing.

Prerequisites for Code Security

Before you can begin using Code Security, you must activate Code Security and decide how to encrypt your data. This can be information like integration credentials, code, or any other information related to your integrations, code repositories, and projects. By default, your data is encrypted with an AWS owned key. This means the key is created, owned, and managed by the service. If you want to own and manage the key used to encrypt your data, you can create a customer managed KMS key.

Activating Code Security

You activate Code Security in the same way that you activate all automated scan types. For more information, see Activating a scan type.

Creating a customer managed key to access AWS KMS

By default, your data is encrypted with an AWS owned key. This means the key is created, owned, and managed by the service. If you want to own and manage the key used to encrypt your data, you can create a customer managed KMS key. Amazon Inspector doesn't interact with your data. Amazon Inspector only ingests metadata from repositories in your source code provider. For information about how to create a customer managed KMS key, see Create a KMS key in the *AWS Key Management Service User Guide*.

Sample policy

When you create your customer managed key, use the following sample policy.

Note

The FAS permissions in the following policy are specific to Amazon Inspector, as they allow Amazon Inspector to perform only those API calls.

JSON

```
{
  "Version":"2012-10-17",
  "Id": "key-policy",
  "Statement": [
    {
      "Sid": "Allow Q to use Encrypt Decrypt GenerateDataKey and
 GenerateDataKeyWithoutPlaintext",
      "Effect": "Allow",
      "Principal": {
        "Service": "q.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
        "aws:SourceAccount": "111122223333"
        },
        "StringLike": {
        "kms:EncryptionContext:aws:qdeveloper:codesecurity-scope": "111122223333"
        },
        "ArnLike": {
        "aws:SourceArn": "arn:aws:inspector2:us-east-1:111122223333:codesecurity-
integration/*"
        }
      }
    },
    {
      "Sid": "Allow Q to use DescribeKey",
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "q.amazonaws.com"
      },
      "Action": "kms:DescribeKey",
      "Resource": "*"
    },
    {
      "Sid": "Allow Inspector to use Encrypt Decrypt GenerateDataKey and
GenerateDataKeyWithoutPlaintext using FAS",
      "Effect": "Allow",
      "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/inspectorCodeSecurity"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "inspector2.us-east-1.amazonaws.com"
        },
        "StringLike": {
        "kms:EncryptionContext:aws:qdeveloper:codesecurity-scope": "111122223333"
        }
      }
    },
    {
      "Sid": "Allow Inspector to use DescribeKey using FAS",
      "Effect": "Allow",
      "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/inspectorCodeSecurity"
      },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "inspector2.us-east-1.amazonaws.com"
        }
      }
```

```
    }
  ]
}
```

After you create your KMS key, you can use the following Amazon Inspector APIs.

- UpdateEncryptionKey – Use with `CODE_REPOSITORY` for `resourceType` and `CODE` as the scan type to configure the use of your customer managed KMS key.
- GetEncryptionKey – Use with `CODE_REPOSITORY` for `resourceType` and `CODE` as the scan type to configure the retrieval of your KMS key configuration.
- ResetEncryptionKey – Use with `CODE_REPOSITORY` for `resourceType` and `CODE` to reset your KMS key configuration and to use an AWS owned KMS key.

Creating an integration between Amazon Inspector your code repository

This section includes topics that describe how to create an integration between Amazon Inspector and your code repository. When you create an integration, all code repositories are listed as projects in the Amazon Inspector console on the **Code Security** page. Other topics in this section describe how to access your integrations and projects.

Code Security only imports up to 100,000 projects, and only the default branch for each repository is monitored. A project can be associated with a maximum of three default scan configurations.

Code Security only supports a maximum of 100 integrations per account. Code Security integrations have no concept of the delegated administrator account/member account relationship.

To avoid encountering restrictions, we recommend not using the same host for an integration more than once.

Integrations with GitHub SaaS, GitHub Enterprise Cloud, and GitHub Enterprise Server require public internet access.

Important

Third-party integrations might be temporarily or permanently disabled without prior notice for any reason, such as to address security concerns.

Creating an integration between Amazon Inspector and GitHub

This topic describes how to create an integration between Amazon Inspector and GitHub.

Note

If this is your first time creating an integration, you're prompted to create a default scan configuration on Step 2. When you create a scan configuration, you choose the scan frequency, scan analysis, and repositories to be scanned. Creating a default scan configuration is the same as creating a general scan configuration. However, the default scan configuration is automatically associated with any new and existing projects imported into Amazon Inspector. If you want to create a default scan configuration, choose **Continue with this configuration**. You can only create a default scan configuration once. If you create a default scan configuration, you won't be prompted to create a default scan configuration again. You can only create a default scan configuration once per account and once per organization. If you don't want to configure a default scan configuration, choose **Skip configuration**. However, will be prompted to create a default scan configuration the next time you create an integration. After you create a default scan configuration or skip creating a default scan configuration, you're directed to Step 3 of the integration workflow where you enter your integration details.

Integrations with GitHub SaaS, GitHub Enterprise Cloud, and GitHub Enterprise Server require public internet access.

Note

Amazon Inspector only scans and monitors your default branch. If you create a new default branch, Amazon Inspector scans and updates the new default branch.

> **Important**
>
> Before you finish creating the integration, you're directed to authorize the connection between Amazon Inspector and GitHub. You must complete this step to finish the procedure. If you close the pop-up, you will not be able to proceed.

To create an integration between Amazon Inspector and GitHub

1. Sign in using your credentials. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**. Choose **Connect to**, and choose GitHub.
3. Under **Integration details**, enter the name of your integration, and choose **Connect to GitHub**.
4. Choose **Authorize** in the pop-up to create a connection between Amazon Inspector and GitHub.
5. In the success banner, choose **Go to GitHub connection creation page**.
6. Enter the installation ID for the GitHub application. If you installed the GitHub application, you can find the installation ID in GitHub from the **GitHub Apps** page or at the end of the GitHub application URL. If you haven't installed the GitHub application, choose **Install a new app**. This directs you to GitHub where you select the GitHub organization and specify the repository scope.
7. Choose **Connect to GitHub**.

After you create the integration, you can encounter a scenario where Amazon Inspector is unable to refresh the access token. This can occur if the integration host is unavailable or Amazon Inspector experiences other communication issues. To remediate the issue, you can re-authenticate the connection from the **Integrations** tab on the **Code Security** page. Under the **Status** column, the integration shows as **Inactive**, and Amazon Inspector provides the option to re-authenticate. Choose **Re-authenticate**. You're redirected to the integration workflow where you can complete the connection setup.

If you delete system settings for your integration, you can lose connection indefinitely. If this occurs, you must delete the integration and create a new integration. When you delete an integration, you lose all projects and scan configurations associated with the integration.

Creating an integration between Amazon Inspector and GitLab Self Managed

This topic describes how to create an integration between Amazon Inspector and your code repository in GitLab Self Managed.

Required information

The following is required when you create a connection:

- Integration name – This is the name added to the body of your integration.
- Endpoint URL – This is the URL used to access your GitLab Self Managed instance.
- Personal access token – The personal access token is created in GitLab Self Managed from an administrator account and must include the following scopes: `api`, `read_api`, and `read_repository`.

> **Note**
>
> Amazon Inspector only scans and monitors your default branch. If you create a new default branch, Amazon Inspector scans and updates the new default branch.

Creating an integration between Amazon Inspector and GitLab Self Managed

The following procedure describes how to create a connection between Amazon Inspector and your code repository in GitLab Self Managed.

> **Note**
>
> If this is your first time creating an integration, you're prompted to create a default scan configuration on Step 2. When you create a scan configuration, you choose the scan frequency, scan analysis, and repositories to be scanned. Creating a default scan configuration is the same as creating a general scan configuration. However, the default scan configuration is automatically associated with any new and existing projects imported into Amazon Inspector. If you want to create a default scan configuration, choose **Continue with this configuration**. You can only create a default scan configuration once. If you create a default scan configuration, you won't be prompted to create a default scan configuration again. You can only create a default scan configuration once per account and

once per organization. If you don't want to configure a default scan configuration, choose **Skip configuration**. However, you will be prompted to create a default scan configuration the next time you create an integration. After you create a default scan configuration or skip creating a default scan configuration, you're directed to Step 3 of the integration workflow where you enter your integration details.

Important

Before you finish creating the integration, you're prompted to authorize the connection between Amazon Inspector and GitLab Self Managed. You must complete this step to finish the procedure. If you close the pop-up, you will not be able to proceed.

To create a connection with GitLab Self Managed

1. Sign in using your credentials. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**. Choose **Connect to** and choose **GitLab Self Managed**.
3. Under **Integration details**, enter the following:
   a. For **Integration name**, enter the name added to the body of your integration.
   b. For **Endpoint URL**, enter the URL used to access your GitLab self-managed instance.
   c. For **Personal access token**, enter your personal access token with the required scopes.
4. Choose connect to GitLab.
5. Choose **Authorize** in the pop-up window to finish creating a connection between Amazon Inspector and GitLab.

After you create the integration, you can encounter a scenario where Amazon Inspector is unable to refresh the access token. This can occur if the integration host is unavailable or Amazon Inspector experiences other communication issues. To remediate the issue, you can re-authenticate the connection from the **Integrations** tab on the **Code Security** page. Under the **Status** column, the integration shows as **Inactive**, and Amazon Inspector provides the option to re-authenticate. Choose **Re-authenticate**. You're redirected to the integration workflow where you can complete the connection setup.

If you delete system settings for your integration, you can lose connection indefinitely. If this occurs, you must delete the integration and create a new integration. When you delete an integration, you lose all projects and scan configurations associated with the integration.

Viewing integrations with code repositories

This topic describes how to view integrations in the Amazon Inspector console.

To view integrations in the Amazon Inspector console

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Integrations**. From this tab, you can review all of your configured integrations and review basic information about all of your integrations. This information includes the name of the integration, status of the integration, and source code provider name.

Re-authenticate to provider

After you create the integration, you can encounter a scenario where Amazon Inspector is unable to refresh the access token. This can occur if the integration host is unavailable or Amazon Inspector experiences other communication issues. To remediate the issue, you can re-authenticate the connection from the **Integrations** tab on the **Code Security** page. Under the **Status** column, the integration shows as **Inactive**, and Amazon Inspector provides the option to re-authenticate. Choose **Re-authenticate**. You're redirected to the integration workflow where you can complete the connection setup.

If you delete system settings for your integration, you can lose connection indefinitely. If this occurs, you must delete the integration and create a new integration. When you delete an integration, you lose all projects and scan configurations associated with the integration.

Viewing code repositories

The topic describes how to view code repositories in the Amazon Inspector console.

To view code repositories in the Amazon Inspector console

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.

2. From the navigation pane, choose **Code Security**.
3. Choose **Code repositories**. From this tab, you can review all of your code repositories, which are listed as projects, and review basic information about them. This information includes the name and scan status for each project. You can also review the configurations associated with your projects and when your projects were last scanned. You can even filter your projects in the search bar.

Viewing details for a project

This topic describes how to view details for a project in the Amazon Inspector console. If your account is the delegated administrator for an organization, you can view details for projects that belong to member accounts.

To view code projects in the Amazon Inspector console

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Code repositories**. From this tab, you can review all of your code repositories, which are listed as projects, and review basic information about them. This information includes the name and scan status for each project. You can also review the configurations associated with your projects and when your projects were last scanned. You can even filter your projects in the search bar.
4. Choose a project. Or select a project, and choose **View details**. From the **Project details** screen, you can review basic information about the project. This information includes the name and ID for the project, as well as the integration ARN. It includes information about when the project was scanned and the provide type. You can even review findings associated with the project, as well as export findings and create suppression rules for findings.

Deleting an integration

The following procedure describes how to delete an integration in the Amazon Inspector console. When you delete an integration, you lose all projects and scan configurations associated with the integration.

To delete an integration in the Amazon Inspector console.

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Integrations**. From this tab, you can review all of your configured integrations and review basic information about all of your integrations. This information includes the name of the integration, status of the integration, and integration provider type.
4. Select an integration, and choose **Delete**.

Creating a scan configuration

Before you create a scan configuration, you must create an integration with Amazon Inspector. The first time you create an integration, you're prompted to create a default scan configuration. This topic describes how to create a general scan configuration. The difference between a default scan configuration and a general scan configuration is that a default scan configuration is automatically attached to new projects. You can skip creating a default scan configuration.

Code Security only supports a maximum of 500 general scan configurations. Code security only supports 1 default scan configuration per account and per organization. A scan configuration only can be associated with a maximum of 100,000 projects.

A project can be associated with a maximum of 4 scan configurations total. This includes a default scan configuration if a default scan configuration was created. Scan configurations for an organization cannot be tagged.

If the delegated administrator for an organization creates a scan configuration, the scan configuration is created at the organization level and applied to all member accounts in the organization. The same occurs if the delegated administrator creates a default scan configuration.

When you create a scan configuration, you choose the scan frequency, scan analysis, and repositories to be scanned. The scan frequency can be change based and periodic or customized. Change-based and periodic scanning gives you the option to enable periodic scanning. If you enable periodic scanning, you set the scan frequency to the day of the week or month when a scan occurs. Customized scanning gives you the option to enable scanning when code is changed and periodic scanning. If you enable scanning when code is changed, you specify the scan trigger to include in merge and pull requests.

Scans can be skipped if a commit ID hasn't changed in a set amount of time. For periodic scanning, scans are skipped if a commit ID hasn't changed between scans in 1 week. For on-demand scans, scans are skipped if a commit ID hasn't changed between scans in 24 hours.

> **Note**
>
> If a scan configuration only has triggers for merge requests and pull requests, only the top 25 critical or high findings are presented and only in the source code management platform. None will be visible in Amazon Inspector.

To create a general scan configuration

1. Sign in using your credentials. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Configurations**, and then choose **Create scan configuration**.
4. Under **Scan details**, do the following:
   - For **Configuration name**, enter a name for the scan configuration.
5. Under **Scan frequency**, specify how often code is scanned by choosing **Change-based and periodic scanning** or **Customized scanning types and triggers**.
   a. (Option 1) If you choose **Change based and periodic scanning**, choose **Enable periodic scanning** or **Disable periodic scanning**.
      - . If you choose **Enable periodic scanning**, set the scan frequency by choosing the week and day you want code to be scanned.
   b. (Option 2) If you choose **Customized scanning**, decide whether to enable scanning when code is changed and periodic scanning.
      i. Choose **Enable scanning when code is changed** or **Disable scanning when code is changed**. If you choose **Enable scanning when code is changed**, specify when scans are triggered from the dropdown.
      ii. Choose **Enable periodic scanning** or **Disable periodic scanning**. If you choose **Enable periodic scanning**, set the scan frequency by choosing the week and day you want code to be scanned. You can also scan on event-based triggers. These events include

when a new pull request is initially opened against the default branch and when a commit is merged or pushed to the default branch. Scans are not triggered on subsequent updates or revisions to an existing pull request. To trigger a new scan, close and reopen the pull request.

6. Under **Scan analysis**, decide whether to configure a complete scanning analysis or customized scanning analysis:

   a. (Option 1) If you choose **Complete scanning analysis**, you apply all of the following scan analyses:

      - *Static Application Security Testing* – Analyzes source code for vulnerabilities.
      - *IaC scanning* – Analyzes scripts and code that configure and provision infrastructure.
      - *Static software composition analysis* – Examines open source packages in applications.

   b. (Option 2) If you choose **Customized scanning analysis**, you must choose at least one type of the previously mentioned scan analysis types from the dropdown menu:

7. (Optional) For **Tags**, create a key-value pair to apply to your project. You can create up to 50 tags.
8. Choose **Next**.
9. Under **Repository selection**, choose **All repositories** or **Specific repositories**.

   a. (Option 1) If you choose **All repositories**, scanning is enabled for any of your existing repositories.

   b. (Option 2) If you choose **Specific repositories**, scanning is enabled only for the repositories that you specify.

10. Choose **Next**.
11. Review your choices, and then choose **Create scan configuration**.

> **Note**
>
> General scan configurations are applied to all existing code repositories only. They will not be applied to new code repositories.

Viewing scan configurations

The following procedure describes how to view scan configurations in the Amazon Inspector console.

> **Note**
>
> When you view your scan configuration at the organization level, some of the details in the **Code Security** screen will differ to reflect your AWS account.

To view details for a scan configuration

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Configurations** to view a list of your scan configurations. If you're the delegated administrator, the list include your organization's scan configurations. You can see the name of each scan configuration and who created each scan configuration (AWS account ID or organization ID). You can also view which scanning types and scan analysis type are applied to the configuration. You can even filter your scan configuration by different fields in the search bar.

Viewing details for a scan configuration

The following procedure describes how to view details for a scan configuration in the Amazon Inspector console.

To view details for a scan configuration

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Configurations**.
4. Choose the configuration you want to view details for. The scan configuration details screen provides an overview of the scan configuration. From this screen, you can view the scan configuration ARN, which scan frequency types are enabled, and which scan analysis types are

enabled. You can also delete the scan configuration from this screen. If you're viewing a scan configuration that belongs to your organization, you can edit from this screen, too.

Editing a scan configuration

You can edit a scan configuration at any time. When editing a scan configuration, you can change the scan frequency, scan analysis, tags, and repositories to be scanned. For example, you edit a scan configuration to pause scanning for a particular repository. The following procedure describes how to edit a scan configuration.

To edit a scan configuration

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code Security**.
3. Choose **Configurations**.
4. Select the configuration you want to edit, and then choose **Edit**. You can also choose the configuration you want to edit, and then choose **Edit**.

Deleting a scan configuration

You can delete a scan configuration at any time. This topic describes how to delete a scan configuration.

To delete a scan configuration

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code security**.
3. Choose **Configurations**.
4. Select the configuration you want to delete, and then choose **Delete**. Or choose the configuration you want to delete, and then choose **Delete**.

Performing an on-demand scan

You can perform an on-demand for your projects. When you perform an on-demand scan, a union of all your configured scan configurations is applied to your selected project. If your account is the delegated administrator account for an organization, you can perform an on-demand scan for projects that belong to member accounts. The following procedure describes how to perform an on-demand scan in the Amazon Inspector console.

To perform an on-demand scan

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Code security**.
3. Choose **Code repositories**.
4. Select the project you want to scan, and then choose **On-demand** scan.

Supported languages for Amazon Inspector code security

This topic includes the supported languages for Amazon Inspector Code Security.

Supported languages for SAST

- C# (all versions but .Net 6.0 and later is recommended)
- C (C11 or earlier)
- C++ (C++ 17 or earlier)
- Go (Go 1.18 only)
- Java (Java 25 or earlier)
- JavaScript (ECMAScript 2021 or earlier)
- JSX (React 17 or earlier)
- Kotlin (Kotlin 2.0 or earlier)
- PHP (PHP 8.2 or earlier)
- Python (Python 3.13 or earlier within the Python 3 series)
- Ruby (Ruby 2.7 and 3.2 only)
- Rust

- Scala (Scala 3.2.2 or earlier)
- Shell
- TSX
- TypeScript (all versions)

Supported languages for software composition analysis

- Go
- Java (Java 25 or earlier)
- JavaScript (ECMAScript 2021 or earlier)
- PHP (PHP 8.2 or earlier)
- Python (Python 3.13 or earlier within the Python 3 series)
- .Net
- Ruby (Ruby 2.7 and 3.2 only)
- Rust

> **Note**
>
> For npm (Node.js) projects, Amazon Inspector code security software composition analysis identifies dependencies from the `package.json` manifest and excludes the `package-lock.json` lockfile from analysis. Dependencies that appear only in `package-lock.json`, including transitive dependencies and the exact versions that it pins, are not evaluated.

Languages for Infrastructure as Code

- AWS CDK (Python and TypeScript)
- CloudFormation (2010–09–09)
- Terraform (1.6.2 or earlier)

Deactivating Code Security

For more information about deactivating Code Security, see Deactivating a scan type.

Understanding Amazon Inspector findings

Amazon Inspector generates a finding when it detects a vulnerability with a fix or pending fix in Amazon EC2 instances, Amazon ECR containers images, and Lambda functions. It also generates findings for code vulnerabilities detected in first-party application source code, third-party application dependencies, and Infrastructure as Code. A finding is a detailed report about a vulnerability impacting one of your AWS resources.

Findings are named after vulnerabilities and provide severity ratings, information about impacted AWS resources and non AWS resources, and details that describe how to remediate detected vulnerabilities. Amazon Inspector stores all of your active findings until you remediate them.

When a resource is deleted, terminated, or no longer eligible for scanning, Amazon Inspector automatically closes findings associated with the resource and then deletes the findings after 3 days. If findings are closed for any other reason, they are deleted after 30 days.

> **Note**
>
> Amazon Inspector will reopen a remediated finding within seven days of closing the finding if the issue that caused the vulnerability reoccurs.

If you disable Amazon Inspector, findings are removed after 24 hours. If a resource is terminated, any finding related to the resource is removed after 3 days. The same occurs for any finding attached to a resource where scanning is no longer eligible. If AWS suspends your account, findings are removed after 90 days. Findings for stopped instances remain active.

Findings states

Amazon Inspector categorizes findings in the following states.

Active

Amazon Inspector categorizes a finding that hasn't been remediated as **Active**.

Suppressed

Amazon Inspector categorizes a finding subject to one or more suppression rules as **Suppressed**.

Closed

When a finding has been remediated, Amazon Inspector categorizes the finding as **Closed**.

Topics

- Amazon Inspector finding types
- Viewing Amazon Inspector findings
- Viewing details for your Amazon Inspector findings
- Viewing the Amazon Inspector score and understanding vulnerability intelligence details
- Understanding severity levels for your Amazon Inspector findings

Amazon Inspector finding types

This section describes the different finding types in Amazon Inspector.

Topics

- Package vulnerability
- Code vulnerability
- Network reachability

Package vulnerability

Package vulnerability findings identify software packages in your AWS environment that are exposed to Common Vulnerabilities and Exposures (CVEs). Attackers can exploit these unpatched vulnerabilities to compromise the confidentiality, integrity, or availability of data, or to access other systems. The CVE system is a reference method for publicly known information security vulnerabilities and exposures. For more information, see https://www.cve.org/.

Amazon Inspector can generate package vulnerability findings for EC2 instances, ECR container images, and Lambda functions. Package vulnerability findings include details that are unique to this type of finding. These details are the Inspector score and vulnerability intelligence.

For Windows EC2 instances, package vulnerability findings can be identified by Microsoft Knowledge Base (KB) IDs instead of individual CVEs. If a KB update addresses one or more CVEs,

Amazon Inspector reports a single KB finding, for example KB5023697, instead of a separate finding for each CVE. A KB finding specifies the highest CVSS score, EPSS score, and exploit availability across all constituent CVEs.

Code vulnerability

Code vulnerability findings help identify lines of code that can be exploited. Code vulnerabilities include missing encryption, data leaks, injection flaws, and weak cryptography. Amazon Inspector generates code vulnerability findings through Lambda function scanning and its Code Security feature.

Amazon Inspector evaluates Lambda function application code using automated reasoning and machine learning to analyzes application code for overall security compliance. It identifies policy violations and vulnerabilities based on internal detectors developed in collaboration with Amazon Q. For a list of possible detections, see Amazon Q Detector Library.

Code scanning captures snippets of code to highlight detected vulnerabilities. For example, a code snippet might show hardcoded credentials or other sensitive materials in plaintext. Amazon Q stores code snippets associated with code vulnerabilities. By default, your code is encrypted with an AWS owned key. However, you can create a customer managed key to encrypt your code if you want more control over this information. For more information, see Encryption at rest for code in your findings.

> **Note**
>
> The delegated administrator for an organization cannot view code snippets that belong to member accounts.

Network reachability

Network reachability findings indicate that there are open network paths to Amazon EC2 instances in your environment. These findings appear when your TCP and UDP ports are reachable from the VPC edges, such as an internet gateway (including instances behind Application Load Balancers or Classic Load Balancers), a VPC peering connection, or a VPN through a virtual gateway. These findings highlight network configurations that may be overly permissive, such as mismanaged security groups, Access Control Lists, or internet gateways, or that may allow for potentially malicious access.

Amazon Inspector only generates network reachability findings for Amazon EC2 instances. Amazon Inspector performs scans for network reachability findings every 12 hours once Amazon Inspector is enabled.

Amazon Inspector evaluates the following configurations when scanning for network paths:

- Amazon EC2 instances
- Application Load Balancers
- Direct Connect
- Elastic Load Balancers
- Elastic Network Interfaces
- Internet Gateways
- Network Access Control Lists
- Route Tables
- Security Groups
- Subnets
- Virtual Private Clouds
- Virtual Private Gateways
- VPC endpoints
- VPC gateway endpoints
- VPC peering connections
- VPN connections

Viewing Amazon Inspector findings

You can view findings in the Amazon Inspector console and with the Amazon Inspector `ListFindings` API. In the Amazon Inspector console, you can view all of your findings in the **Dashboard** and **Findings** screens. By default, these screens only show your active and critical findings. However, you can filter through findings or choose to view findings by category. You can also view some findings in Security Hub CSPM and Amazon ECR if you activate these integrations. The procedures in this section describe how to view findings in Amazon Inspector console and with the Amazon Inspector `ListFindings` API.

> **Note**
>
> Findings from on-demand machine image scans are an exception. Amazon Inspector delivers them only through Amazon EventBridge, so they don't appear in the Amazon Inspector console, aren't returned by `ListFindings`, and aren't sent to Security Hub CSPM. For more information, see Running on-demand machine image scans with Amazon Inspector.

Console

To view Amazon Inspector findings

1. Sign in using your credentials. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. (Optional) From the navigation pane, choose **Dashboard**. The dashboard shows an overview of the coverage for your environment and only your active and critical findings.
3. (Optional) From the navigation pane, choose **Findings**. This screen lists all of your active findings. You can use filter criteria to view specific findings. To exclude findings from the list, create a suppression rule. To view details for a finding, choose the name of the finding.
4. (Optional) From the navigation pane, choose one of the following options to view your findings by category:
   - By vulnerability – Shows vulnerabilities with the most critical findings.
   - By account – Shows accounts with the most critical findings. This category is only available to delegated administrators.
   - By instance – Shows Amazon EC2 instances with the most critical findings. This category does not include information about network availability.
   - By container image – Shows Amazon ECR container images with the most critical findings. This category also provides basic information about your container images. It even includes details, such as how many Amazon ECS tasks and Amazon EKS pods are deployed. From this screen, you can find out how many tasks/pods were running in past 24 hours and stopped.
   - By container repository – Shows container repositories with the most critical findings.
   - By Lambda function – Shows Lambda functions with the most critical findings.

API

To view Amazon Inspector findings

- Run the ListFindings API operation. In the request, specify filterCriteria to return specific findings.

Viewing details for your Amazon Inspector findings

The procedure in this section describes how to view details for Amazon Inspector findings.

To view the details for a finding

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home
2. Select the Region to view findings in.
3. In the navigation pane, choose **Findings** to display the findings list
4. (Optional) Use the filter bar to select a specific finding. For more information, see Filtering your Amazon Inspector findings.
5. Choose a finding to view its details panel.

The **Finding details** panel contains the basic identifying features of the finding. This includes the title of the finding as well as a basic description of the vulnerability identified, remediation suggestions, and a severity score. For information about scoring, see Understanding severity levels for your Amazon Inspector findings.

The details available for a finding varies depending on finding type and the **Resource affected**.

All findings contain the AWS account ID number the finding was identified for, a severity, a finding **Type**, the date the finding was created at, and a **Resource affected** section with details about that resource.

The finding **Type** determines the remediation and vulnerability intelligence information available for the finding. Depending on the finding type, different finding details are available.

Package Vulnerability

Package vulnerability findings are available for EC2 instances, machine images, ECR container images, and Lambda functions. See Package vulnerability for more info.

Package vulnerability findings also include Viewing the Amazon Inspector score and understanding vulnerability intelligence details.

This finding type has the following details:

- **Fix available** – Indicates if the vulnerability is fixed in a newer version of the affected packages. Has one of the following values:
  - `YES`, which means all the affected packages have a fixed version.
  - `NO`, which means no affected packages have a fixed version.
  - `PARTIAL`, which means one or more (but not all) of the affected packages have a fixed version.
- **Exploit available** – Indicates the vulnerability has a known exploit.
  - `YES`, which means the vulnerability discovered in your environment has a known exploit. Amazon Inspector doesn't have visibility into the use of exploits in an environment.
  - `NO`, which means this vulnerability doesn't have a known exploit.
- **Affected packages** – Lists each package identified as vulnerable in the finding, and the details of each package:
- **Filepath** – The EBS volume ID and partition number associated with a finding. This field is present in findings for EC2 instances scanned using Agentless scanning.
- **Installed version / Fixed version** – The version number of the currently installed package that a vulnerability was detected for. Compare the installed version number with the value after the slash (**/**). The second value is the version number of the package that fixes the detected vulnerability as provided by the Common Vulnerabilities and Exposures (CVEs) or advisory associated with the finding. If the vulnerability has been fixed in multiple versions, this field lists the most recent version that includes the fix. If a fix isn't available, this value is `None available`.

  > **Note**
  >
  > If a finding was detected before Amazon Inspector began including this field in findings, the value for this field is empty. However, a fix may be available.

- **Package manager** – The package manager used to configure this package.
- **Remediation** – Remediation commands vary depending on your system configuration. Review finding references or operating system documentation for more specific guidance. When

available, customers should update affected packages to the provided fixed version (or newer).

- **Vulnerability details** – provides a link to the Amazon Inspector preferred source for the CVE identified in the finding, such as National Vulnerability Database (NVD), REDHAT, or another OS vendor. Additionally, you will find the severity scores for the finding. For more information about severity scoring such as, see Understanding severity levels for your Amazon Inspector findings. The following scores are included, including the scoring vectors for each:
  - Exploit Prediction Scoring System (EPSS) score
  - Inspector score
  - CVSS 3.1 from Amazon CVE
  - CVSS 3.1 from NVD
  - CVSS 2.0 from NVD (where applicable, for older CVEs)
- **Related vulnerabilities** – Specifies other vulnerabilities related to the finding. Typically these are other CVEs that impact the same package version, or other CVEs within the same group as the finding CVE, as determined by the vendor.
- **Resources affected** – Includes information about the registry, repository, resource type, image ID, and image operating system. It also includes information, such as when an image was last pushed, how many Amazon ECS tasks and Amazon EKS pods are deployed, and when the image was last used in the past 24 hours. If you have any deployed Amazon ECS tasks and Amazon EKS pods, you can view details by choosing the value for the field. This directs you to a screen where you can view information, such as the cluster ARN, when the resource was last used in the past 24 hours, the running and stopped count for the resource, and the workload name and type.

Code vulnerability

Code vulnerability findings are available for Lambda functions only. See Code vulnerability for more info. This finding type has the following details:

- **Fix available** – For code vulnerabilities this value is always YES.
- **Detector name** – The name of the Amazon Q detector used to detect the code vulnerability. For a list of possible detections, see the Q Detector Library.
- **Detector tags** – The Amazon Q tags associated with the detector, Amazon Q uses tags to categorize detections.
- **Relevant CWE** – IDs of the Common Weakness Enumeration (CWE)s associated with the code vulnerability.

- **File path** – The file location of the code vulnerability.
- **Vulnerability location** – For Lambda code scanning code vulnerabilities, this field shows the exact lines of code where Amazon Inspector found the vulnerability.
- **Suggested remediation** – This suggests how the code can be edited to remediate the finding.

Network reachability

Network reachability findings are only available for EC2 instances. See Network reachability for more info. This finding type has the following details:

- **Open port range** – The port range through which the EC2 instance could be accessed.
- **Open network paths** – Shows the open access path to the EC2 instance. Select an item on the path for more information.
- **Remediation** – Recommends a method for closing the open network path.

Viewing the Amazon Inspector score and understanding vulnerability intelligence details

Amazon Inspector creates a score for Amazon Elastic Compute Cloud (Amazon EC2) instance findings. You can view the Amazon Inspector score and vulnerability intelligence details in the Amazon Inspector console. The Amazon Inspector score provides you with details that you can compare with metrics in the Common Vulnerability Scoring System. These details are only available for package vulnerability findings. This section describes how to interpret the Amazon Inspector score and understand vulnerability intelligence details.

Amazon Inspector score

Amazon Inspector creates a score for each Amazon EC2 finding. Amazon Inspector determines the score by correlating CVSS base score information with information from your compute environment, such as network reachability data and exploitability data. Amazon Inspector supports Amazon, Debian, and RHEL vendors. Each vendor provides a CVSS v3.1 base score. For other vendors, Amazon Inspector uses a CVSS base score provided by the National Vulnerability Database (NVD).

Due to FedRAMP requirements, Amazon Inspector uses the CVSS v3.1 base score as the default score. However, a CVSS 4.0 base score will be included in your vulnerability metadata when one is available. The CVSS 4.0 base score provides additional metrics to improve vulnerability assessment.

You can find the source and version of a CVSS base score in the vulnerability details for a finding and in exported findings.

> **Note**
>
> The Amazon Inspector score is not available for Linux instances running Ubuntu. Ubuntu uses a custom severity rating system that differs from CVSS scores.

Amazon Inspector score details

When you open the details page of a finding you can select the **Inspector score and vulnerability intelligence** Tab. This panel shows the difference between the base score and the **Inspector score**. This section explains how Amazon Inspector assigned the severity rating based on a combination of the Amazon Inspector score and the vendor score for the software package. If the scores differ this panel shows an explanation of why.

In the **CVSS score metrics** section you can see a table with comparisons between the CVSS base score metrics and the **Inspector score**. The metrics compared are the base metrics defined in the CVSS specification document maintained by first.org. The following is a summary of the base metrics:

Attack Vector

The context by which a vulnerability can be exploited. For Amazon Inspector findings this can be Network, **Adjacent Network**, or **Local**.

Attack Complexity

This describes the level of difficulty an attacker will face when exploiting the vulnerability. A **Low** score means that the attacker will need to meet little or no additional conditions to exploit the vulnerability. A **High** score means that an attacker will need invest a considerable amount of effort in order carry out a successful attack with this vulnerability.

Privilege Required

This describes the level of privilege an attacker will need to exploit a vulnerability.

User Interaction

This metric states if a successful attack using this vulnerability requires a human user, other than the attacker.

Scope

This states whether a vulnerability in one vulnerable component impacts resources in components beyond the vulnerable component's security scope. If this value is **Unchanged** the affected resource and the impacted resource are the same. If this value is **Changed** then the vulnerable component can be exploited to impact resources managed by different security authorities.

Confidentiality

This measures the level of impact to the confidentiality of data within a resource when the vulnerability is exploited. This ranges from **None**, where no confidentiality is lost, to **High** where all information within a resource is divulged or confidential information such as passwords or encryption keys can be divulged.

Integrity

This measures the level of impact to the integrity of data within the impacted resource if the vulnerability is exploited. Integrity is at risk when the attacker to modify files within impacted resources. The score ranges from **None**, where the exploit does not allow an attacker to modify any information, to **High**, where if exploited, the vulnerability would allow an attacker to modify any or all files, or the files that could be modified have serious consequences.

Availability

This measures the level of impact to the availability of the impacted resource when the vulnerability is exploited. The score ranges from **None**, when the vulnerability does not impact availability at all, to **High**, where if exploited, the attacker can completely deny availability to the resource, or cause a service to become unavailable.

Vulnerability Intelligence

This section summarizes available intelligence about the CVE from Amazon as well as industry standard security intelligence sources such as Cybersecurity and Infrastructure Security Agency (CISA).

> **Note**
>
> Intel from CISA or Amazon won't be available for all CVEs.

You can view vulnerability intelligence details in the console or by using the BatchGetFindingDetails API. The following details are available in the console:

ATT&CK

This section shows the MITRE tactics, techniques, and procedures (TTPs) associated with the CVE. The associated TTPs are shown, if there are more than two applicable TTPs you can select the link to see a complete list. Selecting a tactic or technique opens information about it on the MITRE website.

CISA

This section covers relevant dates associated with the vulnerability. The date Cybersecurity and Infrastructure Security Agency (CISA) added the vulnerability to Known Exploited Vulnerabilities Catalog, based on evidence of active exploitation, and the Due date CISA expects systems to be patched by. This information is sourced from CISA.

Known malware

This section lists known exploit kits and tools that exploit this vulnerability.

Last time reported

This section shows the Last known public exploit date for this vulnerability.

Understanding severity levels for your Amazon Inspector findings

When Amazon Inspector generates a finding, it assigns a severity rating to the finding. Severity ratings help you assess and prioritize your findings. The severity rating for a finding corresponds to a numerical score and level: **informational**, **low**, **medium**, **high**, and **critical**. Amazon Inspector determines the severity rating for a finding based on the finding type. This section describes how Amazon Inspector determines a severity rating for each finding type.

Software package vulnerability severity

Amazon Inspector uses the NVD/CVSS score as the basis of severity scoring for software package vulnerabilities. The NVD/CVSS score is the vulnerability severity score published by the NVD and defined by the CVSS. The NVD/CVSS score is a composition of security metrics, such as attack complexity, exploit code maturity, and privileges required. Amazon Inspector produces a numerical

score from 1 to 10 that reflects the vulnerability's severity. Amazon Inspector categorizes this as a base score because it reflects the severity of a vulnerability according to its intrinsic characteristics, which are constant over time. This score also assumes the reasonable worst-case impact across different deployed environments. The CVSS v3 standard maps CVSS scores to the following severity ratings.

Score	Rating
0	Informational
0.1–3.9	Low
4.0–6.9	Medium
7.0–8.9	High
9.0–10.0	Critical

Package vulnerability findings can also have a severity of **Untriaged**. This means that the vendor hasn't yet set a vulnerability score for the detected vulnerability. In this case, we recommend using the reference URLs for the finding to research that vulnerability and respond accordingly.

Package vulnerability findings include the following scores and associated scoring vectors as part of their finding details:

- EPSS score
- Inspector score
- CVSS 3.1 from Amazon CVE
- CVSS 3.1 from NVD
- CVSS 2.0 from NVD (where applicable)

Code vulnerability severity

For code vulnerability findings Amazon Inspector uses the severity levels defined by the Amazon Q detectors that generated the finding. Each detector is assigned a severity using the CVSS v3 scoring system. ?

Network reachability severity

Amazon Inspector determines the severity for a network reachability vulnerability based on the service, ports, and protocols that are exposed and by the type of open path. The following table defines these severity ratings. The value in the **Open path rating** column represents open paths from virtual gateways, peered VPCs, and AWS Direct Connect networks. All other exposed services, ports, and protocols have an Informational severity rating.

Service	TCP ports	UDP ports	Internet path rating	Open path rating
DHCP	67, 68, 546, 547	67, 68, 546, 547	Medium	Informational
Elasticsearch	9300, 9200	NA	Medium	Informational
FTP	21	21	High	Medium
Global catalog LDAP	3268	NA	Medium	Informational
Global catalog LDAP over TLS	3269	NA	Medium	Informational
HTTP	80	80	Low	Informational
HTTPS	443	443	Low	Informational
Kerberos	88, 464, 543, 544, 749, 751	88, 464, 749, 750, 751, 752	Medium	Informational
LDAP	389	389	Medium	Informational
LDAP over TLS	636	NA	Medium	Informational
MongoDB	27017, 27018, 27019, 28017	NA	Medium	Informational
MySQL	3306	NA	Medium	Informational
NetBIOS	137, 139	137, 138	Medium	Informational

NFS	111, 2049, 4045, 1110	111, 2049, 4045, 1110	Medium	Informational
Oracle	1521, 1630	NA	Medium	Informational
PostgreSQL	5432	NA	Medium	Informational
Print services	515	NA	High	Medium
RDP	3389	3389	Medium	Low
RPC	111, 135, 530	111, 135, 530	Medium	Informational
SMB	445	445	Medium	Informational
SSH	22	22	Medium	Low
SQL Server	1433	1434	Medium	Informational
Syslog	601	514	Medium	Informational
Telnet	23	23	High	Medium
WINS	1512, 42	1512, 42	Medium	Informational

Managing findings in Amazon Inspector

With Amazon Inspector, you can manage your findings in different ways. You can filter your findings based on their status. You can search your findings based on filter criteria. You can create suppression rules to exclude findings from your list of findings. You can also export findings to AWS Security Hub CSPM, Amazon EventBridge, and Amazon Simple Storage Service (Amazon S3).

Topics

- Filtering your Amazon Inspector findings
- Suppressing Amazon Inspector findings
- Exporting Amazon Inspector findings reports
- Creating custom responses to Amazon Inspector findings with Amazon EventBridge

Filtering your Amazon Inspector findings

You can filter your Amazon Inspector findings using filter criteria. If a finding doesn't match your filter criteria, Amazon Inspector excludes the finding from view. This section describes how to filter your Amazon Inspector findings using filter criteria.

Creating filters in the Amazon Inspector console

In each findings view, you can use the filter functionality to locate findings with specific characteristics. Filters are removed when you move to a different tabbed view.

A filter is made up of a filter criteria, which consists of a filter attribute paired with a filter value. Findings that do not match your filter criteria are excluded from the findings list. For example, to see all findings that are associated with your administrator account, you can choose the AWS account ID attribute and pair it with the value of your twelve digit AWS account ID.

Some filter criteria apply to all findings, while others are available for specific resource types or finding types only.

To apply a filter to the findings view

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.

2. In the navigation pane, choose **Findings**. The default view displays all findings with an **Active** status.
3. To filter findings by criteria, select the *Add filter* bar to see a list of all applicable filter criteria for that view. Different filter criteria are available in different views.
4. Choose a criterion that you want to filter by from the list.
5. From the criterion input pane enter the desired filter values to define that criterion.
6. Choose **Apply** to apply that filter criterion to your current results. You can continue to add other filter criterion by selecting the filter input bar again.
7. (Optional) To view your suppressed or closed findings, choose **Active** in the filter bar, and then choose **Suppressed** or **Closed**. Choose **Show all** to see active, suppressed, and closed findings in the same view.

Suppressing Amazon Inspector findings

You can create suppression rules to hide findings that match criteria. For example, you can create a suppression rule to hide findings based on their severity ratings. If Amazon Inspector generates a finding that matches your suppression rule, Amazon Inspector suppresses the finding and hides it from view. Amazon Inspector stores suppressed findings until they're remediated. Once a suppressed finding is remediated, Amazon Inspector closes the finding. You can view suppressed findings in the console.

You create suppression rules to prioritize your most important findings. Suppression rules don't have any impact on your findings, as they only hide findings from view. You cannot create a suppression rule that closes or remediates findings. You can also suppress unwanted findings in AWS Security Hub CSPM with an Amazon EventBridge rule. The procedures in this section describe how to create, view, edit, and delete a suppression rule.

> **Note**
>
> Only the delegated administrator for an organization can create and manage suppression rules.

Creating a suppression rule

You can create suppression rules to filter the list of findings that are shown by default. You can create a suppression rule programmatically by using the CreateFilter API and specifying SUPPRESS as the value for `action`.

> **Note**
>
> Only stand alone accounts and Amazon Inspector delegated administrators can create and manage suppression rules. Members in an organization will not see an option for suppression rules in the navigation pane.

To create a suppression rule (console)

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the navigation pane, choose **Suppression rules**. Then choose **Create rule**.
3. For each criterion, do the following:
   - Select the filter bar to see a list of filter criteria that you can add to your suppression rule.
   - Select the filter criteria for your suppression rule.
4. When you have finished adding criteria, enter a name for the rule and an optional description.
5. Choose **Save rule**. Amazon Inspector applies the new suppression rule and hides any findings that match the criteria. Note that suppression rules may take up to 24 hours to take effect.

Viewing suppressed findings

By default, Amazon Inspector does not display suppressed findings in the Amazon Inspector console. However, you can view the findings suppressed by a particular rule.

To view suppressed findings

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the navigation pane, select **Suppression rules**.
3. In the suppression rules list, select the title of the rule.

Editing a suppression rule

You can make changes to suppression rules at any time.

To modify suppression rules

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Suppression rules**.
3. Choose the name of the suppression rule that you want to change, and then choose **Edit**.
4. Make your intended changes, and then choose **Save**.

Deleting a suppression rule

You can delete suppression rules. If you delete a suppression rule, Amazon Inspector stops suppressing new and existing occurrences of findings that meet the rule criteria and that aren't suppressed by other rules.

After you delete a suppression rule, new and existing occurrences of findings that met the rule's criteria have a status of **Active**. This means that they appear by default on the Amazon Inspector console. In addition, Amazon Inspector publishes these findings to AWS Security Hub CSPM and Amazon EventBridge as events.

To delete a suppression rule

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the navigation pane, select **Suppression rules**.
3. Select the check box next to the title of the suppression rule you want to delete.
4. Choose **Delete**, and then confirm your choice to permanently delete the rule.

Exporting Amazon Inspector findings reports

A findings report is a CSV or JSON file that provides a detailed snapshot of your findings. You can export a findings report to AWS Security Hub CSPM, Amazon EventBridge, and Amazon Simple Storage Service (Amazon S3). When you configure a findings report, you specify which findings to

include in it. By default, your findings report includes data for all of your active findings. If you're the delegated administrator for an organization, your findings report includes data for all member accounts in the organization. To customize a findings report, create and apply a filter to it.

When you export a findings report, Amazon Inspector encrypts your findings data with an AWS KMS key that you specify. After Amazon Inspector encrypts your findings data, it stores your finding report in an Amazon S3 bucket that you specify. Your AWS KMS key must be used in the same AWS Region as your Amazon S3 bucket. Your AWS KMS key policy must allow Amazon Inspector to use it, and your Amazon S3 bucket policy must allow Amazon Inspector to add objects to it. After you export your findings report, you can download it from your Amazon S3 bucket or transfer it to new location. You can also use your Amazon S3 bucket as a repository for other exported findings reports.

This section describes how to export a findings report in the Amazon Inspector console. The following tasks require that you verify your permissions, configure an Amazon S3 bucket, configure an AWS KMS key, and configure and export a findings report.

> **Note**
>
> If you export a findings report with the Amazon Inspector CreateFindingsReport API, you can only view your active findings. If you want to view your suppressed or closed findings, you must specify SUPPRESSED or CLOSED as part of your filter criteria.

Tasks

- Step 1: Verify your permissions
- Step 2: Configure an S3 bucket
- Step 3: Configure an AWS KMS key
- Step 4: Configure and export a findings report
- Troubleshoot export errors

Step 1: Verify your permissions

> **Note**
>
> After you export a findings report for the first time, steps 1–3 are optional. Following these steps is based on whether you want to use the same Amazon S3 bucket and AWS

KMS key for other exported findings reports. If you want to export a findings report programmatically after completing steps 1–3, use the CreateFindingsReport operation of the Amazon Inspector API.

Before you export a findings report from Amazon Inspector, verify that you have the permissions that you need to both export findings reports and configure resources for encrypting and storing the reports. To verify your permissions, use AWS Identity and Access Management (IAM) to review the IAM policies that are attached to your IAM identity. Then compare the information in those policies to the following list of actions that you must be allowed to perform to export a findings report.

Amazon Inspector

For Amazon Inspector, verify that you're allowed to perform the following actions:

- `inspector2:ListFindings`
- `inspector2:CreateFindingsReport`

These actions allow you to retrieve findings data for your account and to export that data in findings reports.

If you plan to export large reports programmatically, you might also verify that you're allowed to perform the following actions: `inspector2:GetFindingsReportStatus`, to check the status of reports, and `inspector2:CancelFindingsReport`, to cancel exports that are in progress.

AWS KMS

For AWS KMS, verify that you're allowed to perform the following actions:

- `kms:GetKeyPolicy`
- `kms:PutKeyPolicy`

These actions allow you to retrieve and update the key policy for the AWS KMS key that you want Amazon Inspector to use to encrypt your report.

To use the Amazon Inspector console to export a report, also verify that you're allowed to perform the following AWS KMS actions:

- `kms:DescribeKey`
- `kms:ListAliases`

These actions allow you to retrieve and display information about the AWS KMS keys for your account. You can then choose one of these keys to encrypt your report.

If you plan to create a new KMS key for encryption of your report, you also need to be allowed to perform the `kms:CreateKey` action.

Amazon S3

For Amazon S3, verify that you're allowed to perform the following actions:

- `s3:CreateBucket`
- `s3:DeleteObject`
- `s3:PutBucketAcl`
- `s3:PutBucketPolicy`
- `s3:PutBucketPublicAccessBlock`
- `s3:PutObject`
- `s3:PutObjectAcl`

These actions allow you to create and configure the S3 bucket where you want Amazon Inspector to store your report. They also allow you to add and delete objects from the bucket.

If you plan to use the Amazon Inspector console to export your report, also verify that you're allowed to perform the `s3:ListAllMyBuckets` and `s3:GetBucketLocation` actions. These actions allow you to retrieve and display information about the S3 buckets for your account. You can then choose one of these buckets to store the report.

If you're not allowed to perform one or more of the required actions, ask your AWS administrator for assistance before you proceed to the next step.

Step 2: Configure an S3 bucket

After you verify your permissions, you're ready to configure the S3 bucket where you want to store your findings report. It can be an existing bucket for your own account, or an existing bucket that's owned by another AWS account and you're allowed to access. If you want to store your report in a new bucket, create the bucket before you proceed.

The S3 bucket must be in the same AWS Region as the findings data that you want to export. For example, if you're using Amazon Inspector in the US East (N. Virginia) Region and you want to export findings data for that Region, the bucket must also be in the US East (N. Virginia) Region.

In addition, the bucket's policy must allow Amazon Inspector to add objects to the bucket. This topic explains how to update the bucket policy and it provides an example of the statement to add to the policy. For detailed information about adding and updating bucket policies, see Using bucket policies in the *Amazon Simple Storage Service User Guide*.

If you want to store your report in an S3 bucket that's owned by another account, work with the bucket's owner to update the bucket's policy. Also obtain the URI for the bucket. You'll need to enter this URI when you export your report.

To update the bucket policy

1. Sign in using your credentials, and then open the Amazon S3 console at https://console.aws.amazon.com/s3.
2. In the navigation pane, choose **Buckets**.
3. Choose the S3 bucket where you want to store the findings report.
4. Choose the **Permissions** tab.
5. In the **Bucket policy** section, choose **Edit**.
6. Copy the following example statement to your clipboard:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "allow-inspector",
            "Effect": "Allow",
            "Principal": {
                "Service": "inspector2.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
```

```
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:inspector2:us-
east-1:111122223333:report/*"
                }
            }
        }
    ]
}
```

7. In the **Bucket policy** editor on the Amazon S3 console, paste the preceding statement into the policy to add it to the policy.

   When you add the statement, ensure that the syntax is valid. Bucket policies use JSON format. This means that you need to add a comma before or after the statement, depending on where you add the statement to the policy. If you add the statement as the last statement, add a comma after the closing brace for the preceding statement. If you add it as the first statement or between two existing statements, add a comma after the closing brace for the statement.

8. Update the statement with the correct values for your environment, where:

   - *amzn-s3-demo-bucket* is the name of the bucket.
   - *111122223333* is the account ID for your AWS account.
   - *Region* is the AWS Region in which you're using Amazon Inspector and want to allow Amazon Inspector to add reports to the bucket. For example, `us-east-1` for the US East (N. Virginia) Region.

   > **Note**
   >
   > If you're using Amazon Inspector in a manually enabled AWS Region, also add the appropriate Region code to the value for the `Service` field. This field specifies the Amazon Inspector service principal.
   > For example, if you're using Amazon Inspector in the Middle East (Bahrain) Region, which has the Region code `me-south-1`, replace `inspector2.amazonaws.com` with `inspector2.me-south-1.amazonaws.com` in the statement.

   Note that the example statement defines conditions that use two IAM global condition keys:

- aws:SourceAccount – This condition allows Amazon Inspector to add reports to the bucket only for your account. It prevents Amazon Inspector from adding reports to the bucket for other accounts. More specifically, the condition specifies which account can use the bucket for the resources and actions specified by the `aws:SourceArn` condition.

  To store reports for additional accounts in the bucket, add the account ID for each additional account to this condition. For example:

  ```
  "aws:SourceAccount": ["111122223333","444455556666","123456789012"]
  ```

- aws:SourceArn – This condition restricts access to the bucket based on the source of the objects that are being added to the bucket. It prevents other AWS services from adding objects to the bucket. It also prevents Amazon Inspector from adding objects to the bucket while performing other actions for your account. More specifically, the condition allows Amazon Inspector to add objects to the bucket only if the objects are findings reports, and only if those reports are created by the account and in the Region specified in the condition.

  To allow Amazon Inspector to perform the specified actions for additional accounts, add Amazon Resource Names (ARNs) for each additional account to this condition. For example:

  ```
  "aws:SourceArn": [
      "arn:aws:inspector2:Region:111122223333:report/*",
      "arn:aws:inspector2:Region:444455556666:report/*",
      "arn:aws:inspector2:Region:123456789012:report/*"
  ]
  ```

  The accounts specified by the `aws:SourceAccount` and `aws:SourceArn` conditions should match.

  Both conditions help prevent Amazon Inspector from being used as a confused deputy during transactions with Amazon S3. Although we don't recommend it, you can remove these conditions from the bucket policy.

9. When you finish updating the bucket policy, choose **Save changes**.

Step 3: Configure an AWS KMS key

After you verify your permissions and configure the S3 bucket, determine which AWS KMS key you want Amazon Inspector to use to encrypt your findings report. The key must be a customer managed, symmetric encryption KMS key. In addition, the key must be in the same AWS Region as the S3 bucket that you configured to store the report.

The key can be an existing KMS key from your own account, or an existing KMS key that another account owns. If you want to use a new KMS key, create the key before proceeding. If you want to use an existing key that another account owns, obtain the Amazon Resource Name (ARN) of the key. You'll need to enter this ARN when you export your report from Amazon Inspector. For information about creating and reviewing the settings for KMS keys, see Managing keys in the *AWS Key Management Service Developer Guide*.

After you determine which KMS key you want to use, give Amazon Inspector permission to use the key. Otherwise, Amazon Inspector won't be able to encrypt and export the report. To give Amazon Inspector permission to use the key, update the key policy for the key. For detailed information about key policies and managing access to KMS keys, see Key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> The following procedure is for updating an existing key to allow Amazon Inspector to use it. If you don't have an existing key, see Creating keys in the *AWS Key Management Service Developer Guide*.

To update the key policy

1. Sign in using your credentials, and then open the AWS KMS console at https://console.aws.amazon.com/kms.
2. In the navigation pane, choose **Customer managed keys**.
3. Choose the KMS key that you want to use to encrypt the report. The key must be a symmetric encryption (**SYMMETRIC_DEFAULT**) key.
4. On the **Key policy** tab, choose **Edit**. If you do not see a key policy with an **Edit** button, you must first select **Switch to policy view**.
5. Copy the following example statement to your clipboard:

```
{
    "Sid": "Allow Amazon Inspector to use the key",
    "Effect": "Allow",
    "Principal": {
        "Service": "inspector2.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:inspector2:Region:111122223333:report/*"
        }
    }
}
```

6. In the **Key policy** editor on the AWS KMS console, paste the preceding statement into the key policy to add it to the policy.

   When you add the statement, ensure that the syntax is valid. Key policies use JSON format. This means that you need to add a comma before or after the statement, depending on where you add the statement to the policy. If you add the statement as the last statement, add a comma after the closing brace for the preceding statement. If you add it as the first statement or between two existing statements, add a comma after the closing brace for the statement.

7. Update the statement with the correct values for your environment, where:

   - *111122223333* is the account ID for your AWS account.
   - *Region* is the AWS Region in which you want to allow Amazon Inspector to encrypt reports with the key. For example, `us-east-1` for the US East (N. Virginia) Region.

   Note

   If you're using Amazon Inspector in a manually enabled AWS Region, also add the appropriate Region code to the value for the `Service` field. For example,

if you're using Amazon Inspector in the Middle East (Bahrain) Region, replace `inspector2.amazonaws.com` with `inspector2.me-south-1.amazonaws.com`.

Like the example statement for the bucket policy in the preceding step, the `Condition` fields in this example use two IAM global condition keys:

- aws:SourceAccount – This condition allows Amazon Inspector to perform the specified actions only for your account. More specifically, it determines which account can perform the specified actions for the resources and actions specified by the `aws:SourceArn` condition.

  To allow Amazon Inspector to perform the specified actions for additional accounts, add the account ID for each additional account to this condition. For example:

  ```
  "aws:SourceAccount": ["111122223333","444455556666","123456789012"]
  ```

- aws:SourceArn – This condition prevents other AWS services from performing the specified actions. It also prevents Amazon Inspector from using the key while performing other actions for your account. In other words, it allows Amazon Inspector to encrypt S3 objects with the key only if the objects are findings reports, and only if those reports are created by the account and in the Region specified in the condition.

  To allow Amazon Inspector to perform the specified actions for additional accounts, add ARNs for each additional account to this condition. For example:

  ```
  "aws:SourceArn": [
      "arn:aws:inspector2:us-east-1:111122223333:report/*",
      "arn:aws:inspector2:us-east-1:444455556666:report/*",
      "arn:aws:inspector2:us-east-1:123456789012:report/*"
  ]
  ```

  The accounts specified by the `aws:SourceAccount` and `aws:SourceArn` conditions should match.

These conditions help prevent Amazon Inspector from being used as a confused deputy during transactions with AWS KMS. Although we don't recommend it, you can remove these conditions from the statement.

8. When you finish updating the key policy, choose **Save changes**.

Step 4: Configure and export a findings report

> **Note**
>
> You only can export only one findings report a time. If an export is currently in progress, you must wait until the export is complete before exporting another findings report.

After you verify your permissions and you configure resources to encrypt and store your findings report, you're ready to configure and export the report.

To configure and export a findings report

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the navigation pane, under **Findings**, choose **All findings**.
3. (Optional) By using the filter bar above the **Findings** table, add filter criteria that specify which findings to include in the report. As you add criteria, Amazon Inspector updates the table to include only those findings that match the criteria. The table provides a preview of the data that your report will contain.

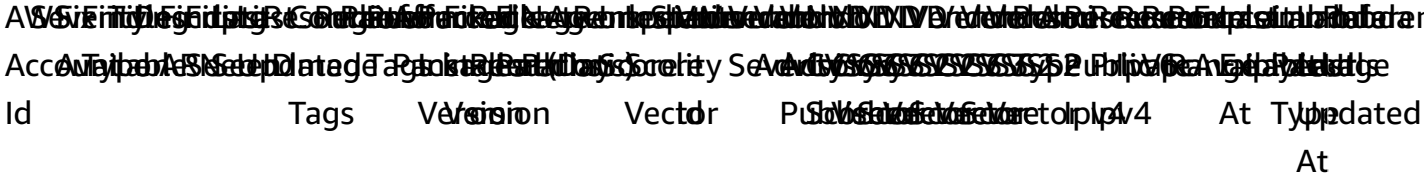
   > **Note**
   >
   > We recommend that you add filter criteria. If you don't, the report will include data for all of your findings in the current AWS Region that have a status of **Active**. If you're the Amazon Inspector administrator for an organization, this includes findings data for all the member accounts in your organization.
   > If a report includes data for all or many findings, it can take a long time to generate and export the report, and you can export only one report at a time.

4. Choose **Export findings**.
5. In the **Export settings** section, for **Export file type**, specify a file format for the report:

   - To create a JavaScript Object Notation (.json) file that contains the data, choose **JSON**.

     If you choose the **JSON** option, the report will include all the fields for each finding. For a list of possible JSON fields see the Finding data type in the Amazon Inspector API reference.
   - To create a comma-separated values (.csv) file that contains the data, choose **CSV**.

If you choose the **CSV** option, the report will include only a subset of the fields for each finding, approximately 45 fields that report key attributes of a finding. The fields include: *Finding Type, Title, Severity, Status, Description, First Seen, Last Seen, Fix Available, AWS account ID, Resource ID, Resource Tags*, and *Remediation*. These are in addition to fields that capture scoring details and reference URLs for each finding. The following is a sample of the CSV headers in a findings report:



6. Under **Export location**, for **S3 URI**, specify the S3 bucket where you want to store the report:

   - To store the report in a bucket that your account owns, choose **Browse S3**. Amazon Inspector displays a table of the S3 buckets for your account. Select the row for the bucket that you want, and then choose **Choose**.

     Tip

     To also specify an Amazon S3 path prefix for the report, append a slash (/) and the prefix to the value in the **S3 URI** box. Amazon Inspector then includes the prefix when it adds the report to the bucket, and Amazon S3 generates the path specified by the prefix.
     For example, if you want to use your AWS account ID as a prefix and your account ID is *111122223333*, append **/111122223333** to the value in the **S3 URI** box.
     A *prefix* is similar to a directory path within an S3 bucket. It allows you to group similar objects together in a bucket, much like you might store similar files together in a folder on a file system. For more information, see Organizing objects in the Amazon S3 console using folders in the *Amazon Simple Storage Service User Guide*.

   - To store the report in a bucket that another account owns, enter the URI for the bucket—for example, **`s3://DOC-EXAMPLE_BUCKET`**, where *DOC-EXAMPLE_BUCKET* is the name of the bucket. The bucket owner can find this information for you in the bucket's properties.

7. For **KMS key**, specify the AWS KMS key that you want to use to encrypt the report:

- To use a key from your own account, choose the key from the list. The list displays customer managed, symmetric encryption KMS keys for your account.
- To use a key that another account owns, enter the Amazon Resource Name (ARN) of the key. The key owner can find this information for you in the key's properties. For more information, see Finding the key ID and key ARN in the *AWS Key Management Service Developer Guide*.

8. Choose **Export**.

Amazon Inspector generates the findings report, encrypts it with the KMS key that you specified, and adds it to the S3 bucket that you specified. Depending on the number of findings that you chose to include in the report, this process can take several minutes or hours. When the export is complete, Amazon Inspector displays a message indicating that your findings report was exported successfully. Optionally choose **View report** in the message to navigate to the report in Amazon S3.

> **Note**
>
> Report generation times out if it takes longer than 12 hours to complete. If your report times out, add filter criteria to reduce the number of findings included in the report, and then export the report again.

Note that you can export only one report a time. If an export is currently in progress, wait until that export is complete before you try to export another report.

Troubleshoot export errors

If an error occurs when you try to export a findings report, Amazon Inspector displays a message describing the error. You can use the information in this topic as a guide to identify possible causes and solutions for the error.

For example, verify that the S3 bucket is in the current AWS Region and the bucket's policy allows Amazon Inspector to add objects to the bucket. Also verify that the AWS KMS key is enabled in the current Region, and ensure that the key policy allows Amazon Inspector to use the key.

After you address the error, try to export the report again.

Cannot have multiple reports error

If you are attempting to create a report but Amazon Inspector is already generating a report, you will receive an error stating **Reason: Cannot have multiple reports in-progress**. This error occurs because Amazon Inspector can only generate one report for an account at a time.

To resolve the error you can wait for the other report to finish or cancel it before requesting a new report.

You can check the status of a report by using the GetFindingsReportStatus operation, this operation returns the report ID of any report that is currently being generated.

If you need to, you can use the report ID given by the `GetFindingsReportStatus` operation to cancel a export that is currently in progress by using the CancelFindingsReport operation.

Creating custom responses to Amazon Inspector findings with Amazon EventBridge

Amazon Inspector creates an event in Amazon EventBridge for newly generated findings and aggregated findings. Amazon Inspector also creates an event for any changes to the state of a finding. This means Amazon Inspector creates a new event for a finding when you take actions like restarting a resource or changing tags associated with a resource. When Amazon Inspector creates a new event for an updated finding, the finding `id` stays the same.

> **Note**
>
> If your account is an Amazon Inspector delegated administrator account, EventBridge publishes events to your account and the member account where the events originated.

When using EventBridge events with Amazon Inspector, you can automate tasks to help you respond to security issues your findings reveal. To receive notifications about Amazon Inspector findings based on EventBridge events, you must create an EventBridge rule and specify a target for Amazon Inspector. The EventBridge rule allows EventBridge to send notifications for Amazon Inspector findings, and the target specifies where to send the notifications.

Amazon Inspector emits events to the default event bus in the AWS Region where you are currently using Amazon Inspector. This means you must configure event rules for each AWS Region where

you activated Amazon Inspector and configured Amazon Inspector to receive EventBridge events. Amazon Inspector emits events on a best-effort basis.

This section provides you with an example of an event schema and describes how to create an EventBridge rule.

Event schema

The following is an example of the Amazon Inspector event format for an EC2 finding event. For example schema of other finding types and event types, see *EventBridge schema*.


```
{
    "version": "0",
    "id": "66a7a279-5f92-971c-6d3e-c92da0950992",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2023-01-19T22:46:15Z",
    "region": "us-east-1",
    "resources": ["i-0c2a343f1948d5205"],
    "detail": {
        "awsAccountId": "111122223333",
        "description": "\n It was discovered that the sound subsystem in the Linux
 kernel contained a\n race condition in some situations. A local attacker could use
 this to cause\n a denial of service (system crash).",
        "exploitAvailable": "YES",
        "exploitabilityDetails": {
            "lastKnownExploitAt": "Oct 24, 2022, 11:08:59 PM"
        },
        "findingArn": "arn:aws:inspector2:us-east-1:111122223333:finding/FINDING_ID",
        "firstObservedAt": "Jan 19, 2023, 10:46:15 PM",
        "fixAvailable": "YES",
        "lastObservedAt": "Jan 19, 2023, 10:46:15 PM",
        "packageVulnerabilityDetails": {
            "cvss": [{
                "baseScore": 4.7,
                "scoringVector": "CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:H",
                "source": "NVD",
                "version": "3.1"
            }],
            "referenceUrls": ["https://lore.kernel.org/all/
CAFcO6XN7JDM4xSXGhtusQfS2mSBcx50VJKwQpCq=WeLt57aaZA@mail.gmail.com/", "https://
```

```
ubuntu.com/security/notices/USN-5792-1", "https://ubuntu.com/security/notices/
USN-5791-2", "https://ubuntu.com/security/notices/USN-5791-1", "https://ubuntu.com/
security/notices/USN-5793-2", "https://git.kernel.org/pub/scm/linux/kernel/git/
torvalds/linux.git/commit/?id=8423f0b6d513b259fdab9c9bf4aaa6188d054c2d", "https://
ubuntu.com/security/notices/USN-5793-1", "https://ubuntu.com/security/notices/
USN-5792-2", "https://ubuntu.com/security/notices/USN-5791-3", "https://ubuntu.com/
security/notices/USN-5793-4", "https://ubuntu.com/security/notices/USN-5793-3",
 "https://git.kernel.org/linus/8423f0b6d513b259fdab9c9bf4aaa6188d054c2d(6.0-rc5)",
 "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-3303"],
            "relatedVulnerabilities": [],
            "source": "UBUNTU_CVE",
            "sourceUrl": "https://people.canonical.com/~ubuntu-security/cve/2022/
CVE-2022-3303.html",
            "vendorCreatedAt": "Sep 27, 2022, 11:15:00 PM",
            "vendorSeverity": "medium",
            "vulnerabilityId": "CVE-2022-3303",
            "vulnerablePackages": [{
                "arch": "X86_64",
                "epoch": 0,
                "fixedInVersion": "0:5.15.0.1027.31~20.04.16",
                "name": "linux-image-aws",
                "packageManager": "OS",
                "remediation": "apt update && apt install --only-upgrade linux-image-
aws",
                "version": "5.15.0.1026.30~20.04.16"
            }]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
            }
        },
        "resources": [{
            "details": {
                "awsEc2Instance": {
                    "iamInstanceProfileArn": "arn:aws:iam::111122223333:instance-
profile/AmazonSSMRoleForInstancesQuickSetup",
                    "imageId": "ami-0b7ff1a8d69f1bb35",
                    "ipV4Addresses": ["172.31.85.212", "44.203.45.27"],
                    "ipV6Addresses": [],
                    "launchedAt": "Jan 19, 2023, 7:53:14 PM",
                    "platform": "UBUNTU_20_04",
                    "subnetId": "subnet-8213f2a3",
                    "type": "t2.micro",
```

```
                    "vpcId": "vpc-ab6650d1"
                }
            },
            "id": "i-0c2a343f1948d5205",
            "partition": "aws",
            "region": "us-east-1",
            "type": "AWS_EC2_INSTANCE"
        }],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2022-3303 - linux-image-aws",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Jan 19, 2023, 10:46:15 PM"
    }
}
```


Creating an EventBridge rule to notify you of Amazon Inspector findings

To increase the visibility of Amazon Inspector findings, you can use EventBridge to set up automated finding alerts that are sent to a messaging hub. This topic shows you how to send alerts for `CRITICAL` and `HIGH` severity findings to email, Slack, or Amazon Chime. You'll learn how to set up an Amazon Simple Notification Service topic and then connect that topic to an EventBridge event rule.

Step 1. Set up an Amazon SNS topic and endpoint

To set up automatic alerts, you must first set up a topic in Amazon Simple Notification Service and add an endpoint. For more information, refer to the SNS guide.

This procedure establishes where you want to send Amazon Inspector findings data. The SNS topic can be added to an EventBridge event rule during or after the creation of the event rule.

Email setup

Creating an SNS topic

1. Sign in to the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. From the navigation pane, select **Topics**, and then select **Create Topic**.

3. In the **Create topic** section, select **Standard**. Next, enter a topic name, such as **`Inspector_to_Email`**. Other details are optional.
4. Choose **Create Topic**. This opens a new panel with details for your new topic.
5. In the **Subscriptions** section, select **Create Subscription**.
6. a. From the **Protocol** menu, select **Email**.

   b. In the **Endpoint** field, enter the email address that you would like to receive notifications.

   > **Note**
   >
   > You will be required to confirm your subscription through your email client after creating the subscription.

   c. Choose **Create subscription**.
7. Look for a subscription message in your inbox and choose **Confirm Subscription**.

Slack setup

Creating an SNS topic

1. Sign in to the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. From the navigation pane, select **Topics**, and then select **Create Topic**.
3. In the **Create topic** section, select **Standard**. Next, enter a topic name, such as **`Inspector_to_Slack`**. Other details are optional. Choose **Create topic** to complete endpoint creation.

Configuring an Amazon Q Developer in chat applications client

1. Navigate to the Amazon Q Developer in chat applications console at https://console.aws.amazon.com/chatbot/.
2. From the **Configured clients** pane, select **Configure new client**.
3. Choose **Slack**, and then choose **Configure** to confirm.

> **Note**
>
> When choosing Slack, you must confirm permissions for Amazon Q Developer in chat applications to access your channel by selecting **allow**.

4. Select **Configure new channel** to open the configuration details pane.
   a. Enter a name for the channel.
   b. For **Slack channel**, choose the channel that you want to use.
   c. In Slack, copy the channel ID of the private channel by right-clicking on the channel name and selecting **Copy Link**.
   d. On the AWS Management Console, in the Amazon Q Developer in chat applications window, paste the channel ID that you copied from Slack into the **Private channel ID** field.
   e. In **Permissions**, choose to create an IAM role using a template if you do not already have a role.
   f. For **Policy** templates, choose **Notification permissions**. This is the IAM policy template for Amazon Q Developer in chat applications. This policy provides the necessary read and list permissions for CloudWatch alarms, events, and logs, and for Amazon SNS topics.
   g. For **Channel guardrail policies**, choose **AmazonInspector2ReadOnlyAccess**.
   h. Choose the Region in which you previously created your SNS topic, and then select the Amazon SNS topic you created to send notifications to the Slack channel.
5. Select **Configure**.

Amazon Chime setup

Creating an SNS topic

1. Sign in to the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. Select **Topics** from the navigation pane, and then select **Create Topic**.
3. In the **Create topic** section, select **Standard**. Next, enter a topic name, such as **`Inspector_to_Chime`**. Other details are optional. Choose **Create topic** to complete.

Configuring an Amazon Q Developer in chat applications client

1. Navigate to the Amazon Q Developer in chat applications console at https://console.aws.amazon.com/chatbot/.
2. From the **Configured clients** panel, select **Configure new client**.
3. Choose **Chime**, and then choose **Configure** to confirm.
4. From the **Configuration details** pane, enter a name for the channel.
5. In Amazon Chime, open the desired chat room.
   a. Choose the gear icon in the upper-right corner and choose **Manage webhooks and bots**.
   b. Select **Copy URL** to copy the webhook URL to your clipboard.
6. On the AWS Management Console, in the Amazon Q Developer in chat applications window, paste the URL you copied into the **Webhook URL** field.
7. In **Permissions**, choose to create an IAM role using a template if you do not already have a role.
8. For **Policy** templates, choose **Notification permissions**. This is the IAM policy template for Amazon Q Developer in chat applications. It provides the necessary read and list permissions for CloudWatch alarms, events, and logs, and for Amazon SNS topics.
9. Choose the Region in which you previously created your SNS topic, and then select the Amazon SNS topic you created to send notifications to the Amazon Chime room.
10. Select **Configure**.

Step 2. Create an EventBridge rule for Amazon Inspector findings

1. Sign in using your credentials.
2. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
3. Select **Rules** from the navigation pane, and then select **Create rule**.
4. Enter a name and optional description for your rule.
5. Select **Rule with an event pattern** and then **Next**.
6. In the **Event Pattern** pane, choose **Custom patterns (JSON editor)**.
7. Paste the following JSON into the editor.

```
{
```

```
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Finding"],
  "detail": {
    "severity": ["HIGH", "CRITICAL"],
    "status": ["ACTIVE"]
  }
}
```

> **Note**
>
> This pattern sends notifications for any active `CRITICAL` or `HIGH` severity finding detected by Amazon Inspector.

Select **Next** when you are finished entering the event pattern.

8. On the **Select targets** page, choose **AWS service**. Then, for **Select target type**, choose **SNS topic**.
9. For **Topic**, select the name of the SNS topic you created in step 1. Then choose **Next**.
10. Add optional tags if needed and choose **Next**.
11. Review your rule and then choose **Create rule**.

EventBridge for Amazon Inspector multi-account environments

If you're an Amazon Inspector delegated administrator, EventBridge rules appear on your account based on applicable findings from your member accounts. If you set up findings notifications through EventBridge in your administrator account, as detailed in the preceding section, you'll receive notifications about multiple accounts. In other words, you'll be notified of findings and events generated by your member accounts in addition to those generated by your own account.

You can use the `accountId` from the finding's JSON details to identify the member account from which the Amazon Inspector finding originated.

Working with the dashboard in Amazon Inspector

The dashboard provides a snapshot of aggregated statistics for resources that Amazon Inspector scans. Use the dashboard to learn about coverage for your environment and critical findings.

> **Note**
>
> If your account is the delegated administrator account for an organization, the dashboard shows information for your account and every other account in the organization.

This topic describes how to view the dashboard and understand the components that make up the dashboard.

Topics

- Viewing the dashboard
- Understanding dashboard components and interpreting data

Viewing the dashboard

The dashboard shows an overview of the coverage for your environment and critical findings. The dashboard refreshes data automatically every five minutes. You can refresh data manually by choosing the refresh icon near the top-right corner of the screen. You can view supporting data for an item by choosing the item.

> **Note**
>
> If your account is the delegated administrator account for an organization, you can view aggregated statistics for a member account by entering the member account ID in the **Account** field.

To view the dashboard:

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Dashboard**.

Understanding dashboard components and interpreting data

Each section of the dashboard provides insight into key metrics and findings data, so you can understand the vulnerability posture of your AWS resources in your current AWS Region.

Environment coverage

The **Environment coverage** section provides statistics about the resources scanned by Amazon Inspector. In this section, you can see the count and percentage of Amazon EC2 instances, Amazon ECR images and AWS Lambda functions scanned by Amazon Inspector. If you manage multiple accounts through AWS Organizations as an Amazon Inspector delegated administrator, you will also see the total number of organization accounts, the number with Amazon Inspector activated, and the resulting coverage percentage for the organization. You can also use this section to determine which resources are not covered by Amazon Inspector. These resources may contain vulnerabilities that could be exploited to put your organization at risk. For more details, see Assessing Amazon Inspector coverage of your AWS environment.

Choosing a coverage group takes you to the **Account management** page for the grouping you select. The account management page shows you details about which accounts, Amazon EC2 instances, and Amazon ECR repositories are covered by Amazon Inspector.

The following coverage groups are available:

- Account
- Instances
- Container repositories
- Container images
- Lambda

Critical findings

The **Critical findings** section provides a count of the critical vulnerabilities in your environment and a total count of all findings in your environment. In this section, the counts are shown per resource and assessment type. For more information about critical findings and how Amazon Inspector determines criticality, see Understanding Amazon Inspector findings.

Choosing a critical finding group takes you to the **All findings** page and automatically applies filters to show all critical findings that match the grouping you selected.

The following critical finding groups are available:

- Amazon Inspector code scan findings
- Amazon EC2 instance findings
- Amazon ECR container image findings
- Lambda function findings

Risk-based remediations

The **Risk-based remediations** section shows the top five software packages with critical vulnerabilities that affect the most resources in your environment. Remediating these packages can significantly reduce the number of critical risks to your environment. Choose the software package name to see associated vulnerability details and affected resources.

Accounts with the most critical findings

The **Accounts with the most critical findings** section shows the top five AWS accounts in your environment with the most critical findings, and the total number of findings for that account. This section is only viewable from the delegated administrator account when Amazon Inspector is configured for multi-account scanning with AWS Organizations. This view helps delegated administrators understand which accounts may be most at risk within the organization.

Choose **Account ID** to see more information about the affected member account.

Amazon ECR repositories with most critical findings

The **Elastic Container Registry (ECR) Repositories with most critical findings** section shows the top five Amazon ECR repositories in your environment with the most critical container image findings. The view shows the repository name, AWS account identifier, the repository creation date, number of critical vulnerabilities, and total number of vulnerabilities. This view helps you identify which repositories may be most at risk.

Choose **Repository name** to see more information about the affected repository.

Container images with most critical findings

The **Container images with most critical findings** section shows the top five container images in your environment with the most critical findings. The view shows image tag data, repository name, image digest, AWS account identifier, number of critical vulnerabilities, and total number of vulnerabilities. This view helps application owners identify which container images may need to be rebuilt and relaunched.

Choose **Container image** to see more information about the affected container image.

Instances with most critical findings

The **Instances with most critical findings** section shows the top five Amazon EC2 instances with the most critical findings. The view shows instance identifier, AWS account identifier, Amazon Machine Image (AMI) identifier, number of critical vulnerabilities, and total number of vulnerabilities. This view helps infrastructure owners identify which instances may require patching.

Choose **Instance ID** to see more information about the affected Amazon EC2 instance.

Amazon Machine Images (AMI) with most critical findings

The **Amazon Machine Images (AMIs) with most critical findings** section shows the top five AMIs in your environment with the most critical findings. The view shows the AMI identifier, AWS account identifier, number of affected EC2 instances running in the environment, the AMI creation date, the operating system platform of the AMI, the number of critical vulnerabilities, and the total number of vulnerabilities. This view helps infrastructure owners identify which AMIs may require rebuilding.

Choose **Affected instances** to see more information about the instances launched from the affected AMI.

AWS Lambda functions with most critical findings

The **AWS Lambda functions with most critical findings** section shows the top five Lambda functions in your environment with the most critical findings. The view shows the Lambda function name, AWS account identifier, runtime environment, the number of critical vulnerabilities, the number of high vulnerabilities, and the total number of vulnerabilities. This view helps infrastructure owners identify which Lambda functions may require remediation.

Choose **Function name** to see more information about the affected AWS Lambda function.

Amazon Inspector code scans with the most critical findings

The **Projects with the most critical code vulnerabilities** section shows the top five projects with critical findings. You can choose a project to view details about the findings. When you choose a project, you're directed to the repository where the findings are located. The findings tab shows the names of your findings and their severity ratings. It shows what type of analysis was used to generate your findings. It also shows how old your findings are and their statuses.

Searching the Amazon Inspector vulnerability database

You can search the Amazon Inspector vulnerability database for common vulnerabilities and exposures (CVE). Amazon Inspector uses information from the vulnerability database to produce details related to a CVE ID. You can view these details on the CVE details screen. Amazon Inspector tracks and produces findings for software vulnerabilities in the vulnerability database. Amazon Inspector only supports CVEs with platforms listed in the **Detection Platforms** section of the CVE details screen. This section describes how to search the Amazon Inspector vulernability database using a CVE ID.

> **Note**
>
> Currently, CVE search doesn't support Microsoft Windows.

Searching the vulnerability database

This section describes how to search the vulnerability database in the console and with the Amazon Inspector API.

> **Note**
>
> You must activate Amazon Inspector in your current AWS Region before you can search the vulnerability database.

Console

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home
2. From the navigation pane, choose **Vulnerability database search**.
3. In the search bar, enter a CVE ID, and choose **Search**.

API

Run the Amazon Inspector SearchVulnerabilities API, and provide a single CVE ID as `filterCriteria` in the following format: `CVE-<year>-<ID>`.

Understanding CVE details

This section descibes how to interpet the CVE details page.

CVE details

The CVE details section includes the following information:

- CVE description and ID
- CVE Severity
- Common Vulnerability Scoring System (CVSS) and Exploit Prediction Scoring System (EPSS) scores
- Detection platforms

> **Note**
>
> If this field is empty, Amazon Inspector doesn't support detection for your CVE ID.

- Common Weakness Enumeration (CWE)
- Vendor created and updated dates

Vulnerability intelligence

The vulnerability intelligence section provides threat intelligence data like exploit targets and the last known public exploit date.

It also provides data from the Cybersecurity and Infrastructure Security Agency (CISA), which includes the remediation action, date the CVE was added to the Known Exploited Vulnerability catalog, and date time CISA expects federal agencies to remediate the CVE.

References

The references section provides links to resources for more information about the CVE.

Exporting SBOMs with Amazon Inspector

A software bill of materials (SBOM) is a nested inventory of all the open-source and third-party software components in your codebase. Amazon Inspector provides SBOMs for individual resources in your environment. You can use the Amazon Inspector console or Amazon Inspector API to generate SBOMs for your resources. You can export SBOMs for all resources that Amazon Inspector supports and monitors. Exported SBOMs provide information about your software supply. You can review the status of your resources by assessing the coverage of your AWS environment. This section describes how to configure and export SBOMs.

Some software components and package managers use version ranges or dynamic references instead of fixed versions for dependencies. This practice creates unresolved hashes, where Amazon Inspector identifies a hash or jar file but cannot map it to a specific name and version for vulnerability detection. Amazon Inspector now includes these unresolved hashes in Software Bill of Materials (SBOM) exports. While these packages cannot be scanned for vulnerabilities, their hash values are available within the exported components list.

Tip

As of February 28, 2026, Amazon Inspector now supports exporting SBOMs for agentless Amazon EC2 Windows instances. Exporting SBOMs for agent-based Amazon EC2 Windows instances is currently not supported.

Amazon Inspector formats

Amazon Inspector supports exporting SBOMs in **CycloneDX 1.4** and **SPDX 2.3** compatible formats. Amazon Inspector exports SBOMs as `JSON` files to the Amazon S3 bucket you choose.

Note

SPDX format exports from Amazon Inspector are compatible with systems using **SPDX 2.3**, however they don't contain the Creative Commons Zero (CC0) field. This is because including this field would allow users to redistribute or edit the material.

Example of CycloneDX 1.4 SBOM format from Amazon Inspector


```
                    {
  "bomFormat": "CycloneDX",
  "specVersion": "1.4",
  "version": 1,
  "metadata": {
    "timestamp": "2023-06-02T01:17:46Z",
    "component": null,
    "properties": [
      {
        "name": "imageId",
        "value":
 "sha256:c8ee97f7052776ef223080741f61fcdf6a3a9107810ea9649f904aa4269fdac6"
      },
      {
        "name": "architecture",
        "value": "arm64"
      },
      {
        "name": "accountId",
        "value": "111122223333"
      },
      {
        "name": "resourceType",
        "value": "AWS_ECR_CONTAINER_IMAGE"
      }
    ]
  },
  "components": [
    {
      "type": "library",
      "name": "pip",
      "purl": "pkg:pypi/pip@22.0.4?path=usr/local/lib/python3.8/site-packages/
pip-22.0.4.dist-info/METADATA",
      "bom-ref": "98dc550d1e9a0b24161daaa0d535c699"
    },
    {
      "type": "application",
      "name": "libss2",
      "purl": "pkg:dpkg/libss2@1.44.5-1+deb10u3?
arch=ARM64&epoch=0&upstream=libss2-1.44.5-1+deb10u3.src.dpkg",
```

```
      "bom-ref": "2f4d199d4ef9e2ae639b4f8d04a813a2"
    },
    {
      "type": "application",
      "name": "liblz4-1",
      "purl": "pkg:dpkg/liblz4-1@1.8.3-1+deb10u1?
arch=ARM64&epoch=0&upstream=liblz4-1-1.8.3-1+deb10u1.src.dpkg",
      "bom-ref": "9a6be8907ead891b070e60f5a7b7aa9a"
    },
    {
      "type": "application",
      "name": "mawk",
      "purl": "pkg:dpkg/mawk@1.3.3-17+b3?
arch=ARM64&epoch=0&upstream=mawk-1.3.3-17+b3.src.dpkg",
      "bom-ref": "c2015852a729f97fde924e62a16f78a5"
    },
    {
      "type": "application",
      "name": "libgmp10",
      "purl": "pkg:dpkg/libgmp10@6.1.2+dfsg-4+deb10u1?
arch=ARM64&epoch=2&upstream=libgmp10-6.1.2+dfsg-4+deb10u1.src.dpkg",
      "bom-ref": "52907290f5beef00dff8da77901b1085"
    },
    {
      "type": "application",
      "name": "ncurses-bin",
      "purl": "pkg:dpkg/ncurses-bin@6.1+20181013-2+deb10u3?
arch=ARM64&epoch=0&upstream=ncurses-bin-6.1+20181013-2+deb10u3.src.dpkg",
      "bom-ref": "cd20cfb9ebeeadba3809764376f43bce"
    }
  ],
  "vulnerabilities": [
    {
      "id": "CVE-2022-40897",
      "affects": [
        {
          "ref": "a74a4862cc654a2520ec56da0c81cdb3"
        },
        {
          "ref": "0119eb286405d780dc437e7dbf2f9d9d"
        }
      ]
    }
  ]
```

```
}
```

Example of SPDX 2.3 SBOM format from Amazon Inspector

```
{
 "name": "409870544328/EC2/i-022fba820db137c64/ami-074ea14c08effb2d8",
 "spdxVersion": "SPDX-2.3",
 "creationInfo": {
  "created": "2023-06-02T21:19:22Z",
  "creators": [
   "Organization: 409870544328",
   "Tool: Amazon Inspector SBOM Generator"
  ]
 },
 "documentNamespace": "EC2://i-022fba820db137c64/AMAZON_LINUX_2/null/x86_64",
 "comment": "",
 "packages": [{
   "name": "elfutils-libelf",
   "versionInfo": "0.176-2.amzn2",
   "downloadLocation": "NOASSERTION",
   "sourceInfo": "/var/lib/rpm/Packages",
   "filesAnalyzed": false,
   "externalRefs": [{
    "referenceCategory": "PACKAGE-MANAGER",
    "referenceType": "purl",
    "referenceLocator": "pkg:rpm/elfutils-libelf@0.176-2.amzn2?
arch=X86_64&epoch=0&upstream=elfutils-libelf-0.176-2.amzn2.src.rpm"
   }],
   "SPDXID": "SPDXRef-Package-rpm-elfutils-libelf-ddf56a513c0e76ab2ae3246d9a91c463"
  },
  {
   "name": "libcurl",
   "versionInfo": "7.79.1-1.amzn2.0.1",
   "downloadLocation": "NOASSERTION",
   "sourceInfo": "/var/lib/rpm/Packages",
   "filesAnalyzed": false,
   "externalRefs": [{
     "referenceCategory": "PACKAGE-MANAGER",
     "referenceType": "purl",
     "referenceLocator": "pkg:rpm/libcurl@7.79.1-1.amzn2.0.1?
arch=X86_64&epoch=0&upstream=libcurl-7.79.1-1.amzn2.0.1.src.rpm"
```

```
    },
    {
     "referenceCategory": "SECURITY",
     "referenceType": "vulnerability",
     "referenceLocator": "CVE-2022-32205"
    }
   ],
   "SPDXID": "SPDXRef-Package-rpm-libcurl-710fb33829bc5106559bcd380cddb7d5"
  },
  {
   "name": "hunspell-en-US",
   "versionInfo": "0.20121024-6.amzn2.0.1",
   "downloadLocation": "NOASSERTION",
   "sourceInfo": "/var/lib/rpm/Packages",
   "filesAnalyzed": false,
   "externalRefs": [{
    "referenceCategory": "PACKAGE-MANAGER",
    "referenceType": "purl",
    "referenceLocator": "pkg:rpm/hunspell-en-US@0.20121024-6.amzn2.0.1?
arch=NOARCH&epoch=0&upstream=hunspell-en-US-0.20121024-6.amzn2.0.1.src.rpm"
   }],
   "SPDXID": "SPDXRef-Package-rpm-hunspell-en-US-de19ae0883973d6cea5e7e079d544fe5"
  },
  {
   "name": "grub2-tools-minimal",
   "versionInfo": "2.06-2.amzn2.0.6",
   "downloadLocation": "NOASSERTION",
   "sourceInfo": "/var/lib/rpm/Packages",
   "filesAnalyzed": false,
   "externalRefs": [{
     "referenceCategory": "PACKAGE-MANAGER",
     "referenceType": "purl",
     "referenceLocator": "pkg:rpm/grub2-tools-minimal@2.06-2.amzn2.0.6?
arch=X86_64&epoch=1&upstream=grub2-tools-minimal-2.06-2.amzn2.0.6.src.rpm"
    },
    {
     "referenceCategory": "SECURITY",
     "referenceType": "vulnerability",
     "referenceLocator": "CVE-2021-3981"
    }
   ],
   "SPDXID": "SPDXRef-Package-rpm-grub2-tools-minimal-c56b7ea76e5a28ab8f232ef6d7564636"
  },
  {
```

```
   "name": "unixODBC-devel",
   "versionInfo": "2.3.1-14.amzn2",
   "downloadLocation": "NOASSERTION",
   "sourceInfo": "/var/lib/rpm/Packages",
   "filesAnalyzed": false,
   "externalRefs": [{
    "referenceCategory": "PACKAGE-MANAGER",
    "referenceType": "purl",
    "referenceLocator": "pkg:rpm/unixODBC-devel@2.3.1-14.amzn2?
arch=X86_64&epoch=0&upstream=unixODBC-devel-2.3.1-14.amzn2.src.rpm"
   }],
   "SPDXID": "SPDXRef-Package-rpm-unixODBC-devel-1bb35add92978df021a13fc9f81237d2"
  }
 ],
 "relationships": [{
   "spdxElementId": "SPDXRef-DOCUMENT",
   "relatedSpdxElement": "SPDXRef-Package-rpm-elfutils-libelf-
ddf56a513c0e76ab2ae3246d9a91c463",
   "relationshipType": "DESCRIBES"
  },
  {
   "spdxElementId": "SPDXRef-DOCUMENT",
   "relatedSpdxElement": "SPDXRef-Package-rpm-yajl-8476ce2db98b28cfab2b4484f84f1903",
   "relationshipType": "DESCRIBES"
  },
  {
   "spdxElementId": "SPDXRef-DOCUMENT",
   "relatedSpdxElement": "SPDXRef-Package-rpm-unixODBC-
devel-1bb35add92978df021a13fc9f81237d2",
   "relationshipType": "DESCRIBES"
  }
 ],
 "SPDXID": "SPDXRef-DOCUMENT"
}
```


Filters for SBOMs

When you export SBOMs you can include filters to create reports for specific subsets of resources. If you don't supply a filter the SBOMs for all active, supported resources are exported. And if you are a delegated administrator this includes resources for all members too. The following filters are available:

- **AccountID** — This filter can be used to export SBOMs for any resources associated with specific Account ID.
- **EC2 instance tag** — This filter can be used to export SBOMs for EC2 instances with specific tags.
- **Function name** — This filter can be used to export SBOMs for specific Lambda functions.
- **Image tag** — This filter can be used to export SBOMs for container images with specific tags.
- **Lambda function tag** — This filter can be used to export SBOMs for Lambda functions with specific tags.
- **Resource type** — This filter can be used to filter resource type: EC2/ECR/Lambda.
- **Resource ID** — This filter can be used to export an SBOM for a specific resource.
- **Repository name** —This filter can be used to generate SBOMs for container images in specific repositories.

Configure and export SBOMs

To export SBOMs, you must first configure an Amazon S3 bucket and a AWS KMS key that Amazon Inspector is allowed to use. You can use filters to export SBOMs for specific subsets of your resources. To export SBOMs for multiple accounts in an AWS Organization, follow these steps while signed in as the Amazon Inspector delegated administrator.

Prerequisites

- Supported resources that are being actively monitored by Amazon Inspector.
- An Amazon S3 bucket configured with a policy that allows Amazon Inspector to add object to. For information on configuring the policy see Configure export permissions.
- An AWS KMS key configured with a policy that allows Amazon Inspector to use to encrypt your reports. For information on configuring the policy see Configure an AWS KMS key for export.

> **Note**
>
> If you have previously configured an Amazon S3 bucket and an AWS KMS key for findings export you can use the same bucket and key for SBOM export.

Choose your preferred access method to export an SBOM.

Console

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Using the AWS Region selector in the upper-right corner of the page, select the Region with the resources you want to export SBOM for.
3. In the navigation pane, choose **Export SBOMs**.
4. (Optional) In the **Export SBOMs** page, use the **Add filter** menu to select a subset of resources to create reports for. If no filter is provided Amazon Inspector will export reports for all active resources. If you are a delegated administrator this will include all active resources in your organization.
5. Under **Export setting** select the format you want for the SBOM.
6. Enter an **Amazon S3 URI** or choose **Browse Amazon S3** to select an Amazon S3 location to store the SBOM.
7. Enter a **AWS KMS key** configured for Amazon Inspector to use to encrypt your reports.

API

- To export SBOMs for your resources programmatically, use the CreateSbomExport operation of the Amazon Inspector API.

  In your request, use the `reportFormat` parameter to specify the SBOM output format, choose `CYCLONEDX_1_4` or `SPDX_2_3`. The `s3Destination` parameter is required and you must specify an S3 bucket configured with a policy that allows Amazon Inspector to write to it. Optionally use `resourceFilterCriteria` parameters to limit the scope of the report to specific resources.

AWS CLI

- To export SBOMs for your resources using the AWS Command Line Interface run the following command:

  ```
  aws inspector2 create-sbom-export --report-format
  FORMAT --s3-destination bucketName=amzn-s3-demo-
  bucket1,keyPrefix=PREFIX,kmsKeyArn=arn:aws:kms:Region:111122223333:key/123
  ```

In your request, replace *FORMAT* with the format of your choice, CYCLONEDX_1_4 or SPDX_2_3. Then replace the *user input placeholders* for the s3 destination with the name of the S3 bucket to export to, the prefix to use for the output in S3, and the ARN for the KMS key you are using to encrypt the reports.

Amazon EventBridge event schema for Amazon Inspector events

Amazon EventBridge delivers a stream of real-time data from applications and other AWS services to targets, such as AWS Lambda functions, Amazon Simple Notification Service topics, and data streams in Amazon Kinesis Data Streams. To support integration with other applications, services, and systems, Amazon Inspector automatically publishes findings to EventBridge as events. You can use Amazon Inspector to publish events for findings, coverage, and scans. This section provides example schemas for EventBridge events.

Topics

- Amazon EventBridge base schema for Amazon Inspector
- Amazon Inspector finding event schema example
- Amazon Inspector initial scan complete event schema example
- Amazon Inspector on-demand machine image scan event schema examples
- Amazon Inspector coverage event schema example
- Amazon Inspector auto enable schema example

Amazon EventBridge base schema for Amazon Inspector

The following is an example of the basic schema for an EventBridge event for Amazon Inspector. Event details differ based on the type of event.

```
{
    "version": "0",
    "id": "Event ID",
    "detail-type": "Inspector2 *event type*",
    "source": "aws.inspector2",
    "account": "AWS account ID (string)",
    "time": "event timestamp (string)",
    "region": "AWS Region (string)",
    "resources": [
        *IDs or ARNs of the resources involved in the event*
    ],
    "detail": {
        *Details of an Amazon Inspector event type*
```

```
    }
}
```

Amazon Inspector finding event schema example

The following includes examples of the schema for an EventBridge event for Amazon Inspector findings. Finding events are created when Amazon Inspector identifies a software vulnerability or network issue in one of your resources. For a guide to creating notifications in response to this type of event, see Creating custom responses to Amazon Inspector findings with Amazon EventBridge.

The following fields identify a finding event:

- `detail-type` is set to `Inspector2 Finding`.
- `detail` describes the finding.
- `detail.resources.tags` is where key-value data is stored.

Each object in `detail.resources` can also include the following fields. These fields are present for all resources, including AWS resources.

For AWS resources, these fields appear alongside the AWS-specific fields. You can rely on a single, provider-agnostic representation regardless of where the resource is hosted.

- `provider` is the cloud provider of the resource, for example `AWS` or `AZURE`.
- `providerAccountId` is the cloud provider account identifier for the resource, such as an Azure subscription ID. For AWS resources, this matches the `account` field.
- `providerOrgId` is the cloud provider organization or tenant identifier for the resource, such as an Azure tenant ID. Amazon Inspector populates this field for `AZURE` resources.
- `detail.resources.details` contains a provider-agnostic resource detail object that corresponds to the resource `type`: `vm` for a virtual machine, `image` for a container image, or `serverlessFunction` for a serverless function. These are the multi-cloud equivalents of the `awsEc2Instance`, `awsEcrContainerImage`, and `awsLambdaFunction` objects. For AWS resources, Amazon Inspector also includes the provider-agnostic object alongside the corresponding AWS-specific object. For example, an Amazon EC2 instance finding includes both `awsEc2Instance` and `vm`.

You can filter the tabs to see finding event schemas for different resources and finding types.

Amazon EC2 package vulnerability finding


```
{
    "version": "0",
    "id": "4d621919-f1f4-4201-a0e2-37e4e330ff51",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T17:00:36Z",
    "region": "eu-central-1",
    "resources": [
        "i-12345678901234567"
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "In snapd versions prior to 2.62, snapd failed to properly
 check the destination of symbolic links when extracting a snap. The snap format
 is a squashfs file-system image and so can contain symbolic links and other file
 types. Various file entries within the snap squashfs image (such as icons and
 desktop files etc) are directly read by snapd when it is extracted. An attacker who
 could convince a user to install a malicious snap which contained symbolic links
 at these paths could then cause snapd to write out the contents of the symbolic
 link destination into a world-readable directory. This in-turn could allow an
 unprivileged user to gain access to privileged information.",
        "epss": {
            "score": 0.00043
        },
        "exploitAvailable": "NO",
        "findingArn": "arn:aws:inspector2:eu-
central-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:59:44.356 UTC 2024",
        "fixAvailable": "YES",
        "inspectorScore": 4.8,
        "inspectorScoreDetails": {
            "adjustedCvss": {
                "adjustments": [],
                "cvssSource": "UBUNTU_CVE",
                "score": 4.8,
                "scoreSource": "UBUNTU_CVE",
                "scoringVector": "CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:L",
                "version": "3.1"
            }
        },
```

```
        "lastObservedAt": "Wed Sep 04 16:59:44.476 UTC 2024",
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 4.8,
                    "scoringVector": "CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:L",
                    "source": "UBUNTU_CVE",
                    "version": "3.1"
                },
                {
                    "baseScore": 7.3,
                    "scoringVector": "CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H",
                    "source": "NVD",
                    "version": "3.1"
                }
            ],
            "referenceUrls": [
                "https://www.cve.org/CVERecord?id=CVE-2024-29069",
                "https://ubuntu.com/security/notices/USN-6940-1"
            ],
            "relatedVulnerabilities": [
                "USN-6940-1"
            ],
            "source": "UBUNTU_CVE",
            "sourceUrl": "https://people.canonical.com/~ubuntu-security/cve/2024/
CVE-2024-29069.html",
            "vendorCreatedAt": "Thu Jul 25 20:15:00.000 UTC 2024",
            "vendorSeverity": "medium",
            "vulnerabilityId": "CVE-2024-29069",
            "vulnerablePackages": [
                {
                    "arch": "ALL",
                    "epoch": 0,
                    "fixedInVersion": "0:2.63+22.04ubuntu0.1",
                    "name": "snapd",
                    "packageManager": "OS",
                    "remediation": "apt-get update && apt-get upgrade",
                    "version": "2.63"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
```

```
            }
        },
        "resources": [
            {
                "details": {
                    "awsEc2Instance": {
                        "iamInstanceProfileArn":
 "arn:aws:iam::123456789012:instance-profile/AmazonSSMRoleForInstancesQuickSetup",
                        "imageId": "ami-02ff980600c693b38",
                        "ipV4Addresses": [
                            "192.0.2.10",
                            "203.0.113.20"
                        ],
                        "ipV6Addresses": [],
                        "launchedAt": "Wed Sep 04 16:57:40.000 UTC 2024",
                        "platform": "UBUNTU_22_04",
                        "subnetId": "subnet-12345678",
                        "type": "t2.small",
                        "vpcId": "vpc-12345678"
                    },
                    "vm": {
                        "type": "t2.small",
                        "vmImageReference": "ami-02ff980600c693b38",
                        "ipV4Addresses": [
                            "192.0.2.10",
                            "203.0.113.20"
                        ],
                        "ipV6Addresses": [],
                        "networkId": "vpc-12345678",
                        "subnetIds": [
                            "subnet-12345678"
                        ],
                        "launchedAt": "Wed Sep 04 16:57:40.000 UTC 2024",
                        "platform": "UBUNTU_22_04",
                        "executionRole": "arn:aws:iam::123456789012:instance-
profile/AmazonSSMRoleForInstancesQuickSetup"
                    }
                },
                "id": "i-12345678901234567",
                "partition": "aws",
                "provider": "AWS",
                "providerAccountId": "123456789012",
                "region": "eu-central-1",
                "type": "AWS_EC2_INSTANCE"
```

```
            }
        ],
        "severity": "MEDIUM",
        "status": "CLOSED",
        "title": "CVE-2024-29069 - snapd",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 17:00:36.951 UTC 2024"
    }
}
```

Amazon EC2 network reachability finding

```
{
    "version": "0",
    "id": "9eb1603b-4263-19ec-8be2-33184694cb92",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-05T13:06:56Z",
    "region": "eu-central-1",
    "resources": ["i-12345678901234567"],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "On the instance i-12345678901234567, the port range
 22-22 is reachable from the InternetGateway igw-261bab4d from an attached ENI
 eni-094ad651219472857.",
        "findingArn": "arn:aws:inspector2:eu-
central-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Thu Sep 05 13:06:56.334 UTC 2024",
        "lastObservedAt": "Thu Sep 05 13:06:56.334 UTC 2024",
        "networkReachabilityDetails": {
            "networkPath": {
                "steps": [{
                    "componentId": "igw-261bab4d",
                    "componentType": "AWS::EC2::InternetGateway"
                }, {
                    "componentId": "acl-171b527d",
                    "componentType": "AWS::EC2::NetworkAcl"
                }, {
                    "componentId": "sg-0d34debf87410f2d9",
                    "componentType": "AWS::EC2::SecurityGroup"
```

```
                }, {
                    "componentId": "eni-094ad651219472857",
                    "componentType": "AWS::EC2::NetworkInterface"
                }, {
                    "componentId": "i-12345678901234567",
                    "componentType": "AWS::EC2::Instance"
                }]
            },
            "openPortRange": {
                "begin": 22,
                "end": 22
            },
            "protocol": "TCP"
        },
        "remediation": {
            "recommendation": {
                "text": "You can restrict access to your instance by modifying the
 Security Groups or ACLs in the network path."
            }
        },
        "resources": [{
            "details": {
                "awsEc2Instance": {
                    "iamInstanceProfileArn": "arn:aws:iam::123456789012:instance-
profile/AmazonSSMRoleForInstancesQuickSetup",
                    "imageId": "ami-02ff980600c693b38",
                    "ipV4Addresses": ["192.0.2.10", "203.0.113.20"],
                    "ipV6Addresses": [],
                    "launchedAt": "Wed Sep 04 17:41:24.000 UTC 2024",
                    "platform": "UBUNTU_22_04",
                    "subnetId": "subnet-12345678",
                    "type": "t2.small",
                    "vpcId": "vpc-12345678"
                },
                "vm": {
                    "type": "t2.small",
                    "vmImageReference": "ami-02ff980600c693b38",
                    "ipV4Addresses": ["192.0.2.10", "203.0.113.20"],
                    "ipV6Addresses": [],
                    "networkId": "vpc-12345678",
                    "subnetIds": ["subnet-12345678"],
                    "launchedAt": "Wed Sep 04 17:41:24.000 UTC 2024",
                    "platform": "UBUNTU_22_04",
```

```
                    "executionRole": "arn:aws:iam::123456789012:instance-profile/
AmazonSSMRoleForInstancesQuickSetup"
                }
            },
            "id": "i-12345678901234567",
            "partition": "aws",
            "provider": "AWS",
            "providerAccountId": "123456789012",
            "region": "eu-central-1",
            "type": "AWS_EC2_INSTANCE"
        }],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "Port 22 is reachable from an Internet Gateway - TCP",
        "type": "NETWORK_REACHABILITY",
        "updatedAt": "Thu Sep 05 13:06:56.334 UTC 2024"
    }
}
```

Amazon ECR package vulnerability finding

```
{
    "version": "0",
    "id": "5325facf-a1aa-7d97-6bce-25fde6f6d2fc",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T16:55:38Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:ecr:eu-central-1:123456789012:repository/inspector2/
sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d"
    ],
    "detail.resources.tags.testkey": "allow",
    "detail": {
        "awsAccountId": "123456789012",
        "description": "Possible denial of service in X.509 name checks",
        "epss": {
            "score": 0.00045
        },
        "exploitAvailable": "NO",
```

```
        "findingArn": "arn:aws:inspector2:eu-
central-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:55:38.411 UTC 2024",
        "fixAvailable": "YES",
        "lastObservedAt": "Wed Sep 04 16:55:38.411 UTC 2024",
        "packageVulnerabilityDetails": {
            "cvss": [],
            "referenceUrls": [
                "https://www.cve.org/CVERecord?id=CVE-2024-6119",
                "https://ubuntu.com/security/notices/USN-6986-1"
            ],
            "relatedVulnerabilities": [
                "USN-6986-1"
            ],
            "source": "UBUNTU_CVE",
            "sourceUrl": "https://people.canonical.com/~ubuntu-security/cve/2024/
CVE-2024-6119.html",
            "vendorCreatedAt": "Tue Sep 03 00:00:00.000 UTC 2024",
            "vendorSeverity": "medium",
            "vulnerabilityId": "CVE-2024-6119",
            "vulnerablePackages": [
                {
                    "arch": "ARM64",
                    "epoch": 0,
                    "fixedInVersion": "0:3.0.13-0ubuntu3.4",
                    "name": "libssl3t64",
                    "packageManager": "OS",
                    "release": "0ubuntu3.2",
                    "remediation": "apt-get update && apt-get upgrade",
                    "sourceLayerHash":
 "sha256:1567e7ea90b67fc95ccdeeec39bdc3045098dee7e0c604975b957a9f8c0e9616",
                    "version": "3.0.13"
                },
                {
                    "arch": "ARM64",
                    "epoch": 0,
                    "fixedInVersion": "0:3.0.13-0ubuntu3.4",
                    "name": "openssl",
                    "packageManager": "OS",
                    "release": "0ubuntu3.2",
                    "remediation": "apt-get update && apt-get upgrade",
                    "sourceLayerHash":
 "sha256:1567e7ea90b67fc95ccdeeec39bdc3045098dee7e0c604975b957a9f8c0e9616",
                    "version": "3.0.13"
```

```
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
            }
        },
        "resources": [
            {
                "details": {
                    "awsEcrContainerImage": {
                        "architecture": "arm64",
                        "imageHash":
 "sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d",
                        "imageTags": [
                            "ubuntu_latest"
                        ],
                        "platform": "UBUNTU_24_04",
                        "pushedAt": "Wed Sep 04 16:55:28.000 UTC 2024",
                        "registry": "123456789012",
                        "repositoryName": "inspector2"
                    },
                    "image": {
                        "repositoryName": "inspector2",
                        "registry": "123456789012",
                        "imageTags": [
                            "ubuntu_latest"
                        ],
                        "imageDigest":
 "sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d",
                        "pushedAt": "Wed Sep 04 16:55:28.000 UTC 2024",
                        "architecture": "arm64",
                        "platform": "UBUNTU_24_04"
                    }
                },
                "id": "arn:aws:ecr:eu-central-1:123456789012:repository/inspector2/
sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d",
                "partition": "aws",
                "provider": "AWS",
                "providerAccountId": "123456789012",
                "region": "eu-central-1",
                "type": "AWS_ECR_CONTAINER_IMAGE"
            }
```

```
        ],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2024-6119 - libssl3t64, openssl",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 16:55:38.411 UTC 2024"
    }
}
```


Lambda package vulnerability finding


```
{
    "version": "0",
    "id": "9eadd71a-e49c-9864-6ba9-2a5d3f83c88f",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T16:50:37Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:lambda:eu-central-1:123456789012:function:VulnerableFunction:
$LATEST"
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "Flask is a lightweight WSGI web application framework. When
 all of the following conditions are met, a response containing data intended for
 one client may be cached and subsequently sent by the proxy to other clients. If
 the proxy also caches `Set-Cookie` headers, it may send one client's `session`
 cookie to other clients. The severity depends on the application's use of the
 session and the proxy's behavior regarding cookies. The risk depends on all these
 conditions being met.\n\n1. The application must be hosted behind a caching proxy
 that does not strip cookies or ignore responses with cookies. 2. The application
 sets `session.permanent = True` 3. The application does not access or modify the
 session at any point during a request. 4. `SESSION_REFRESH_EACH_REQUEST` enabled
 (the default). 5. The application does not set a `Cache-Control` header to indicate
 that a page is private or should not be cached.\n\nThis happens because vulnerable
 versions of Flask only set the `Vary: Cookie` header when the session is ac",
        "epss": {
            "score": 0.00208
        },
```

```
        "exploitAvailable": "YES",
        "exploitabilityDetails": {
            "lastKnownExploitAt": "Sat Aug 31 00:04:50.000 UTC 2024"
        },
        "findingArn": "arn:aws:inspector2:eu-
central-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:50:37.627 UTC 2024",
        "fixAvailable": "YES",
        "inspectorScore": 7.5,
        "inspectorScoreDetails": {
            "adjustedCvss": {
                "cvssSource": "NVD",
                "score": 7.5,
                "scoreSource": "NVD",
                "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N",
                "version": "3.1"
            }
        },
        "lastObservedAt": "Wed Sep 04 16:50:37.627 UTC 2024",
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 7.5,
                    "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N",
                    "source": "NVD",
                    "version": "3.1"
                }
            ],
            "referenceUrls": [
                "https://www.debian.org/security/2023/dsa-5442",
                "https://lists.debian.org/debian-lts-announce/2023/08/msg00024.html"
            ],
            "relatedVulnerabilities": [],
            "source": "NVD",
            "sourceUrl": "https://nvd.nist.gov/vuln/detail/CVE-2023-30861",
            "vendorCreatedAt": "Tue May 02 18:15:52.000 UTC 2023",
            "vendorSeverity": "HIGH",
            "vendorUpdatedAt": "Sun Aug 20 21:15:09.000 UTC 2023",
            "vulnerabilityId": "CVE-2023-30861",
            "vulnerablePackages": [
                {
                    "epoch": 0,
                    "filePath": "requirements.txt",
                    "fixedInVersion": "2.3.2",
```

```
                    "name": "flask",
                    "packageManager": "PIP",
                    "version": "2.0.0"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
            }
        },
        "resources": [
            {
                "details": {
                    "awsLambdaFunction": {
                        "architectures": [
                            "X86_64"
                        ],
                        "codeSha256": "O7jkFEmfPB+CK3Y6Pby5zW9gjG
+zusAaqRRMGS8B27c=",
                        "executionRoleArn": "arn:aws:iam::123456789012:role/service-
role/VulnerableFunction-role-f9vs5mq8",
                        "functionName": "VulnerableFunction",
                        "lastModifiedAt": "Wed Sep 04 16:50:20.000 UTC 2024",
                        "packageType": "ZIP",
                        "runtime": "PYTHON_3_11",
                        "version": "$LATEST"
                    },
                    "serverlessFunction": {
                        "serverlessFunctionName": "VulnerableFunction",
                        "runtime": "PYTHON_3_11",
                        "version": "$LATEST",
                        "codeDigest": "O7jkFEmfPB+CK3Y6Pby5zW9gjG
+zusAaqRRMGS8B27c=",
                        "lastModifiedAt": "Wed Sep 04 16:50:20.000 UTC 2024",
                        "executionRole": "arn:aws:iam::123456789012:role/service-
role/VulnerableFunction-role-f9vs5mq8",
                        "packageType": "ZIP",
                        "architectures": [
                            "X86_64"
                        ]
                    }
                },
```

```
                "id": "arn:aws:lambda:eu-
central-1:123456789012:function:VulnerableFunction:$LATEST",
                "partition": "aws",
                "provider": "AWS",
                "providerAccountId": "123456789012",
                "region": "eu-central-1",
                "type": "AWS_LAMBDA_FUNCTION"
            }
        ],
        "severity": "HIGH",
        "status": "ACTIVE",
        "title": "CVE-2023-30861 - flask",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 16:50:37.627 UTC 2024"
    }
}
```

Lambda code vulnerability finding

```
{
    "version": "0",
    "id": "e764f7be-f931-ff1b-204b-8cab2d91724b",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T16:51:01Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:lambda:eu-central-1:123456789012:function:VulnerableFunction:
$LATEST"
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "codeVulnerabilityDetails": {
            "cwes": [
                "CWE-798"
            ],
            "detectorId": "python/hardcoded-credentials@v1.0",
            "detectorName": "Hardcoded credentials",
            "detectorTags": [
                "secrets",
```

```
                "security",
                "owasp-top10",
                "top25-cwes",
                "cwe-798",
                "Python"
            ],
            "filePath": {
                "endLine": 6,
                "fileName": "lambda_function.py",
                "filePath": "lambda_function.py",
                "startLine": 6
            },
            "ruleId": "python-detect-hardcoded-aws-credentials"
        },
        "description": "Access credentials, such as passwords and access keys,
 should not be hardcoded in source code. Hardcoding credentials may cause leaks even
 after removing them. This is because version control systems might retain older
 versions of the code. Credentials should be stored securely and obtained from the
 runtime environment.",
        "findingArn": "arn:aws:inspector2:eu-
central-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:51:01.869 UTC 2024",
        "lastObservedAt": "Wed Sep 04 16:51:01.869 UTC 2024",
        "remediation": {
            "recommendation": {
                "text": "Your code uses hardcoded AWS credentials which might
 allow unauthorized users access to your AWS account. These attacks can occur
 a long time after the credentials are removed from the code. We recommend that
 you set AWS credentials with environment variables or an AWS profile instead.
 You should consider deleting the affected account or rotating the secret key
 and then monitoring Amazon CloudWatch for unexpected activity.\n[https://
boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html](https://
boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html)"
            }
        },
        "resources": [
            {
                "details": {
                    "awsLambdaFunction": {
                        "architectures": [
                            "X86_64"
                        ],
                        "codeSha256": "O7jkFEmfPB+CK3Y6Pby5zW9gjG
+zusAaqRRMGS8B27c=",
```

```
                        "executionRoleArn": "arn:aws:iam::123456789012:role/service-
role/VulnerableFunction-role-f9vs5mq8",
                        "functionName": "VulnerableFunction",
                        "lastModifiedAt": "Wed Sep 04 16:50:20.000 UTC 2024",
                        "packageType": "ZIP",
                        "runtime": "PYTHON_3_11",
                        "version": "$LATEST"
                    },
                    "serverlessFunction": {
                        "serverlessFunctionName": "VulnerableFunction",
                        "runtime": "PYTHON_3_11",
                        "version": "$LATEST",
                        "codeDigest": "O7jkFEmfPB+CK3Y6Pby5zW9gjG
+zusAaqRRMGS8B27c=",
                        "lastModifiedAt": "Wed Sep 04 16:50:20.000 UTC 2024",
                        "executionRole": "arn:aws:iam::123456789012:role/service-
role/VulnerableFunction-role-f9vs5mq8",
                        "packageType": "ZIP",
                        "architectures": [
                            "X86_64"
                        ]
                    }
                },
                "id": "arn:aws:lambda:eu-
central-1:123456789012:function:VulnerableFunction:$LATEST",
                "partition": "aws",
                "provider": "AWS",
                "providerAccountId": "123456789012",
                "region": "eu-central-1",
                "type": "AWS_LAMBDA_FUNCTION"
            }
        ],
        "severity": "CRITICAL",
        "status": "ACTIVE",
        "title": "CWE-798 - Hardcoded credentials",
        "type": "CODE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 16:51:01.869 UTC 2024"
    }
}
```

Azure virtual machine package vulnerability finding


```
{
    "version": "0",
    "id": "a1b2c3d4-1111-2222-3333-444455556666",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T17:00:36Z",
    "region": "us-east-1",
    "resources": [
        "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/resourcegroups/example-
rg/providers/microsoft.compute/virtualmachines/example-vm"
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "A package vulnerability was detected on an Azure virtual
 machine monitored by Amazon Inspector.",
        "exploitAvailable": "NO",
        "findingArn": "arn:aws:inspector2:us-
east-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:59:44.356 UTC 2024",
        "fixAvailable": "YES",
        "lastObservedAt": "Wed Sep 04 16:59:44.476 UTC 2024",
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 4.8,
                    "scoringVector": "CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:L",
                    "source": "UBUNTU_CVE",
                    "version": "3.1"
                }
            ],
            "referenceUrls": [
                "https://www.cve.org/CVERecord?id=CVE-2024-29069"
            ],
            "relatedVulnerabilities": [],
            "source": "UBUNTU_CVE",
            "sourceUrl": "https://people.canonical.com/~ubuntu-security/cve/2024/
CVE-2024-29069.html",
            "vendorSeverity": "medium",
            "vulnerabilityId": "CVE-2024-29069",
            "vulnerablePackages": [
```

```
                {
                    "arch": "ALL",
                    "epoch": 0,
                    "fixedInVersion": "0:2.63+22.04ubuntu0.1",
                    "name": "snapd",
                    "packageManager": "OS",
                    "remediation": "apt-get update && apt-get upgrade",
                    "version": "2.63"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
            }
        },
        "resources": [
            {
                "details": {
                    "vm": {
                        "type": "Standard_D2s_v3",
                        "vmName": "example-vm",
                        "vmImageReference": "Canonical:0001-com-ubuntu-server-
jammy:22_04-lts:latest",
                        "ipV4Addresses": [
                            "10.0.0.4"
                        ],
                        "ipV6Addresses": [],
                        "networkId": "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-
eeeeeeeeeeee/resourcegroups/example-rg/providers/microsoft.network/virtualnetworks/
example-vnet",
                        "subnetIds": [
                            "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/
resourcegroups/example-rg/providers/microsoft.network/virtualnetworks/example-vnet/
subnets/default"
                        ],
                        "securityGroupIds": [
                            "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/
resourcegroups/example-rg/providers/microsoft.network/networksecuritygroups/example-
nsg"
                        ],
                        "launchedAt": "Wed Sep 04 16:57:40.000 UTC 2024",
                        "platform": "UBUNTU_22_04",
```

```
                        "executionRole": "/subscriptions/aaaaaaaa-bbbb-cccc-
dddd-eeeeeeeeeeee/resourcegroups/example-rg/providers/microsoft.managedidentity/
userassignedidentities/example-identity",
                        "keyName": "example-key"
                    }
                },
                "id": "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/
resourcegroups/example-rg/providers/microsoft.compute/virtualmachines/example-vm",
                "provider": "AZURE",
                "providerAccountId": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
                "providerOrgId": "11111111-2222-3333-4444-555555555555",
                "region": "eastus",
                "type": "Microsoft.Compute/virtualMachines"
            }
        ],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2024-29069 - snapd",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 17:00:36.951 UTC 2024"
    }
}
```

Azure container image package vulnerability finding

```
{
    "version": "0",
    "id": "b2c3d4e5-2222-3333-4444-555566667777",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T16:55:38Z",
    "region": "us-east-1",
    "resources": [
        "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/
resourcegroups/example-rg/providers/microsoft.containerregistry/
registries/exampleregistry/repositories/inspector2/images/
sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d"
    ],
    "detail": {
        "awsAccountId": "123456789012",
```

```
        "description": "A package vulnerability was detected in an Azure Container
 Registry image monitored by Amazon Inspector.",
        "exploitAvailable": "NO",
        "findingArn": "arn:aws:inspector2:us-
east-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:55:38.411 UTC 2024",
        "fixAvailable": "YES",
        "lastObservedAt": "Wed Sep 04 16:55:38.411 UTC 2024",
        "packageVulnerabilityDetails": {
            "cvss": [],
            "referenceUrls": [
                "https://www.cve.org/CVERecord?id=CVE-2024-6119"
            ],
            "relatedVulnerabilities": [],
            "source": "UBUNTU_CVE",
            "sourceUrl": "https://people.canonical.com/~ubuntu-security/cve/2024/
CVE-2024-6119.html",
            "vendorSeverity": "medium",
            "vulnerabilityId": "CVE-2024-6119",
            "vulnerablePackages": [
                {
                    "arch": "ARM64",
                    "epoch": 0,
                    "fixedInVersion": "0:3.0.13-0ubuntu3.4",
                    "name": "openssl",
                    "packageManager": "OS",
                    "remediation": "apt-get update && apt-get upgrade",
                    "version": "3.0.13"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
            }
        },
        "resources": [
            {
                "details": {
                    "image": {
                        "repositoryName": "inspector2",
                        "registry": "exampleregistry.azurecr.io",
                        "imageTags": [
                            "ubuntu_latest"
```

```
                        ],
                        "imageDigest":
 "sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d",
                        "pushedAt": "Wed Sep 04 16:55:28.000 UTC 2024",
                        "architecture": "arm64",
                        "author": "example-author",
                        "platform": "UBUNTU_24_04"
                    }
                },
                "id": "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/
resourcegroups/example-rg/providers/microsoft.containerregistry/
registries/exampleregistry/repositories/inspector2/images/
sha256:84f507df33c6864d49c296fb734192696e4cb6f78166ac51ac8b9b118181085d",
                "provider": "AZURE",
                "providerAccountId": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
                "providerOrgId": "11111111-2222-3333-4444-555555555555",
                "region": "eastus",
                "type": "Microsoft.ContainerRegistry/registry/containerImage"
            }
        ],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2024-6119 - openssl",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 16:55:38.411 UTC 2024"
    }
}
```

Azure function app package vulnerability finding

```
{
    "version": "0",
    "id": "c3d4e5f6-3333-4444-5555-666677778888",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-09-04T16:50:37Z",
    "region": "us-east-1",
    "resources": [
        "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/resourcegroups/example-
rg/providers/microsoft.web/sites/example-functionapp"
```

```
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "A package vulnerability was detected in an Azure function
 app monitored by Amazon Inspector.",
        "exploitAvailable": "YES",
        "findingArn": "arn:aws:inspector2:us-
east-1:123456789012:finding/FINDING_ID",
        "firstObservedAt": "Wed Sep 04 16:50:37.627 UTC 2024",
        "fixAvailable": "YES",
        "lastObservedAt": "Wed Sep 04 16:50:37.627 UTC 2024",
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 7.5,
                    "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N",
                    "source": "NVD",
                    "version": "3.1"
                }
            ],
            "referenceUrls": [],
            "relatedVulnerabilities": [],
            "source": "NVD",
            "sourceUrl": "https://nvd.nist.gov/vuln/detail/CVE-2023-30861",
            "vendorSeverity": "HIGH",
            "vulnerabilityId": "CVE-2023-30861",
            "vulnerablePackages": [
                {
                    "epoch": 0,
                    "filePath": "requirements.txt",
                    "fixedInVersion": "2.3.2",
                    "name": "flask",
                    "packageManager": "PIP",
                    "version": "2.0.0"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "None Provided"
            }
        },
        "resources": [
            {
```

```
                "details": {
                    "serverlessFunction": {
                        "serverlessFunctionName": "example-functionapp",
                        "runtime": "PYTHON_3_11",
                        "version": "1",
                        "codeDigest": "O7jkFEmfPB+CK3Y6Pby5zW9gjG
+zusAaqRRMGS8B27c=",
                        "lastModifiedAt": "Wed Sep 04 16:50:20.000 UTC 2024",
                        "executionRole": "/subscriptions/aaaaaaaa-bbbb-cccc-
dddd-eeeeeeeeeeee/resourcegroups/example-rg/providers/microsoft.managedidentity/
userassignedidentities/example-identity",
                        "packageType": "ZIP",
                        "architectures": [
                            "X86_64"
                        ]
                    }
                },
                "id": "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/
resourcegroups/example-rg/providers/microsoft.web/sites/example-functionapp",
                "provider": "AZURE",
                "providerAccountId": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
                "providerOrgId": "11111111-2222-3333-4444-555555555555",
                "region": "eastus",
                "type": "Microsoft.Web/sites"
            }
        ],
        "severity": "HIGH",
        "status": "ACTIVE",
        "title": "CVE-2023-30861 - flask",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Wed Sep 04 16:50:37.627 UTC 2024"
    }
}
```


Note

The detail value returns the JSON details of a single finding as an object. It does not return the entire findings response syntax, which supports multiple findings within an array.

Amazon Inspector initial scan complete event schema example

The following is an example of the EventBridge event schema for an Amazon Inspector event for completing an initial scan. This event is created when Amazon Inspector completes an initial scan of one of your resources.

The following fields identify an initial scan complete event:

- The `detail-type` field is set to `Inspector2 Scan`.
- The `detail` object contains a `finding-severity-counts` object that details the number of findings in the applicable severity categories, such as `CRITICAL`, `HIGH`, and `MEDIUM`.

Select from the options to see different initial scan event schemas by resource type.

Amazon EC2 instance initial scan

```
{
    "version": "0",
    "id": "28a46762-6ac8-6cc4-4f55-bc9ab99af928",
    "detail-type": "Inspector2 Scan",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2023-01-20T22:52:35Z",
    "region": "us-east-1",
    "resources": [
        "i-087d63509b8c97098"
    ],
    "detail": {
        "scan-status": "INITIAL_SCAN_COMPLETE",
        "finding-severity-counts": {
            "CRITICAL": 0,
            "HIGH": 0,
            "MEDIUM": 0,
            "TOTAL": 0
        },
        "instance-id": "i-087d63509b8c97098",
        "version": "1.0"
    }
}
```

Amazon ECR image initial scan


```
{
    "version": "0",
    "id": "fdaa751a-984c-a709-44f9-9a9da9cd3606",
    "detail-type": "Inspector2 Scan",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2023-01-20T23:15:18Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecr:us-east-1:111122223333:repository/inspector2"
    ],
    "detail": {
        "scan-status": "INITIAL_SCAN_COMPLETE",
        "repository-name": "arn:aws:ecr:us-east-1:111122223333:repository/
inspector2",
        "finding-severity-counts": {
            "CRITICAL": 0,
            "HIGH": 0,
            "MEDIUM": 0,
            "TOTAL": 0
        },
        "image-digest":
 "sha256:965fbcae990b0467ed5657caceaec165018ef44a4d2d46c7cdea80a9dff0d1ea",
        "image-tags": [
            "ubuntu22"
        ],
        "version": "1.0"
    }
}
```


Lambda function initial scan


```
{
  "version": "0",
  "id": "4f290a7c-361b-c442-03c8-a629f6f20d6c",
  "detail-type": "Inspector2 Scan",
  "source": "aws.inspector2",
```

```
  "account": "111122223333",
  "time": "2023-02-23T18:06:03Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:lambda:us-west-2:111122223333:function:lambda-example:$LATEST"
  ],
  "detail": {
    "scan-status": "INITIAL_SCAN_COMPLETE",
    "finding-severity-counts": {
      "CRITICAL": 0,
      "HIGH": 0,
      "MEDIUM": 0,
      "TOTAL": 0
    },
    "version": "1.0"
  }
}
```

Machine image initial scan

```
{
  "version": "0",
  "id": "6b1f9d0e-2c47-4a52-9f10-7a1b8c3d5e2f",
  "detail-type": "Inspector2 Scan",
  "source": "aws.inspector2",
  "account": "111122223333",
  "time": "2026-08-13T13:42:52Z",
  "region": "us-east-1",
  "resources": [
    "ami-0abcdef1234567890"
  ],
  "detail": {
    "scan-status": "INITIAL_SCAN_COMPLETE",
    "finding-severity-counts": {
      "CRITICAL": 0,
      "HIGH": 2,
      "MEDIUM": 5,
      "TOTAL": 7
    },
    "machine-image-id": "ami-0abcdef1234567890",
```

```
    "version": "1.0"
  }
}
```

Amazon Inspector on-demand machine image scan event schema examples

The following are examples of the EventBridge event schemas for an on-demand machine image scan. Amazon Inspector publishes these events when you start a scan with the StartOnDemandScan API operation. EventBridge is the only delivery mechanism for on-demand scan findings. For more information, see Running on-demand machine image scans with Amazon Inspector.

Amazon Inspector publishes two kinds of events for an on-demand scan:

- A **scan complete event**, published once when the scan reaches a terminal state. The `detail-type` field is set to `Inspector2 OnDemand Scan Complete`. A successful event contains a `findingSeverityCounts` object. A failed event contains a `scanStatusReason` field instead.
- A **findings event**, published once for each vulnerability that Amazon Inspector detects. The `detail-type` field is set to `OnDemand AMI Scan Findings`. These events don't include machine image metadata beyond the machine image ID.

Every event contains the `scanId` of the scan, so you can correlate findings events with the scan complete event for the same scan.

Select from the options to see the different on-demand scan event schemas.

Scan complete – successful

```
{
    "version": "0",
    "id": "7bf7f4b1-9f6c-4d3e-8a2b-5c1d0e9f3a44",
    "detail-type": "Inspector2 OnDemand Scan Complete",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2026-06-12T01:30:00Z",
```

```
    "region": "us-east-1",
    "resources": [
        "ami-0abcdef1234567890"
    ],
    "detail": {
        "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d",
        "awsAccountId": "111122223333",
        "resourceId": "ami-0abcdef1234567890",
        "resourceType": "MACHINE_IMAGE",
        "scanStatus": "SUCCESSFUL",
        "startedAt": "2026-06-12T00:00:00.000Z",
        "completedAt": "2026-06-12T01:30:00.000Z",
        "findingSeverityCounts": {
            "CRITICAL": 0,
            "HIGH": 2,
            "MEDIUM": 5,
            "LOW": 1,
            "INFORMATIONAL": 0,
            "UNTRIAGED": 0,
            "TOTAL": 8
        },
        "version": "1.0"
    }
}
```

Scan complete – failed

```
{
    "version": "0",
    "id": "3c2e9a17-8b4d-4f61-9a0c-2d7e6f1b8c33",
    "detail-type": "Inspector2 OnDemand Scan Complete",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2026-06-12T01:30:00Z",
    "region": "us-east-1",
    "resources": [
        "ami-0abcdef1234567890"
    ],
    "detail": {
        "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d",
        "awsAccountId": "111122223333",
```

```
        "resourceId": "ami-0abcdef1234567890",
        "resourceType": "MACHINE_IMAGE",
        "scanStatus": "FAILED",
        "scanStatusReason": "UNSUPPORTED_OS",
        "startedAt": "2026-06-12T00:00:00.000Z",
        "completedAt": "2026-06-12T01:30:00.000Z",
        "version": "1.0"
    }
}
```

For the list of `scanStatusReason` values that Amazon Inspector can return for a failed on-demand scan, see Scan status values for on-demand machine image scans.

Findings

```
{
    "source": "inspector2.ami.ondemand",
    "detail-type": "OnDemand AMI Scan Findings",
    "detail": {
        "findingArn": "arn:aws:inspector2:us-east-1:111122223333:finding/
ondemand/9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d/cea7c73a3d57d9c142992e845644d317",
        "scanId": "9f1c4b8d2e6a4f0b7c3d5e1a8b2f6c4d",
        "awsAccountId": "111122223333",
        "type": "PACKAGE_VULNERABILITY",
        "title": "CVE-2022-3171 - com.google.protobuf:protobuf-java",
        "description": "A parsing issue with binary data in protobuf-java...",
        "severity": "HIGH",
        "status": "ACTIVE",
        "firstObservedAt": "2026-06-12T01:30:00.000Z",
        "lastObservedAt": "2026-06-12T01:30:00.000Z",
        "updatedAt": "2026-06-12T01:30:00.000Z",
        "resources": [
            {
                "type": "MACHINE_IMAGE",
                "id": "ami-0abcdef1234567890",
                "partition": "aws",
                "region": "us-east-1"
            }
        ],
        "inspectorScore": 7.5,
        "inspectorScoreDetails": {
            "adjustedCvss": {
```

```
                "scoreSource": "NVD",
                "cvssSource": "NVD",
                "version": "3.1",
                "score": 7.5,
                "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H",
                "adjustments": []
            }
        },
        "packageVulnerabilityDetails": {
            "vulnerabilityId": "CVE-2022-3171",
            "vulnerablePackages": [
                {
                    "name": "com.google.protobuf:protobuf-java",
                    "version": "2.5.0",
                    "epoch": 0,
                    "packageManager": "JAVA",
                    "filePath": "/usr/lib/jars/protobuf-java-2.5.0.jar",
                    "fixedInVersion": "3.16.3"
                }
            ],
            "source": "NVD",
            "sourceUrl": "https://nvd.nist.gov/vuln/detail/CVE-2022-3171",
            "vendorSeverity": "high"
        },
        "remediation": {
            "recommendation": {
                "text": "Update com.google.protobuf:protobuf-java to 3.16.3 or
 later"
            }
        },
        "fixAvailable": "YES",
        "exploitAvailable": "NO"
    }
}
```


Note

EventBridge delivers events at least once, so your rule can receive the same event more than once. Use the `findingArn` field to deduplicate findings events.

Amazon Inspector coverage event schema example

The following is an example of the EventBridge event schema for an Amazon Inspector event for coverage. This event is created when Amazon Inspector scan coverage for a resource is changed. The following fields identify a coverage event:

- The `detail-type` field is set to `Inspector2 Coverage`.
- The `detail` object contains a `scanStatus` object that indicates the new scanning status for the resource.

The `detail` object also includes the following fields. These fields are present for all resources, including AWS resources.

- `provider` is the cloud provider of the resource, for example `AWS` or `AZURE`.
- `providerAccountId` is the cloud provider account identifier for the resource, such as an Azure subscription ID. For AWS resources, this matches the `account` field.
- `providerRegion` is the cloud provider region of the resource. For AWS resources, this matches the `region` field.
- `providerOrgId` is the cloud provider organization or tenant identifier for the resource, such as an Azure tenant ID. Amazon Inspector populates this field for `AZURE` resources.

Select from the options to see different coverage event schemas by resource type.

Amazon EC2 instance coverage

```
{
    "version": "0",
    "id": "000adda5-0fbf-913e-bc0e-10f0376412aa",
    "detail-type": "Inspector2 Coverage",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2023-01-20T22:51:39Z",
    "region": "us-east-1",
    "resources": [
        "i-087d63509b8c97098"
    ],
    "detail": {
```

```
        "scanStatus": {
            "reason": "UNMANAGED_EC2_INSTANCE",
            "statusCodeValue": "INACTIVE"
        },
        "scanType": "PACKAGE",
        "provider": "AWS",
        "providerAccountId": "111122223333",
        "providerRegion": "us-east-1",
        "eventTimestamp": "2023-01-20T22:51:35.665501Z",
        "version": "1.0"
    }
}
```

Azure virtual machine coverage

```
{
    "version": "0",
    "id": "111b1dda-0fbf-913e-bc0e-10f0376412bb",
    "detail-type": "Inspector2 Coverage",
    "source": "aws.inspector2",
    "account": "111122223333",
    "time": "2023-01-20T22:51:39Z",
    "region": "us-east-1",
    "resources": [
        "/subscriptions/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee/resourcegroups/example-
rg/providers/microsoft.compute/virtualmachines/example-vm"
    ],
    "detail": {
        "scanStatus": {
            "reason": "SUCCESSFUL",
            "statusCodeValue": "ACTIVE"
        },
        "scanType": "PACKAGE",
        "provider": "AZURE",
        "providerAccountId": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
        "providerRegion": "eastus",
        "providerOrgId": "11111111-2222-3333-4444-555555555555",
        "eventTimestamp": "2023-01-20T22:51:35.665501Z",
        "version": "1.0"
    }
}
```

Amazon Inspector auto enable schema example

The auto-enable event is sent to the delegated admin when Amazon Inspector is unable to support the number of members in an organization. The following fields identify an auto-enable event:

- The `detail-type` field is set to `Inspector2 AutoEnable`.
- The `detail` object describes why the auto enable event failed.

```
{
    "version": "0",
    "id": "85fc3613-e913-7fc4-a80c-a3753e4aa9ae",
    "detail-type": "Inspector2 AutoEnable",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2024-08-21T02:36:48Z",
    "region": "us-east-1",
    "detail": {
        “version”: “1.0.0”,
        “AutoEnableStatus”: “Failed”,
        “Reason”: "The number of member accounts enabled with AWS Inspector has reached
 the maximum limit of 10,000"
        }
}
```

Amazon Inspector VM Scanner

Overview

Amazon Inspector uses the Amazon Inspector VM Scanner to perform vulnerability assessments on Amazon EC2 instances. Amazon Inspector VM Scanner leverages the inventory collection capabilities of Amazon Inspector SBOM Generator to generate a Software Bill of Materials (SBOM) and submits the SBOM for evaluation through the Inspector Telemetry channel. When you enable Enhanced EC2 Scanning on your account, Inspector VM Scanner replaces Inspector SSM Plugin.

> **Recommendation**
>
> We recommend the Amazon Inspector VM Scanner over the legacy Amazon Inspector SSM plugin. The VM Scanner uses the same scan mechanism across all supported operating systems and produces more consistent findings, and it uses fewer compute resources. On Windows in particular, it avoids the per-query timeouts that can cause the Amazon Inspector SSM plugin to report findings inconsistently.

Amazon Inspector provides two approaches for deploying the VM Scanner:

- **Automatic installation (recommended)** – When you enable Enhanced EC2 Scanning in the Amazon Inspector console, Amazon Inspector uses Amazon EC2 Systems Manager (SSM) to automatically install and manage the VM Scanner on your Amazon EC2 instances. This is the simplest approach and requires no manual intervention. For more information, see Enabling Inspector VM Scanner.
- **Manual installation** – You can manually install the VM Scanner using standard package managers (RPM, DEB, APK, MSI, PKG). This approach does not require SSM. For more information, see Manual installation and configuration.

Benefits

Inspector VM Scanner provides the following benefits over the previous Inspector SSM Plugin:

- **Reduced resource usage** – Requires less compute during operation

- **More granular package collection** – Provides detailed package-level inventory for high fidelity evaluation
- **Improved scanning mechanism** – Uses Inspector SBOM Generator for consistent behavior with other Inspector supported resources
- **Separation of duties** – Security teams can enable scanning at the account level, while instance administrators retain control over manual installation and configuration on individual instances

Deep inspection

Enhanced scanning support

Enhanced scanning allows deep inspection paths on Windows, Linux, and macOS.

With Enhanced EC2 Scanning, the Amazon Inspector VM Scanner extends Amazon EC2 scanning coverage to include deep inspection. With deep inspection, Amazon Inspector detects package vulnerabilities for application programming language packages in addition to operating system packages. Amazon Inspector scans default locations for programming language package libraries, and you can configure custom paths in addition to the locations that Amazon Inspector scans by default.

Important

Amazon Inspector scans a default set of locations that varies by operating system. Software in a non-standard location – such as a database or application server on a non-system drive like `D:\` – is scanned only if you add it as a custom path. For more information, see Custom paths for Amazon Inspector deep inspection.

For the programming languages that deep inspection supports, see Supported programming languages: Amazon EC2 deep inspection.

Enabling Inspector VM Scanner

When you enable Enhanced EC2 Scanning in the Amazon Inspector console, Amazon Inspector uses Amazon EC2 Systems Manager (SSM) to automatically install the VM Scanner on your managed

Amazon EC2 instances. Once installed, the scanner executes periodically (every 3 hours by default) and sends results to the Amazon Inspector Telemetry Service.

Requirements

To use the automatic installation method, your Amazon EC2 instances must meet the following requirements:

- The SSM Agent must be installed and running on the instance. For more information, see Working with SSM Agent in the *AWS Systems Manager User Guide*.
- The instance must have an IAM instance profile that allows SSM to manage the instance. For more information, see Configure instance permissions for Systems Manager in the *AWS Systems Manager User Guide*.
- The instance must have network connectivity to the SSM service endpoints.

> **Note**
>
> If your instances do not have SSM Agent installed or cannot meet these requirements, you can use the manual installation method instead. For more information, see Manual installation and configuration.

Enabling Enhanced EC2 Scanning

To enable Enhanced EC2 Scanning and automatically install the VM Scanner:

1. Open the Amazon Inspector console at Getting Started with the AWS Management Console.
2. In the navigation pane, choose **Account management**.
3. Under **EC2 scanning**, choose **Edit**.
4. Enable **Enhanced EC2 Scanning**.

After you enable Enhanced EC2 Scanning, Amazon Inspector creates an SSM association that installs the VM Scanner on all eligible instances in your account. The scanner begins executing vulnerability assessments automatically.

Note

Warning: Enabling Enhanced EC2 Scanning with the Inspector VM Scanner permanently disables the Inspector SSM plugin. This applies to both individual AWS accounts and organizations that the delegated administrator enables. After you enable this feature, you cannot revert to the Inspector SSM plugin.

Note

Before you enable Enhanced EC2 Scanning, consider testing in a non-production account first. If you do not test first, you cannot revert to the Inspector SSM plugin if issues arise. For more information, see Manual installation and configuration.

Amazon VPC endpoint requirements for Enhanced EC2 Scanning on private Amazon EC2 instances

You can run Enhanced EC2 Scanning on Amazon EC2 instances over an Amazon network. However, if you want to run Enhanced EC2 Scanning on private Amazon EC2 instances, you must create Amazon VPC endpoints. The following endpoints are required:

- `com.amazonaws.`*`region`*`.inspector2-telemetry`
- `com.amazonaws.`*`region`*`.s3`
- `com.amazonaws.`*`region`*`.ssm`
- `com.amazonaws.`*`region`*`.ssmmessages`

Where *`region`* is the Region code for the applicable AWS Region.

For more information, see Improve the security of Amazon EC2 instances by using Amazon VPC endpoints for Systems Manager in the *AWS Systems Manager User Guide*.

Note

Beginning with SSM Agent version 3.3.40.0, SSM Agent uses the `ssmmessages` endpoint instead of the `ec2messages` endpoint whenever it's available. Earlier versions of SSM Agent use the `ec2messages` endpoint. This endpoint is supported only in AWS Regions

launched before 2024. For more information, see Reference: ec2messages, ssmmessages, and other API operations in the *AWS Systems Manager User Guide*.

Frequently asked questions

Why should I migrate to Inspector VM Scanner? How is Inspector VM Scanner different from Inspector SSM Plugin?

Inspector VM Scanner uses Amazon Inspector SBOM Generator for system package scanning and deep inspection, providing a consistent scan mechanism regardless of how packages were installed. This scan mechanism is also more performant, with the biggest performance gain being seen on Windows. This mechanism is fully owned and maintained by Inspector, which allows Inspector to respond quickly to any issues that arise, as well as expand detections into new ecosystems.

How is Inspector VM Scanner different from Inspector SBOM Generator? Why not use SBOM Generator directly?

Inspector VM Scanner uses SBOM Generator under the hood for inventory collection, but it also contains extra features specifically designed for Inspector EC2 Scanning. Inspector VM Scanner can communicate through the Inspector Telemetry channel, whereas Inspector SBOM Generator cannot. Additionally, Inspector VM Scanner comes with orchestration to be invoked regularly for scheduled scanning.

How will Center for Internet Security (CIS) scans be managed? Will Inspector VM Scanner be used for CIS?

CIS scans will be kept in Inspector SSM Plugin. At this time, there is no intention to migrate CIS support to Inspector VM Scanner.

Does Inspector VM Scanner require SSM?

No. SSM is used for the automatic installation when you enable Enhanced EC2 Scanning in the console, but it is not a hard dependency. You can manually install the VM Scanner using standard package managers without SSM. For more information, see Manual installation and configuration.

Does Inspector VM Scanner scan the same locations as Inspector SSM Plugin?

Not necessarily. Inspector VM Scanner scans a default set of locations that varies by operating system and can differ from Inspector SSM Plugin. Software installed in a custom location is

scanned only if you add it as a custom path. With Enhanced EC2 Scanning, you can set custom paths for Linux, Windows, and macOS instances. If you are activating or re-activating Amazon Inspector, review your custom paths before you rely on the results. For more information, see Custom paths for Amazon Inspector deep inspection.

Third-party software attribution

Inspector VM Scanner includes third-party software components that are licensed under open source licenses. In accordance with the terms of those licenses, the following attribution document lists the third-party components, their licenses, and the applicable license text.

```
https://inspector-vm-scanner.s3.amazonaws.com/latest/THIRD-PARTY-LICENSES.txt
```

Manual installation and configuration

This section describes how to manually install and configure Inspector VM Scanner on your Amazon EC2 instances. Manual installation is considered *agent-based scanning* and does not require Amazon EC2 Systems Manager (SSM).

> **Note**
>
> If you enable Enhanced EC2 Scanning in the Amazon Inspector console, Amazon Inspector automatically installs and manages the VM Scanner using SSM. Manual installation is only required if you cannot use SSM or prefer to manage the scanner independently.

Manually installing and uninstalling Inspector VM Scanner

Inspector VM Scanner is available for standalone installation through the following links:

Inspector VM Scanner packages

Package Type	Architecture	URL
.apk	ARM	https://inspector-vm-scanner.s3.amazonaws.com/latest/APK-ARM/inspector-vm-scanner-latest-r0.apk

Package Type	Architecture	URL
	X86_64	https://inspector-vm-scanner.s3.amazonaws.com/latest/APK-X86_64/inspector-vm-scanner-latest-r0.apk
.deb	ARM	https://inspector-vm-scanner.s3.amazonaws.com/latest/DEB-ARM/inspector-vm-scanner_latest_arm64.deb
	X86_64	https://inspector-vm-scanner.s3.amazonaws.com/latest/DEB-X86_64/inspector-vm-scanner_latest_amd64.deb
.msi	X86_64	https://inspector-vm-scanner.s3.amazonaws.com/latest/MSI-X86_64/inspector-vm-scanner-x86_64-latest.msi
.pkg	ARM	https://inspector-vm-scanner.s3.amazonaws.com/latest/PKG-ARM/inspector-vm-scanner.latest.arm64.pkg
.rpm	ARM	https://inspector-vm-scanner.s3.amazonaws.com/latest/RPM-ARM/inspector-vm-scanner-latest.arm64.rpm
	X86_64	https://inspector-vm-scanner.s3.amazonaws.com/latest/RPM-X86_64/inspector-vm-scanner-latest.x86_64.rpm

To view the procedure for manually installing Inspector VM Scanner on the specified package manager, choose a link from the following list:

- APK
- DEB
- MSI
- PKG
- RPM

APK

Installation

ARM

```
curl --output-dir /etc/apk/keys -O https://inspector-vm-scanner.s3.amazonaws.com/
latest/APK-ARM/inspector-vm-scanner.pem.pub
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/APK-ARM/inspector-vm-
scanner-latest-r0.apk
apk add inspector-vm-scanner-latest-r0.apk
```

X86_64

```
curl --output-dir /etc/apk/keys -O https://inspector-vm-scanner.s3.amazonaws.com/
latest/APK-X86_64/inspector-vm-scanner.pem.pub
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/APK-X86_64/inspector-vm-
scanner-latest-r0.apk
apk add inspector-vm-scanner-latest-r0.apk
```

Uninstallation

```
apk del inspector-vm-scanner
```

DEB

Installation

ARM

```
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/DEB-ARM/inspector-vm-
scanner.gpg.pub
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/DEB-ARM/inspector-vm-
scanner_latest_arm64.deb
gpg --import inspector-vm-scanner.gpg.pub
gpg --verify inspector-vm-scanner_latest_arm64.deb
sudo dpkg -i inspector-vm-scanner_latest_arm64.deb
```

X86_64

```
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/DEB-X86_64/inspector-vm-
scanner.gpg.pub
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/DEB-X86_64/inspector-vm-
scanner_latest_amd64.deb
gpg --import inspector-vm-scanner.gpg.pub
gpg --verify inspector-vm-scanner_latest_amd64.deb
sudo dpkg -i inspector-vm-scanner_latest_amd64.deb
```

Uninstallation

```
sudo dpkg -r inspector-vm-scanner
```

MSI

Installation

X86_64

```
Invoke-WebRequest https://inspector-vm-scanner.s3.amazonaws.com/latest/MSI-X86_64/
inspector-vm-scanner-x86_64-latest.msi -OutFile inspector-vm-scanner-x86_64-latest.msi
msiexec /i inspector-vm-scanner-x86_64-latest.msi /qn
```

Uninstallation

To uninstall Inspector VM Scanner on Windows, use the Windows Programs and Features control panel or the following PowerShell command:

```
Get-WmiObject -Class Win32_Product | Where-Object {$_.Name -eq "Inspector VM Scanner"}
 | ForEach-Object {$_.Uninstall()}
```

PKG

Installation

ARM

```
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/PKG-ARM/inspector-vm-
scanner.latest.arm64.pkg
pkgutil --check-signature inspector-vm-scanner.latest.arm64.pkg
sudo installer -pkg inspector-vm-scanner.latest.arm64.pkg -target /
```

Uninstallation

```
sudo rm /opt/aws/inspector/bin/inspector-vm-scanner
sudo rm -rf /var/lib/amazon/inspector
```

RPM

Installation

ARM

```
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/RPM-ARM/inspector-vm-
scanner.gpg.pub
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/RPM-ARM/inspector-vm-
scanner-latest.arm64.rpm
rpm --import inspector-vm-scanner.gpg.pub
rpm --checksig inspector-vm-scanner-latest.arm64.rpm
sudo yum install inspector-vm-scanner-latest.arm64.rpm
```

X86_64

```
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/RPM-X86_64/inspector-vm-
scanner.gpg.pub
curl -O https://inspector-vm-scanner.s3.amazonaws.com/latest/RPM-X86_64/inspector-vm-
scanner-latest.x86_64.rpm
rpm --import inspector-vm-scanner.gpg.pub
rpm --checksig inspector-vm-scanner-latest.x86_64.rpm
sudo yum install inspector-vm-scanner-latest.x86_64.rpm
```

Uninstallation

```
sudo yum remove inspector-vm-scanner
```

Installation paths

On all Unix-based operating systems (including macOS), Inspector VM Scanner installs to `/opt/aws/inspector/bin/inspector-vm-scanner`. The exception is Alpine-based operating systems (including Chainguard), which use the alternate path `/usr/opt/aws/inspector/bin/inspector-vm-scanner`.

On Windows, Inspector VM Scanner installs to `C:\Program Files\Amazon\Inspector\inspector-vm-scanner.exe`.

These installation paths (aside from Alpine) are identical to Inspector SSM Plugin, which stores all Inspector binaries in a single location.

Uninstalling Inspector VM Scanner

If you disable Enhanced EC2 Scanning, Inspector automatically uninstalls Inspector VM Scanner on all Inspector-managed instances. To remove a manual installation, see the uninstallation instructions for your package manager in the preceding sections.

Running Inspector VM Scanner

Inspector VM Scanner expects a **Scan Type** to be passed as the first parameter. At this time, the only supported value is `sbom`.

Default usage command:

```
./inspector-vm-scanner sbom --send-results telemetry
```

Print options for SBOM scan:

```
./inspector-vm-scanner sbom --help
```

Viewing output

The default Inspector workflow does not save an SBOM locally. However, if any failures occur with sending the resource SBOM through telemetry, it will be written to the following locations:

- `/var/lib/amazon/inspector/state/sbom.json` on Unix
- `C:\ProgramData\Amazon\Inspector\State\sbom.json` on Windows

Users can override this path during VM Scanner invocation. See Advanced configuration for more details.

Viewing logs

Inspector VM Scanner logs always start with `inspector-vm-scanner` and end with `.log`.

On Unix systems, Inspector VM Scanner writes logs to `/var/log/amazon/inspector`. On Windows, Inspector VM Scanner writes logs to `C:\ProgramData\Amazon\Inspector\Logs`.

Both of these paths are identical to what Inspector SSM Plugin used. This means that you might see old Inspector SSM Plugin logs alongside Inspector VM Scanner logs.

Inspector VM Scanner can also output optional metric logs, which track CPU and memory usage throughout program execution. These logs are written to the log directories mentioned previously, under a subdirectory named `metrics` (`Metrics` on Windows). See Advanced configuration for more details.

Viewing logs with systemd

On Linux systems using systemd, you can use **journalctl** to view Inspector VM Scanner logs.

View real-time logs

```
journalctl -u inspector-vm-scanner -f
```

View recent logs

```
journalctl -u inspector-vm-scanner --since "1 hour ago"
```

View logs from today

```
journalctl -u inspector-vm-scanner --since today
```

View service status and recent runs

```
systemctl status inspector-vm-scanner
```

Advanced configuration

This section describes advanced configuration options for Inspector VM Scanner.

Configuring local outputs

Inspector VM Scanner provides the following options to configure how local outputs are written:

- `--send-results` must be set to `telemetry` or `disabled`. If you pass `disabled`, Inspector VM Scanner proceeds without sending the SBOM.

> **Tip**
>
> Use `--state-dir` with `--send-results disabled` to save the SBOM locally.

- `--log-dir` configures where logs are written. By default, logs are written to stdout.
- `--log-level` configures the granularity of logs. By default, this is INFO.
- `--log-retention` configures how many days to retain logs. If a log file older than `--log-retention` is found in `--log-dir`, it is deleted. By default, this is 7 days.
- `--debug` configures debug level logging and forces a dedicated log file for the current execution (rather than trying to maintain one log file for each day).
- `--state-dir` configures where SBOMs are written. By default, SBOMs are not saved.
- `--metric-dir` configures where metric logs are written. By default, metric logs are not saved.
- `--cpu-profile` enables the Go runtime CPU profiler and configures where the result is written.
- `--mem-profile` enables the Go runtime memory profiler and configures where the result is written.
- `--config-path` directs Inspector VM Scanner to derive arguments from a local configuration file. If the same argument is passed in both the CLI and configuration file, the CLI value is prioritized.
  - Inspector VM Scanner configuration files are specified in TOML, with all argument names identical to the CLI.
  - By default, Inspector VM Scanner looks for a configuration file at `/opt/aws/inspector/etc/config` on Linux and macOS, or `C:\\ProgramData\\Amazon\\Inspector\\Config\\config` on Windows. The file has no extension but uses TOML format.
  - When you specify `--config-path`, the specified path takes precedence over the default path.

The following example shows a configuration file:

```
# Configuration file for Inspector VM Scanner
log-level = "INFO"
send-results = "telemetry"
```

```
cpu-profile = "cpuprofile"
mem-profile = "memprofile"
log-dir = "log"
state-dir = "state"
debug = false
log-retention = 7
scan-timeout = 300

[sbom]
max-scan-depth = 5
target-directory = ["~"]
```

Configuring resource usage

Inspector VM Scanner provides the following options to configure resource usage:

- `--scan-timeout` forces the scanner to timeout after a specified number of seconds. By default, the scanner does not timeout.
- `--nice-priority` sets the `nice` priority for the process (available for Unix systems). By default, this is 3.
- `--cpu-limit` sets a hard cap on CPU usage (available for Linux systems using `cgroups`). By default, this is 65%.
- `--process-priority` configures priority for the process (available for Windows systems). By default, this is the `BELOW NORMAL` priority.

> **Note**
>
> The default values for `--cpu-limit` and `--process-priority` are identical to Inspector SSM Plugin.

Configuring scan targets

Inspector VM Scanner leverages Inspector SBOM Generator for inventory collection. As a result, many of Inspector VM Scanner's scan coverage options are taken directly from SBOM Generator.

By default, Inspector VM Scanner does not scan your entire filesystem. It collects operating system packages, and on Windows it uses Windows Update (Knowledge Base) data to identify

system software that needs updating. It also inspects common installation locations for popular programming language packages and their dependencies.

Software installed in a custom or non-standard location – such as a database or application server on a non-system drive – is scanned only if you add it. To cover such software, add it as a custom path in your Amazon EC2 scan settings. Under Enhanced EC2 Scanning, custom paths apply to Linux, Windows, and macOS instances, and are scanned in addition to the default locations. For more information, see Custom paths for Amazon Inspector deep inspection.

You can also add directories per instance during VM Scanner invocation with the advanced options in this section. Directories that you add with `--target-directory` are scanned in addition to the defaults.

Inspector VM Scanner provides the following options to configure scan targets:

- `--max-scan-depth` configures the maximum number of directory levels that a scan traverses from each scan location.
- `--target-directory` adds one or more directories to scan in addition to the defaults. In a configuration file, specify it under `[sbom]` as an array, for example `target-directory = ["D:\\oracle"]`.
- `--override-scanners` configures exact filescanners, overriding Inspector VM Scanner defaults.
- `--additional-scanners` configures filescanners to use in addition to Inspector VM Scanner defaults.

You can use the following command to list all available scanners:

```
./inspector-vm-scanner sbom --list-scanners
```

Managing periodic execution

When you install Inspector VM Scanner through a package manager, the installation creates a scheduled task that executes scans automatically. You can view, modify, or disable this schedule.

Linux (systemd)

View service status and recent runs

```
systemctl status inspector-vm-scanner
```

View real-time logs

```
journalctl -u inspector-vm-scanner -f
```

View recent logs

```
journalctl -u inspector-vm-scanner --since "1 hour ago"
```

Check current timer interval

```
systemctl cat inspector-vm-scanner.timer
```

Change the scan interval

> **Important**
>
> Do not edit the unit files in `/usr/lib/systemd/system/` directly. Package updates overwrite those files. Use `systemctl edit` instead, which creates a persistent override that survives updates.

The default interval is every 3 hours. To change it, run:

```
sudo systemctl edit inspector-vm-scanner.timer
```

In the editor that opens, add the following (this example sets the interval to 6 hours):

```
[Timer]
OnBootSec=
OnUnitActiveSec=
OnBootSec=0
OnUnitActiveSec=21600s
```

The empty assignments clear the defaults before your values take effect. After saving, restart the timer:

```
sudo systemctl daemon-reload
sudo systemctl restart inspector-vm-scanner.timer
```

To verify that the override took effect, run `systemctl cat inspector-vm-scanner.timer`. To revert to defaults, run `sudo systemctl revert inspector-vm-scanner.timer`.

Enable or disable automatic execution

```
sudo systemctl enable inspector-vm-scanner.timer   # Enable automatic runs.
sudo systemctl disable inspector-vm-scanner.timer  # Disable automatic runs.
```

Windows (Task Scheduler)

View task status and last run

```
Get-ScheduledTask -TaskName "Inspector VM Scanner" | Get-ScheduledTaskInfo
```

View recent task logs

```
Get-ScheduledTaskInfo -TaskName "Inspector VM Scanner"
```

View detailed task history

```
schtasks /query /tn "Inspector VM Scanner" /v /fo list
```

View current task schedule

```
Get-ScheduledTask -TaskName "Inspector VM Scanner" | Select-Object -ExpandProperty
 Triggers
```

Update task schedule

To change the scan frequency:

```
# Modify trigger to run daily at 2 AM
$trigger = New-ScheduledTaskTrigger -Daily -At 2:00AM
Set-ScheduledTask -TaskName "Inspector VM Scanner" -Trigger $trigger
```

Enable or disable task

```
Enable-ScheduledTask -TaskName "Inspector VM Scanner"   # Enable automatic runs
Disable-ScheduledTask -TaskName "Inspector VM Scanner"  # Disable automatic runs
```

macOS (launchd)

View launchd task

```
sudo launchctl print system/com.amazon.inspector.vm-scanner
```

Execute single task

```
sudo launchctl start com.amazon.inspector.vm-scanner
```

Amazon Inspector SSM plugin for Linux and Windows

This topic describes the Amazon Inspector SSM plugin for Linux and Windows instances.

> ⚠ **We recommend Enhanced EC2 Scanning**
>
> The Amazon Inspector SSM plugin is the legacy scanner. We recommend that you upgrade to Enhanced EC2 Scanning with the Amazon Inspector VM Scanner. The VM Scanner uses the same scan mechanism across all supported operating systems and produces more consistent findings. On Windows in particular, it avoids the per-query timeouts that can cause the Amazon Inspector SSM plugin to report findings inconsistently.

The Amazon Inspector SSM plugin for Linux

If you have not enabled Enhanced EC2 Scanning with the Amazon Inspector VM Scanner, Amazon Inspector uses the Amazon Inspector SSM plugin to perform deep inspection scans on Linux instances. The Amazon Inspector SSM plugin is automatically installed on Linux instances in the `/opt/aws/inspector/bin` directory. The name of the executable is `inspectorssmplugin`.

Amazon Inspector uses Systems Manager Distributor to deploy the plugin on your instance. To perform deep inspection scans, Systems Manager Distributor and Amazon Inspector must support your Amazon EC2 instance operating system. For information about operating systems that

Systems Manager Distributor supports, see Supported package platforms and architectures in the *AWS Systems Manager User Guide*.

Amazon Inspector creates file directories to manage data collected for deep inspection by the Amazon Inspector SSM plugin. These file directories include `/opt/aws/inspector/var/input` and `/opt/aws/inspector/var/output`.

The `packages.txt` file in `/opt/aws/inspector/var/output` stores the full paths to packages that deep inspection discovers. If Amazon Inspector detects the same package multiple times on your instance, the `packages.txt` file lists each location where the package was found.

Amazon Inspector stores logs for the plugin in the `/var/log/amazon/inspector` directory.

Uninstalling the Amazon Inspector SSM plugin

If the `inspectorssmplugin` file is inadvertently deleted, the SSM association `InspectorLinuxDistributor-do-not-delete` will try to reinstall the `inspectorssmplugin` file at the next scan interval.

If you deactivate Amazon EC2 scanning, the plugin will be automatically uninstalled from all Linux hosts.

The Amazon Inspector SSM plugin for Windows

If you have not enabled Enhanced EC2 Scanning with the Amazon Inspector VM Scanner, the Amazon Inspector SSM plugin is required for Amazon Inspector to scan your Windows instances. The Amazon Inspector SSM plugin is automatically installed on your Windows instances in `C:\Program Files\Amazon\Inspector`, and the executable binary file is named `InspectorSsmPlugin.exe`.

The following file locations are created to store data the Amazon Inspector SSM plugin collects:

- `C:\ProgramData\Amazon\Inspector\Input`
- `C:\ProgramData\Amazon\Inspector\Output`
- `C:\ProgramData\Amazon\Inspector\Logs`

> **Note**
>
> By default, the Amazon Inspector SSM plugin runs at below normal priority.

> **Note**
>
> Scanning Windows instances requires `ssm:PutInventory` and `ssm:GetParameter` permissions. If an IAM instance profile is configured on the instance, Amazon Inspector uses that profile and ignores the Default Host Management Configuration (DHMC) role. The instance profile must include these permissions. If no instance profile is set, Amazon Inspector uses the configured Default Host Management Configuration role, which must include these permissions.

Uninstalling the Amazon Inspector SSM plugin

If the `InspectorSsmPlugin.exe` file is inadvertently deleted, the `InspectorDistributor-do-not-delete` association will reinstall the `InspectorSsmPlugin.exe` file at the next Windows scan interval. If you want to uninstall the Amazon Inspector SSM plugin, you can use the **Uninstall** action in the `AmazonInspector2-ConfigureInspectorSsmPlugin` document. However, the Amazon Inspector SSM plugin will be automatically uninstalled from all Windows hosts if you deactivate Amazon EC2 scanning.

> **Note**
>
> If you uninstall the SSM Agent before deactivating Amazon Inspector, the Amazon Inspector SSM plugin will remain on the Windows host, but will not send data to the Amazon Inspector SSM plugin. For more information, see Deactivating Amazon Inspector.

Integrating Amazon Inspector with Microsoft Azure

You can create an integration to Microsoft Azure in Amazon Inspector to detect software vulnerabilities in your Azure resources. After you create the connector, Amazon Inspector continuously scans your Azure Virtual Machines (VMs), Azure Function Apps, and Azure Container Registry (ACR) images for known vulnerabilities and generates findings with remediation guidance.

How it works

An integration to Microsoft Azure enables Amazon Inspector to discover and scan your Azure resources for software vulnerabilities. The connector uses IAM Outbound Identity Federation to securely authenticate to your Azure environment without storing long-lived credentials.

1. You create a connector in the Amazon Inspector console or using the API. You specify your Azure tenant ID, Azure Subscriptions, and Azure Regions, and select which capabilities to enable (VM scanning, Function App scanning, ACR scanning).
2. You run the generated Azure setup script to configure the Azure app registration, Event Hub infrastructure, and permissions.
3. Amazon Inspector discovers your Azure resources and begins scanning for vulnerabilities.

How AWS Config is used

AWS Config is used internally to collect resource configuration data and detect changes in your Azure environment (such as new VMs being launched or new images pushed to ACR). The connector handles this automatically. You do not need to enable AWS Config separately, configure a recorder, or pay for AWS Config usage. These costs are included in your Amazon Inspector pricing for Azure resources.

Connector types

Amazon Inspector supports two connector types for integrations to Microsoft Azure:

- **Service-linked connector (SLC)** – Created automatically when you set up an integration to Microsoft Azure through AWS Security Hub CSPM. You cannot edit an SLC directly in Amazon Inspector. Manage it through AWS Security Hub CSPM instead. The AWS Security Hub CSPM

scope (the monitored Azure Subscriptions and monitored Azure Regions) must always be greater than or equal to any existing customer-managed connector scope.

- **Customer-managed connector (CMC)** – Created directly in the Amazon Inspector console or via API. You have full control over scope and capabilities, including the ability to create, modify, and delete the connector.

For standalone Amazon Inspector usage, create a customer-managed connector. If you already have an AWS Security Hub CSPM integration to Microsoft Azure, Amazon Inspector automatically receives a service-linked connector and you do not need to create a separate one.

Scanning methods

Amazon Inspector uses the following methods to scan Azure resources:

- **Virtual Machines** – Amazon Inspector uses the Amazon Inspector VM Scanner agent to scan VMs for operating system and application vulnerabilities. You can choose between managed deployment (recommended) or manual installation. For more information about the agent, see *Amazon Inspector VM Scanner*.
- **Function Apps** – Amazon Inspector automatically extracts and analyzes function code dependencies for known vulnerabilities.
- **Container Registry images** – Amazon Inspector automatically scans container images stored in Azure Container Registry (ACR) for vulnerabilities in operating system packages and application dependencies.

Prerequisites for integrating Amazon Inspector with Microsoft Azure

Before you create a connector, verify that your AWS account and Azure environment meet the following requirements. The Azure setup script handles most Azure configuration automatically (app registration, Event Hub, diagnostic settings, Azure role assignments, and Microsoft Graph API permissions).

AWS account setup – all capabilities

IAM Outbound Identity Federation and OIDC provider

Your AWS account must have an IAM Token Issuer URL configured for outbound identity federation. This enables Amazon Inspector to securely authenticate to your Azure environment using short-lived tokens.

For VM scanning, you must also have an IAM OIDC identity provider configured with the issuer URL `sts.windows.net/tenant-id/`. This provider enables the Amazon Inspector VM Scanner agent on your Azure VMs to authenticate back to AWS using `sts:AssumeRoleWithWebIdentity`. Without this provider, the VM scanner cannot send telemetry to Amazon Inspector.

If you have already configured outbound identity federation for another AWS security service (such as AWS Security Hub CSPM or AWS Security Hub CSPM CSPM), you do not need to configure it again. For more information, see Identity providers and federation in the *IAM User Guide*.

On the Azure side, the setup script creates the federated identity credential that trusts the AWS STS OIDC issuer. You do not need to configure this manually.

Service-linked role

The API creates the Amazon Inspector service-linked role on behalf of the customer during connector creation. No manual SLR provisioning is required.

Config connector

A connector ARN is required when creating a connector via API. The Config connector captures Azure resource configuration changes. In the console, you can either select an existing connector or create a new one during connector creation.

Console IAM permissions

The IAM principal (user or role) that creates the connector requires the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "config:GetConnector",
```

```
        "config:ListConnectors",
        "config:PutConnector",
        "inspector2:CreateConnector",
        "inspector2:DeleteConnector",
        "inspector2:ListConnectors",
        "inspector2:ListTagsForResource",
        "inspector2:TagResource",
        "inspector2:UntagResource",
        "inspector2:UpdateConnector"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS account setup – VM scanning only

In addition to the requirements above, VM scanning requires:

- Amazon EC2 Systems Manager configured in the same Region as the connector
- Amazon EC2 Systems Manager Automation permissions for managed deployment of the Amazon Inspector VM Scanner agent

Azure account setup

Before you run the Azure setup script, ensure that you have the following:

Azure tenant

A Microsoft Azure commercial tenant. Azure Government and Azure China are not supported.

Microsoft Entra ID log availability

Resource recording requires Microsoft Graph audit logs and sign-in logs from your Microsoft Entra ID tenant. Some Microsoft Entra ID logs require Microsoft Entra ID P1 or P2 licensing.

If the tenant lacks required logs, resource recording might report partial failures. The app registration and Azure role assignments do not supply missing log data.

Azure role

The person configuring Azure must have the Global Administrator role in the Azure tenant.

Azure CLI

The Azure CLI installed and authenticated (`az login`).

> **Note**
>
> The person who sets up the Azure environment does not need to be the same person who creates the connector in the Amazon Inspector console.

Configuring Microsoft Azure to integrate with Amazon Inspector

The Azure setup process configures your Azure environment to allow Amazon Inspector to authenticate and receive resource change notifications. Amazon Inspector generates a customized setup script during connector creation. The script performs the following steps.

> **Note**
>
> If you have already completed Azure environment setup for AWS Security Hub CSPM or AWS Security Hub CSPM CSPM, the same app registration is shared. The script detects the existing registration and adds the required federated identity credential for Amazon Inspector without modifying existing configuration.

What the setup script does

The setup script performs a series of tasks. The following sections describe each task that the script performs. If your organization restricts script execution, you can perform these tasks manually, using this topic as a guide.

Step 1: Register an Azure application

Register a new application in Microsoft Entra ID. This application serves as the identity that AWS uses to authenticate to your Azure environment using federated credentials. No client secrets are required.

```
$ az ad app create --display-name "AWSSecurityHubIntegration"
```

Note the **Application (client) ID** from the output (the `appId` attribute). You will provide this value when creating the connector in the Amazon Inspector console.

Step 2: Create a service principal

Create a service principal for the registered application. You use Azure role assignments to grant permissions to the service principal. The service principal performs actions in your Azure environment.

```
$ az ad sp create --id application-client-id
```

Where *`application-client-id`* is the Application (client) ID from the previous step.

Step 3: Configure federated identity credentials

Configure federated identity credentials on the application to enable AWS to authenticate without client secrets. This establishes an OIDC trust between AWS and Azure. No long-lived secrets or credentials are stored.

```
$ az ad app federated-credential create \
  --id application-client-id \
  --parameters '{
    "name": "AWSConfigFederation",
    "issuer": "token-issuer-url",
    "subject": "arn:aws:iam::account-id:role/aws-service-role/
thirdparty.config.amazonaws.com/AWSServiceRoleForConfigThirdParty",
    "audiences": ["api://AzureADTokenExchange"],
    "description": "Federation for AWS Config third-party cloud resource discovery"
  }'
```

Key values:

- **Issuer** – Your Token Issuer URL from IAM Outbound Identity Federation. This follows the format `https://`*`uuid`*`.tokens.sts.global.api.aws`.
- **Subject** – The ARN of the `AWSServiceRoleForConfigThirdParty` service-linked role in your AWS account. AWS creates this service-linked role automatically when you create the connector.
- **Audience** – `api://AzureADTokenExchange` (standard value for workload identity federation).

Step 4: Assign the Reader role

Assign the **Reader** role to the service principal at the tenant root management group scope. This grants read-only access to all Azure resources across all subscriptions in the tenant.

```
$ az role assignment create \
  --assignee application-client-id \
  --role "Reader" \
  --scope "/providers/Microsoft.Management/managementGroups/tenant-id"
```

> **Note**
>
> If this step fails, verify that *Access management for Azure resources* is set to **Yes** in Microsoft Entra ID > Properties. Then sign out and sign back in to refresh your token.

Step 5: Configure Microsoft Graph API permissions

Grant the required Microsoft Graph API permissions (Application type) to the application: `Directory.Read.All`, `AuditLog.Read.All`, and `Policy.Read.All`. Then grant admin consent.

Step 6: Set up Azure Event Hub

AWS Config uses an Azure Event Hub to receive near real-time notifications of resource configuration changes in your Azure environment. The setup script creates the Event Hub infrastructure and tags it for discovery.

```
$ az group create \
  --name "aws-securityhub-integration" \
  --location event-hub-region

$ az eventhubs namespace create \
  --resource-group "aws-securityhub-integration" \
  --name "aws-securityhub-account-id" \
  --location event-hub-region \
  --sku Standard

$ az eventhubs eventhub create \
  --resource-group "aws-securityhub-integration" \
```

```
  --namespace-name "aws-securityhub-account-id" \
  --name "activitylog" \
  --message-retention 1 \
  --partition-count 4

$ az eventhubs eventhub consumer-group create \
  --resource-group "aws-securityhub-integration" \
  --namespace-name "aws-securityhub-account-id" \
  --eventhub-name "activitylog" \
  --name "AWSConfig"
```

After creating the Event Hub, tag the namespace for discovery and assign the Data Receiver role:

```
$ az tag create \
  --resource-id namespace-resource-id \
  --tags "AWSConfig-account-id-region=activitylog"

$ az role assignment create \
  --assignee application-client-id \
  --role "Azure Event Hubs Data Receiver" \
  --scope namespace-resource-id
```

Key configuration:

- **Namespace SKU** – Standard (required for consumer groups).
- **Event Hub name** – `activitylog`.
- **Consumer group** – `AWSConfig`.
- **Tag** – `AWSConfig-`*`account-id`*`-`*`region`* with value `activitylog`. This tag enables AWS Config to discover the Event Hub automatically.
- **Role assignment** – Azure Event Hubs Data Receiver on the namespace.

> **Note**
>
> The tag format is `AWSConfig-`*`account-id`*`-`*`region`* where *`account-id`* is your 12-digit AWS account identifier and *`region`* is the AWS Region where you created the connector (for example, `us-east-1`).

Step 7: Configure Activity Log and Entra ID Audit Log export

Configure Azure to export Activity Logs and Entra ID Audit Logs to the Event Hub.

Activity Log export:

Configure diagnostic settings on each subscription to export Activity Logs to the Event Hub. The setup script configures this automatically for all monitored subscriptions. Activity Logs capture subscription-level activity including role assignments and resource changes.

Entra ID Audit Log export:

Configure diagnostic settings on Microsoft Entra ID to export audit logs to the same Event Hub. Navigate to *Microsoft Entra ID > Monitoring > Diagnostic settings > Add diagnostic setting*. Enable **AuditLogs** and route them to your Event Hub.

Running the setup script

To run the Azure setup script:

1. Open a terminal with the Azure CLI authenticated to your Azure tenant (`az login`). You must have Contributor and Owner permissions on the target subscriptions.
2. Copy the setup script from the Amazon Inspector console (generated during connector creation).
3. Run the setup script. The script completes the Azure environment setup described above.
4. After the script completes successfully, return to the Amazon Inspector console to finish creating the connector.

> **Important**
>
> If you later change the connector scope (for example, add a new Azure region), the console regenerates the setup script to provision Event Hub infrastructure in the new region. You must re-run the updated script.

Configuring Amazon Inspector to integrate with Microsoft Azure

After you configure your Azure environment, create a connector in Amazon Inspector to begin scanning your Azure resources for vulnerabilities.

To create an Azure connector

1. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the navigation pane, choose **Integrations**.
3. Choose **Create connector**.
4. Provide the following information:

   - **Name** – A unique name for this connector. For name and description length constraints, see the CreateConnector operation in the Amazon Inspector API Reference.
   - **Description** (optional) – A description of this connector.

   > **Important**
   >
   > Names and descriptions are used to identify your content, and we recommend you do not include sensitive, confidential, or personally identifiable information (PII) in them.

   - **Azure Tenant ID** – Your Microsoft Entra ID tenant identifier. Found in the Azure portal under Microsoft Entra ID, Properties.
5. Select **capabilities** to enable:

   - VM scanning (Amazon Inspector VM Scanner agent deployed to Azure VMs)
   - Function App scanning (code dependency analysis)
   - ACR image scanning (container image vulnerability detection)

   You can enable any combination. Capabilities can be changed later via `UpdateConnector`.
6. Configure your **monitoring targets**:

- **Azure subscriptions** – Choose whether to monitor all subscriptions in your tenant or specify individual subscription IDs.
- **Azure regions** – Choose whether to enable all Azure regions or select specific regions.

7. Configure the **recording source**:

   Select an existing recording source, or create a new one. The recording source ARN is a required field in the API (`inspector2:CreateConnector`).

8. Copy the generated Azure setup script, run it in your Azure environment (see the section called "Configuring Azure"), and then choose **Create connector**.

Managed Amazon Inspector VM Scanner deployment

If you enabled VM scanning, Amazon Inspector uses Amazon EC2 Systems Manager Automation to deploy and manage the Amazon Inspector VM Scanner agent on your Azure VMs. With managed deployment:

- No manual installation is required.
- The agent stays up to date automatically.
- Amazon Inspector manages the full lifecycle (install, update, health monitoring).

Amazon Inspector uses Amazon EC2 Systems Manager Automation to deploy and manage the agent.

To enable VM scanning, you must create the following IAM resources in your account.

IAM OIDC identity provider

Create an IAM OIDC identity provider for Microsoft Entra ID with the following configuration:

- **Issuer URL:** `https://sts.windows.net/`*`tenant-id`*`/`
- **Client ID list:** `api://`*`app-id`*, `api://AzureADTokenExchange`
- **Thumbprint:** A 40-character string of all zeros (`0` repeated 40 times). AWS STS validates tokens via the JWKS endpoint rather than the thumbprint value.

Inspector2VmScannerRole-*tenant-id*

Used by the Amazon Inspector VM Scanner agent running on Azure VMs to send scan telemetry to Amazon Inspector. This role uses OIDC web identity federation through the Microsoft Entra ID identity provider.

Trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::account-id:oidc-provider/
sts.windows.net/tenant-id/"
            },
            "Action": "sts:AssumeRoleWithWebIdentity"
        }
    ]
}
```

Permission policy (TelemetryPolicy):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "inspector2-telemetry:*",
            "Resource": "*"
        }
    ]
}
```

Tags:

- `inspector-managed` = `true`

Inspector2SSMFederationRole-*tenant-id*

Enables OIDC federation to Azure for secure cross-cloud authentication. This role is assumed by the AssumeRole and DispatchRole, and by Amazon EC2 Systems Manager, to obtain web identity tokens for Azure operations.

Trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/caller": "SSM"
                },
                "ArnLike": {
                    "aws:PrincipalArn": [
                        "arn:aws:iam::account-id:role/Inspector2SSMAssumeRole-tenant-
id",
                        "arn:aws:iam::account-id:role/Inspector2SSMDispatchRole-tenant-
id"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ssm.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Permission policy (FederationPolicy):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "sts:IdentityTokenAudience": [
                        "api://AzureADTokenExchange"
                    ]
                }
            },
            "Action": [
                "sts:GetWebIdentityToken",
                "sts:TagGetWebIdentityToken"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Tags:

- `caller` = `SSM`
- `inspector-managed` = `true`

Inspector2SSMAssumeRole-*tenant-id*

Executes Amazon EC2 Systems Manager Automation to onboard Azure VMs as managed instances. This role creates and manages hybrid activations, passes roles to Amazon EC2 Systems Manager, and retrieves automation documents.

Trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ssm.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account-id"
                }
            }
        }
    ]
}
```

Permission policy (AssumeRolePolicy):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::account-id:role/
Inspector2SSMFederationRole-tenant-id"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account-id:role/
AmazonEC2RunCommandRoleForManagedInstances",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ssm.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:CreateActivation",
                "ssm:DeleteActivation",
                "ssm:DescribeActivations",
                "ssm:DescribeInstanceInformation",
                "ssm:PutConfigurePackageResult",
                "iam:ListRoleTags"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:AddTagsToResource",
            "Resource": [
                "arn:aws:ssm:*:account-id:activation/*",
                "arn:aws:ssm:*:account-id:managed-instance/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "ssm:GetDocument",
            "Resource": [
                "arn:aws:ssm:*:*:document/AWS-InstallDistributorPackageOnAzure",
                "arn:aws:ssm:*:*:document/AWS-InstallSsmAgentOnAzure",
                "arn:aws:ssm:*::document/AmazonInspector2-InspectorVmScanner"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "ssm:GetCloudConnector",
            "Resource": "*"
        }
    ]
}
```

Tags:

- `caller` = `SSM`
- `AzureTenantId` = *`tenant-id`*
- `AzureApplicationId` = *`app-id`*
- `inspector-managed` = `true`

Inspector2SSMDispatchRole-*tenant-id*

Resolves Azure VM targets and dispatches Amazon Inspector VM Scanner agent installation via Amazon EC2 Systems Manager Automation. This role starts automation executions and passes the AssumeRole to Amazon EC2 Systems Manager.

Trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ssm.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "account-id"
                }
            }
        }
    ]
}
```

Permission policy (DispatchPolicy):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account-id:role/Inspector2SSMAssumeRole-tenant-
id",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ssm.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ssm:StartAutomationExecution",
            "Resource": [
                "arn:aws:ssm:*:*:document/AWS-InstallDistributorPackageOnAzure",
                "arn:aws:ssm:*:account-id:automation-execution/*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::account-id:role/
Inspector2SSMFederationRole-tenant-id"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:DescribeInstanceInformation",
                "ssm:ListCloudConnectors",
                "ssm:GetCloudConnector",
                "iam:ListRoleTags"
            ],
            "Resource": "*"
        }
    ]
}
```

Tags:

- `caller` = `SSM`
- `inspector-managed` = `true`

The Azure app registration must have its Application ID URI set to `api://app-id`.

Each target Azure VM must have a system-assigned managed identity enabled. To automatically assign managed identity to new VMs, use a DeployIfNotExists policy in your Azure environment.

For more information about the Amazon Inspector VM Scanner agent, see *Amazon Inspector VM Scanner*.

Alternatively, you can manually install the Amazon Inspector VM Scanner agent. For manual installation at scale across many Azure VMs, use an Azure VM custom extension. For instructions, see *Amazon Inspector VM Scanner*.

To exclude specific Azure VMs from scanning, apply the following tag to the VM:

- **Key:** `InspectorExclusion`
- **Value:** `true`

Adjusting the scope of a connector

After you create a connector, you can adjust its scope by changing the Azure Subscriptions or Azure Regions that it monitors. If you adjust the scope, most changes take effect within approximately 15 minutes.

Before you adjust the scope, note the following:

- If you add a region to the scope, the console generates an updated setup script that creates the necessary Event Hub infrastructure in the new region.
- If a service-linked connector exists (created via AWS Security Hub CSPM), you cannot reduce its scope in Amazon Inspector directly. Modify the scope in AWS Security Hub CSPM instead.
- The scope of a AWS Security Hub CSPM connector can never be less than any existing customer-managed connector in Amazon Inspector for the same Azure tenant.

Verifying the connector

After you create the connector, Amazon Inspector begins discovering and scanning your Azure resources. It can take up to 24 hours for the initial set of findings to appear. To verify that the connector is working:

1. In the Amazon Inspector console, navigate to **Summary** to confirm Azure resource counts are populating.
2. Navigate to **Coverage** to verify Azure resources appear with scan status.
3. Navigate to **Integrations**, select your integration to Microsoft Azure, and verify the connector health shows `Active`.
4. For VM scanning, verify that Azure VMs appear with Amazon Inspector VM Scanner agent status in the Coverage view.
5. Navigate to **Findings** and filter by resource type to view Azure resource findings.

> **Note**
>
> Connector health status is eventually consistent. When a permission or configuration issue occurs, the connector status changes to **Degraded** quickly with an actionable message. However, after you fix the issue, the status can take up to 24 hours to return to **Connected**.
>
> - A **Degraded** status after a recent fix does not necessarily indicate an ongoing problem.

- For real-time visibility into recording failures, check the CloudWatch metrics in the namespace.

Troubleshooting

If the connector status shows Error or resources are not appearing after 24 hours, review the following:

Connector shows Error status

Verify that the Azure app registration has the correct federated identity credential and that the IAM Token Issuer URL is configured in your AWS account.

Unable to record some Azure resources

The error can occur for any of the following reasons:

- The Azure environment does not meet the prerequisites for resource recording.
- The app registration does not have admin consent for Microsoft Graph API permissions.
- The Reader role assignment does not cover all required scopes.
- The Microsoft Entra ID tenant does not have required log data.

To address the error, do the following:

1. To confirm that your environment meets all requirements, review Prerequisites for integrating Amazon Inspector with Microsoft Azure.
2. Confirm that the app registration has the required Microsoft Graph API permissions with admin consent.
3. Verify that the service principal has the Reader role at the tenant root management group scope.

Resources not appearing

Confirm that the Event Hub is receiving events. Check Event Hub metrics in the Azure portal and verify the `AWSConfig` consumer group exists.

VMs not scanning

Verify that VMs have network connectivity to Amazon EC2 Systems Manager endpoints and are running a supported operating system.

ACR images not appearing

Verify that diagnostic settings are enabled on your container registries.

Scope change not taking effect

When you change the connector scope to add a new Azure region, the console regenerates the setup script. You must re-run the updated script to provision Event Hub infrastructure in the new region.

How Amazon Inspector handles resource identifiers

By enabling your Azure integration for Amazon Inspector, resource identifiers from other cloud providers will be stored in AWS Config, Amazon Inspector, and other AWS services (as needed) as metadata related to the management of the corresponding resource configuration data collected from the other cloud providers. Such resource identifiers do not constitute Your Content, and we recommend you do not include sensitive, confidential, or personally identifiable information in them.

The following identifiers from your connected cloud environment are stored and used by AWS to provide multicloud security capabilities:

- **Resource identifiers:** Azure Tenant ID, Subscription ID, Location (region), Resource ID (Resource Group IDs or Names, Resource Provider, Resource Type)

Relationships among these identifiers - including how resources relate to one another and how findings relate to resources - are also stored as service metadata. AWS uses these identifiers for resource correlation, associating findings to resources, service operational logging, and de-duplication.

Deleting an Azure connector in Amazon Inspector

You can delete an Azure connector when you no longer want Amazon Inspector to scan resources in that Azure tenant.

What happens when you delete a connector

When you delete an Azure connector:

- Amazon Inspector stops scanning all Azure resources associated with the connector.

- Existing findings for Azure resources are closed.
- Azure resources no longer appear in the Coverage view.
- If a service-linked connector exists (created via AWS Security Hub CSPM), deleting a customer-managed connector does not affect the service-linked connector. Amazon Inspector continues scanning through the service-linked connector.
- The Azure app registration, Event Hub infrastructure, and permissions configured in your Azure environment are *not* automatically removed. You must clean up Azure resources manually if they are no longer needed by other AWS services.

Note

If you share the Azure app registration with AWS Security Hub CSPM or AWS Security Hub CSPM CSPM, do not delete the app registration after removing the Amazon Inspector connector. Other services still depend on it.

Deleting a connector

To delete an Azure connector

1. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the navigation pane, choose **Integrations**.
3. Select the Azure connector you want to delete.
4. Choose **Delete**.
5. In the confirmation dialog, enter **`delete`** and choose **Delete connector**.

You can also delete a connector by using the `inspector2:DeleteConnector` API operation.

Important

If you delete a connector and create a new one within 6 hours, resource detection may take up to 6 hours. Plan connector deletion and recreation accordingly.

Amazon Inspector SBOM Generator

A Software Bill of Materials (SBOM) is a formally structured list of components, libraries, and modules required to build a piece of software. The Amazon Inspector SBOM Generator (Sbomgen) is a tool that produces an SBOM for archives, container images, directories, local systems, and compiled Go and Rust binaries. Sbomgen scans for files that contain information about installed packages. When Sbomgen finds a relevant file, it extracts package names, versions, and other metadata. Sbomgen then transforms package metadata into a CycloneDX SBOM. You can use Sbomgen to generate the CycloneDX SBOM as a file or in STDOUT and send SBOMs to Amazon Inspector for vulnerability detection. You can also use Sbomgen as part of the CI/CD integration, which scans container images automatically as part of your deployment pipeline.

Amazon Inspector uses Sbomgen for several managed scanning features. When Amazon Inspector performs Amazon ECR container image scanning, AWS Lambda function scanning, and Agentless scanning of Amazon Elastic Compute Cloud instances through Amazon EBS volume analysis, it invokes Sbomgen behind the scenes to generate the software inventory used for vulnerability assessment. The standalone Sbomgen tool described on this page provides the same underlying technology, enabling you to integrate SBOM generation directly into your workflows.

Supported packages types

Sbomgen collects inventory for the following package types:

- Alpine APK
- Debian/Ubuntu DPKG
- Red Hat RPM
- Homebrew
- C#
- Go
- Java
- Node.js
- PHP
- Python
- Ruby
- Rust

Supported container image configuration checks

Sbomgen can scan standalone Dockerfiles and build history from exisiting images for security issues. For more information, see Amazon Inspector Dockerfile checks.

Sbomgen can also evaluate supported CIS Docker Benchmark checks on a container image with the `--enable-cis-docker` argument, and record indicators that describe a system's FIPS-related cryptographic configuration on Linux container and localhost scans with the `--enable-fips-checks` argument. Both arguments are experimental and disabled by default. For more information, see the section called “Evaluate CIS Docker Benchmark checks” and the section called “Record FIPS configuration indicators”.

Installing Sbomgen

Sbomgen is only available for Linux operating systems.

You must have Docker installed if you want Sbomgen to analyze locally cached images. Docker isn't required to analyze images exported as `.tar` files or images hosted in remote container registries.

Amazon Inspector recommends that you run Sbomgen from a system with at least the following hardware specs:

- 4x core CPU
- 8 GB RAM

To install Sbomgen

1. Download the latest Sbomgen zip file from the correct URL for your architecture:

   Linux AMD64: https://amazon-inspector-sbomgen.s3.amazonaws.com/latest/linux/amd64/inspector-sbomgen.zip

   Linux ARM64: https://amazon-inspector-sbomgen.s3.amazonaws.com/latest/linux/arm64/inspector-sbomgen.zip

   Alternatively, you can download previous versions of the Amazon Inspector SBOM Generator zip file.
2. Unzip the download using the following command:

```
unzip inspector-sbomgen.zip
```

3. Check for the following files in the extracted directory:

   - `inspector-sbomgen` – This is the tool you will execute to generate SBOMs.
   - `README.txt` – This is the documentation for using Sbomgen.
   - `LICENSE.txt` – This file contains the software license for Sbomgen.
   - `licenses` – This folder contains license info for third party packages used by Sbomgen.
   - `checksums.txt` – This file provides hashes of the Sbomgen tool.
   - `sbom.json` – This is a CycloneDX SBOM for the Sbomgen tool.
   - `WhatsNew.txt` – This file contains a summarized change log, so you can view major changes and improvements between Sbomgen versions quickly.

4. (Optional) Verify the authenticity and integrity of the tool using the following command:

   ```
   sha256sum < inspector-sbomgen
   ```

   - Compare the results with the contents of the `checksums.txt` file.

5. Grant executable permissions to the tool using the following command:

   ```
   chmod +x inspector-sbomgen
   ```

6. Verify that Sbomgen is successfully installed using the following command:

   ```
   ./inspector-sbomgen --version
   ```

   You should see the output similar to the following:

   ```
   Version: 1.X.X
   ```

Using Sbomgen

This section describes different ways you can use Sbomgen. You can learn more about how to use Sbomgen through built-in examples. To view these examples, run the `list-examples` command:

```
./inspector-sbomgen list-examples
```

Generate an SBOM for a container image and output the result

You can use Sbomgen to generate SBOMs for container images and output the result to a file. This capability can be enabled using the `container` subcommand.

Example command

In the following snippet, you can replace *`image:tag`* with the ID of your image and *`output_path.json`* with the path to the output you want to save.

```
# generate SBOM for container image
./inspector-sbomgen container --image image:tag -o output_path.json
```

> **Note**
>
> Scan time and performance depends on the image size and how small the number of layers are. Smaller images not only improve Sbomgen performance, but also reduce the potential attack surface. Smaller images also improve image build, download, and upload times.

When using Sbomgen with ScanSbom, the Amazon Inspector Scan API won't process SBOMs that contain more than 5,000 packages. In this scenario, the Amazon Inspector Scan API returns an HTTP 400 response.

If an image includes bulk media files or directories, consider excluding them from Sbomgen using the `--skip-files` argument.

Example: Common error cases

Container image scanning can fail due to the following errors:

- `InvalidImageFormat` – Occurs when scanning malformed container images with corrupt TAR headers, manifest files, or config files .
- `ImageValidationFailure` – Occurs when checksum or content-length validation fails for container image components, such as mismatched Content-Length headers, incorrect manifest digests, or failed SHA256 checksum verification.
- `ErrUnsupportedMediaType` – Occurs when image components include unsupported media types. For information about supported media types, see Supported operating systems and media types.

Amazon Inspector does not support the `application/vnd.docker.distribution.manifest.list.v2+json` media type. However, Amazon Inspector does support manifest lists. When scanning images that use manifest lists, you can explicitly specify which platform to use with the `--platform` argument. If the `--platform` argument is not specified, the Amazon Inspector SBOM Generator automatically selects the manifest based on the platform where its running.

Generate an SBOM from directories and archives

You can use Sbomgen to generate SBOMs from directories and archives. This capability can be enabled using the `directory` or `archive` subcommands. Amazon Inspector recommends using this feature when you want to generate an SBOM from a project folder, such as a downloaded git repository.

Example command 1

The following snippet shows a subcommand that generates an SBOM from a directory file.

```
# generate SBOM from directory
./inspector-sbomgen directory --path /path/to/dir -o /tmp/sbom.json
```

Example command 2

The following snippet shows a subcommand that generates an SBOM from an archive file. The only supported archive formats are `.zip`, `.tar`, and `.tar.gz`.

```
# generate SBOM from archive file (tar, tar.gz, and zip formats only)
./inspector-sbomgen archive --path testData.zip -o /tmp/sbom.json
```

Generate an SBOM from Go or Rust compiled binaries

You can use Sbomgen to generate SBOMs from compiled Go and Rust binaries. You can enable this cabapility through the `binary` subcommand:

```
./inspector-sbomgen binary --path /path/to/your/binary
```

Generate an SBOM from mounted volumes

You can use Amazon Inspector SBOM Generator to generate SBOMs from mounted volumes. This capability can be enabled using the `volume` subcommand. We recommend using this feature when

you want to analyze storage volumes, such as Amazon EBS volumes that have been mounted to your system. Unlike the directory subcommand, mounted volume scanning detects OS packages and OS information.

You can scan an Amazon EBS volume by attaching it to an Amazon EC2 instance where Amazon Inspector SBOM Generator is installed and mounting it on that instance. For Amazon EBS volumes that are currently in use by other Amazon EC2 instances, you can create an Amazon EBS snapshot of the volume and then create a new Amazon EBS volume from that snapshot for scanning purposes. For more information about Amazon EBS, see What is Amazon EBS? in the *Amazon Elastic Block Store User Guide*.

Example command

The following snippet shows a subcommand that generates an SBOM from a mounted volume. The `--path` argument should specify the root directory where the volume is mounted.

```
# generate SBOM from mounted volume
./inspector-sbomgen volume --path /mount/point/of/volume/root
```

Example command

The following snippet shows a subcommand that generates an SBOM from a mounted volume while excluding specific file paths with the `--exclude-suffix` argument. The `--exclude-suffix` argument is particularly useful when a volume contains bulk files (such as log files or media files). Files and directories whose paths end with the specified suffixes will be excluded from scanning, which can reduce scan time and memory usage.

```
# generate SBOM from mounted volume with exclusions
./inspector-sbomgen volume --path /mount/point/of/volume/root \
--exclude-suffix .log \
--exclude-suffix cache
```

All file paths in the target volume are normalized to their original paths. For example, when scanning a volume mounted at `/mnt/volume` that contains a file at `/mnt/volume/var/lib/rpm/rpmdb.sqlite`, the path will be normalized to `/var/lib/rpm/rpmdb.sqlite` in the generated SBOM.

Send an SBOM to Amazon Inspector for vulnerability identification

In addition to generating an SBOM, you can send an SBOM for scanning with a single command from the Amazon Inspector Scan API. Amazon Inspector evaluates the contents of the SBOM for vulnerabilites before returning findings to Sbomgen. Depending on your input, the findings can be displayed or written to a file.

> **Note**
>
> You must have an active AWS account with read permissions to `InspectorScan-ScanSbom` to use this capability.

To enable this capability, you pass the `--scan-sbom` argument to the Sbomgen CLI. You can also pass the `--scan-sbom` argument to any of the following Sbomgen subcommands: `archive`, `binary`, `container`, `directory`, `localhost`.

> **Note**
>
> The Amazon Inspector Scan API doesn't process SBOMs with more than 5,000 packages. In this scenario, the Amazon Inspector Scan API returns an HTTP 400 response.

You can authenticate to Amazon Inspector through an AWS profile or an IAM role with the following AWS CLI arguments:

```
--aws-profile profile
--aws-region region
--aws-iam-role-arn role_arn
```

You can also authenticate to Amazon Inspector by providing the following environment variables to Sbomgen.

```
AWS_ACCESS_KEY_ID=$access_key \
AWS_SECRET_ACCESS_KEY=$secret_key \
AWS_DEFAULT_REGION=$region \
```

```
./inspector-sbomgen arguments
```

To specify the response format, use the `--scan-sbom-output-format cyclonedx` argument or `--scan-sbom-output-format inspector` argument.

Example command 1

This command creates an SBOM for the latest Alpine Linux release, scans the SBOM, and writes the vulnerability results to a JSON file.

```
./inspector-sbomgen container --image alpine:latest \
                          --scan-sbom \
                          --aws-profile your_profile \
                          --aws-region your_region \
                          --scan-sbom-output-format cyclonedx \
                          --outfile /tmp/inspector_scan.json
```

Example command 2

This command authenticates to Amazon Inspector using AWS credentials as environment variables.

```
AWS_ACCESS_KEY_ID=$your_access_key \
AWS_SECRET_ACCESS_KEY=$your_secret_key \
AWS_DEFAULT_REGION=$your_region \
./inspector-sbomgen container --image alpine:latest \
                          -o /tmp/sbom.json \
                          --scan-sbom \
                          --scan-sbom-output-format inspector
```

Example command 3

This command authenticates to Amazon Inspector using the ARN for an IAM role.

```
./inspector-sbomgen container --image alpine:latest \
                          --scan-sbom \
                          --aws-profile your_profile \
                          --aws-region your_region \
                          --outfile /tmp/inspector_scan.json
                          --aws-iam-role-arn arn:aws:iam::123456789012:role/your_role
```

Use additional scanners to enhance detection capabilities

The Amazon Inspector SBOM Generator applies predefined scanners based on the command being used.

Default scanner groups

Each Amazon Inspector SBOM Generator subcommand applies the following default scanner groups automatically.

- For the `directory` subcommand: binary, programming-language-packages, dockerfile scanner groups
- For the `localhost` subcommand: os, programming-language-packages, extra-ecosystems scanner groups
- For the `container` subcommand: os, programming-language-packages, extra-ecosystems, dockerfile, binary scanner groups

Special scanners

To include scanners beyond the default scanner groups, use the `--additional-scanners` option followed by the name of the scanner to be added. The following is an example command showing how to do this.

```
# Add WordPress installation scanner to directory scan
./inspector-sbomgen directory --path /path/to/directory/ --additional-scanners
 wordpress-installation -o output.json
```

The following is an example command showing how to add multiple scanners with a comma-separated list.

```
./inspector-sbomgen container --image image:tag --additional-scanners scanner1,scanner2
 -o output.json
```

Optimize container scans by adjusting maximum file size to scan

When you analyze and process a container image, Sbomgen scans files that are 200 MB or less by default. Files larger than 200 MB rarely contain package metadata. You can encounter misses when you inventory a Go or Rust binary that exceeds 200MB. To adjust the size limit, use the `--max-file-size` argument. This allows you to increase the limit to include large files and decrease the limit to reduce resource usage by excluding large files.

Example

The following example shows how to use the `--max-file-size` argument to increase the file size.

```
# Increase the file size limit to scan files up to 300 MB
./inspector-sbomgen container --image alpine:latest \
--outfile /tmp/sbom.json \
--max-file-size 300000000
```

Adjusting this setting helps control disk usage, memory consumption, and overall scan duration.

Limit container image pull size

When you scan a container image, you can reject an oversized image before Sbomgen downloads any layer. Sbomgen reads the compressed size of each layer from the image manifest and compares it against the limits that you set, so an image that exceeds a limit is rejected without downloading its layers. Use the `--image-max-pull-size` and `--layer-max-pull-size` arguments to set these limits in bytes.

- `--image-max-pull-size` – Rejects the image when the compressed sizes of all of its layers add up to more than the specified number of bytes.
- `--layer-max-pull-size` – Rejects the image when any single layer's compressed size exceeds the specified number of bytes.

> **Note**
>
> Both arguments apply to container scans only and are independent of each other. Each one defaults to `0`, which disables the limit.

By default, a layer whose compressed size exceeds `--layer-max-pull-size` causes Sbomgen to reject the whole image. To skip the oversized layer and continue inventorying the rest of the image instead, add the `--continue-on-layer-size-exceeded` argument. It has no effect unless `--layer-max-pull-size` is set, and `--image-max-pull-size` still rejects an oversized image.

Each skipped layer is reported as a `data` sub-component of the image metadata component, named by its diff ID, with the following properties:

- `amazon:inspector:sbom_generator:layer:skip_reason` – Why the layer was skipped (`layer_size_exceeded`).
- `amazon:inspector:sbom_generator:layer:size_bytes` – The size of the skipped layer.
- `amazon:inspector:sbom_generator:layer:size_limit_bytes` – The limit that the layer exceeded.

> **Note**
>
> A skipped layer's whiteouts are not applied, so a path that the layer deletes or replaces is reported as the layer beneath it left it.

Example

The following example rejects the image if the compressed sizes of its layers total more than 50 GiB (`53687091200` bytes), or if any single layer's compressed size exceeds 10 GiB (`10737418240` bytes).

```
# Reject an oversized image before downloading any layer
./inspector-sbomgen container --image image:tag \
--outfile /tmp/sbom.json \
--image-max-pull-size 53687091200 \
--layer-max-pull-size 10737418240
```

Example

The following example skips any layer larger than 10 GiB (`10737418240` bytes) and inventories the remaining layers instead of rejecting the image.

```
# Skip oversized layers instead of rejecting the image
```

```
./inspector-sbomgen container --image image:tag \
--outfile /tmp/sbom.json \
--layer-max-pull-size 10737418240 \
--continue-on-layer-size-exceeded
```

Detect scratch images with filesystem heuristics

When Sbomgen can't identify a container image's operating system, you can use the `--enable-scratch-heuristics` argument to inspect the filesystem for signals that indicate a scratch image. If at least two signals are present, Sbomgen labels the operating-system component `scratch` instead of `unknown`. An identified operating system is never relabeled.

> **Note**
>
> This argument is experimental, applies to container scans only, and is disabled by default.

Sbomgen evaluates the following signals and records each one it detects as an `amazon:inspector:sbom_generator:scratch:heuristic:*` property on the operating-system component:

- `missing_lib_dir` – The `/lib` directory is absent.
- `missing_bin_dir` – The `/bin` directory is absent.
- `missing_var_dir` – The `/var` directory is absent.
- `missing_shell` – No interactive shell is present (none of `sh`, `bash`, `ash`, `dash`, `zsh`, or `busybox` found under `/bin` or `/usr/bin`).

Example

The following example shows how to use the `--enable-scratch-heuristics` argument.

```
# enable scratch image detection heuristics
./inspector-sbomgen container --image image:tag \
--outfile /tmp/sbom.json \
--enable-scratch-heuristics
```

> **Note**
>
> This behavior is heuristic and may produce false positives. It only changes the operating-system component.

Collect running process metadata

When you scan a localhost, you can use the `--collect-processes` argument to tag components with metadata about the running processes that Sbomgen gathers from the host. This helps you correlate an inventoried component with the processes that are running from it.

> **Note**
>
> This argument is opt-in, disabled by default, and applies to localhost scans only. If you provide it for any other scan type, Sbomgen logs a warning and ignores it.

When a component matches a running process, Sbomgen adds the following properties to that component, where *`pid`* is the process ID:

- `amazon:inspector:sbom_generator:host:process_id:`*`pid`*`:path` – The executable path of the matched running process, for example `/usr/sbin/sshd`.
- `amazon:inspector:sbom_generator:host:process_id:`*`pid`*`:started_at` – The time the process started, in RFC 3339 format (UTC), for example `2026-08-26T15:43:40Z`.

Example

The following example shows how to use the `--collect-processes` argument.

```
# Tag components with running process metadata
./inspector-sbomgen localhost \
--outfile /tmp/sbom.json \
--collect-processes
```

Evaluate CIS Docker Benchmark checks

When you scan a container image, you can use the `--enable-cis-docker` argument to evaluate supported CIS Docker Benchmark checks and record the result of each check as a property on the container component. Sbomgen evaluates the checks from the image configuration and build history.

> **Note**
>
> This argument is experimental, applies to container scans only, and is disabled by default.

Sbomgen evaluates the following checks:

- `4.1` – The image is configured with a non-root user.
- `4.6` – The image is configured with an enabled `HEALTHCHECK`.
- `4.7` – Every package-manager update command is followed by an install command in the same `RUN` instruction.
- `4.9` – The build history doesn't contain an `ADD` instruction.
- `4.10` – No supported secret pattern is present in an `ARG`, `ENV`, or `RUN` instruction in the build history, or in the image's final environment variables.

Sbomgen records the following properties on the container component, where *`check`* is the identifier of an individual check:

- `amazon:inspector:sbom_generator:image:cis:docker:`*`check`* – The `pass` or `fail` result of an individual check, one per supported check (currently 4.1, 4.6, 4.7, 4.9, and 4.10).
- `amazon:inspector:sbom_generator:image:cis:docker:checks_passed` – `true` if every supported check passed, or `false` if any check failed.

Example

The following example shows how to use the `--enable-cis-docker` argument.

```
# Evaluate CIS Docker Benchmark checks
./inspector-sbomgen container --image image:tag \
--outfile /tmp/sbom.json \
```

```
--enable-cis-docker
```

Record FIPS configuration indicators

When you scan a Linux container image or localhost, you can use the `--enable-fips-checks` argument to record indicators that describe the system's FIPS-related cryptographic configuration.

> **Note**
>
> This argument is experimental, applies to Linux container and localhost scans only, and is disabled by default.

Sbomgen records the following properties, each valued `pass`, `fail`, or unknown:

- `amazon:inspector:sbom_generator:fips:aide_fips_hashes`
- `amazon:inspector:sbom_generator:fips:boot_fips_arg`
- `amazon:inspector:sbom_generator:fips:crypto_policy_fips`
- `amazon:inspector:sbom_generator:fips:dracut_fips_module`
- `amazon:inspector:sbom_generator:fips:kernel_fips_enabled`
- `amazon:inspector:sbom_generator:fips:os_fips_certified`
- `amazon:inspector:sbom_generator:fips:stig_crypto_subpolicy`

Example

The following example shows how to use the `--enable-fips-checks` argument on a localhost scan.

```
# Record FIPS configuration indicators
./inspector-sbomgen localhost \
--outfile /tmp/sbom.json \
--enable-fips-checks
```

Disable progress indicator

Sbomgen displays a spinning progress indicator that can result in excessive slash characters in CI/CD environments.

```
INFO[2024-02-01 14:58:46]coreV1.go:53: analyzing artifact
|
\
/
|
\
/
INFO[2024-02-01 14:58:46]coreV1.go:62: executing post-processors
```

You can disable the progress indicator using the `--disable-progress-bar` arguement:

```
./inspector-sbomgen container --image alpine:latest \
--outfile /tmp/sbom.json \
--disable-progress-bar
```

Authenticating to private registries with Sbomgen

By providing your private registry authentication credentials, you can generate SBOMs from containers that are hosted in private registries. You can provide these credentials through the following methods:

Authenticate using cached credentials (recommended)

For this method, you authenticate to your container registry. For example, if using Docker, you can authenticate to your container registry using the Docker loging command: `docker login`.

1. Authenticate to your container registry. For example, if using Docker, you can authenticate to your registry using the Docker `login` command:
2. After you authenticate to your container registry, use Sbomgen on a container image that's in the registry. To use the following example, replace *`image:tag`* with the name of the image to scan:

```
./inspector-sbomgen container --image image:tag
```

Authenticate using the interactive method

For this method, provide your username as a parameter, and Sbomgen will prompt you for secure password entry when needed.

To use the following example, replace *image:tag* with the name of the image that you want to scan and *your_username* with a username that has access to the image:

```
./inspector-sbomgen container --image image:tag --username your_username
```

Authenticate using the non-interactive method

For this method, store your password or registry token in a `.txt` file.

> **Note**
>
> The current user should only be able to read this file. The file should also contain your password or token on a single line.

To use the following example, replace *your_username* with your username, *password.txt* with the `.txt` file that includes your password or token on a single line, and *image:tag* with the name of the image to scan:

```
INSPECTOR_SBOMGEN_USERNAME=your_username \
INSPECTOR_SBOMGEN_PASSWORD=`cat password.txt` \
./inspector-sbomgen container --image image:tag
```

Example outputs from Sbomgen

The following is an example of an SBOM for a container image inventoried using Sbomgen.

Container image SBOM

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "serialNumber": "urn:uuid:828875ef-8c32-4777-b688-0af96f3cf619",
  "version": 1,
  "metadata": {
    "timestamp": "2023-11-17T21:36:38Z",
    "tools": [
      {
        "vendor": "Amazon Web Services, Inc. (AWS)",
        "name": "Amazon Inspector SBOM Generator",
```

```
        "version": "1.0.0",
        "hashes": [
          {
            "alg": "SHA-256",
            "content":
"10ab669cfc99774786301a745165b5957c92ed9562d19972fbf344d4393b5eb1"
          }
        ]
      }
    ],
    "component": {
      "bom-ref": "comp-1",
      "type": "container",
      "name": "fedora:latest",
      "properties": [
        {
          "name": "amazon:inspector:sbom_generator:image_id",
          "value":
"sha256:c81c8ae4dda7dedc0711daefe4076d33a88a69a28c398688090c1141eff17e50"
        },
        {
          "name": "amazon:inspector:sbom_generator:layer_diff_id",
          "value":
"sha256:eddd0d48c295dc168d0710f70364581bd84b1dda6bb386c4a4de0b61de2f2119"
        }
      ]
    }
  },
  "components": [
    {
      "bom-ref": "comp-2",
      "type": "library",
      "name": "dnf",
      "version": "4.18.0",
      "purl": "pkg:pypi/dnf@4.18.0",
      "properties": [
        {
          "name": "amazon:inspector:sbom_generator:source_file_scanner",
          "value": "python-pkg"
        },
        {
          "name": "amazon:inspector:sbom_generator:source_package_collector",
          "value": "python-pkg"
        },
```

```
        {
          "name": "amazon:inspector:sbom_generator:source_path",
          "value": "/usr/lib/python3.12/site-packages/dnf-4.18.0.dist-info/METADATA"
        },
        {
          "name": "amazon:inspector:sbom_generator:is_duplicate_package",
          "value": "true"
        },
        {
          "name": "amazon:inspector:sbom_generator:duplicate_purl",
          "value": "pkg:rpm/fedora/python3-dnf@4.18.0-2.fc39?
arch=noarch&distro=39&epoch=0"
        }
      ]
    },
    {
      "bom-ref": "comp-3",
      "type": "library",
      "name": "libcomps",
      "version": "0.1.20",
      "purl": "pkg:pypi/libcomps@0.1.20",
      "properties": [
        {
          "name": "amazon:inspector:sbom_generator:source_file_scanner",
          "value": "python-pkg"
        },
        {
          "name": "amazon:inspector:sbom_generator:source_package_collector",
          "value": "python-pkg"
        },
        {
          "name": "amazon:inspector:sbom_generator:source_path",
          "value": "/usr/lib64/python3.12/site-packages/libcomps-0.1.20-py3.12.egg-
info/PKG-INFO"
        },
        {
          "name": "amazon:inspector:sbom_generator:is_duplicate_package",
          "value": "true"
        },
        {
          "name": "amazon:inspector:sbom_generator:duplicate_purl",
          "value": "pkg:rpm/fedora/python3-libcomps@0.1.20-1.fc39?
arch=x86_64&distro=39&epoch=0"
        }
```

```
      ]
    }
  ]
}
```

Previous versions of the Amazon Inspector SBOM Generator

This topic includes links to the latest and previous versions of the Amazon Inspector SBOM Generator. For information about installing Sbomgen, see Installing Sbomgen.

Latest version

- https://amazon-inspector-sbomgen.s3.amazonaws.com/latest/linux/amd64/inspector-sbomgen.zip
- https://amazon-inspector-sbomgen.s3.amazonaws.com/latest/linux/arm64/inspector-sbomgen.zip

Platform	Version	SHA-256 checksum
Linux AMD64	1.17.7	3a50a5f6f0d825aa1aef5e568a2273a0e01285c5bbfda16e3155023a8ca9d816
Linux ARM64	1.17.7	566d8a2bbe6936e80cc60a36c630a3ae706ec9a2fba18f6c18a2c0588910b3fd
Linux AMD64	1.17.6	200ce844667d227425f36ba0ab937013aeb5
Linux ARM64	1.17.6	

Platform	Version	SHA-256 checksum	
		7eae12860 b114cd240 0324a255ea e6b1f730c 314205e32 dcd19def99e6e7683c 31406e6d8 f28603e5a 02921cb4d2	
Linux AMD64 Linux ARM64	1.17.5 1.17.5	28bc3e374 541469662 988f3c7c1 a37399716 d3061204e 127763dff4f9af99e4 7 90229e9a8 d35c24b3e 4640347f2d0efd8af2 8d723abc2 cc01d1226 430190aa0f	

Platform	Version	SHA-256 checksum	
Linux AMD64 Linux ARM64	1.17.4 1.17.4	82b45c466 7253a4009 897216fda 212e05974 e90fb47c8 4a5b5c7b0 8fd059627d bbee03184 91111f550 ab632c36b b0d4cfeb1 d3ad10c74 caa2a3ed74d23eb8f4 1	
Linux AMD64 Linux ARM64	1.17.3 1.17.3	4effd790af0a4f28e5 6decd435df178df80b 256f3fb740344cfd29 64618d3104 d61dfe9d8 68965130b 9cdcc0766 15c902d97 ca94da033 ac0d6c78d 8e8ad15d9a	

Platform	Version	SHA-256 checksum	
Linux AMD64 Linux ARM64	1.17.2 1.17.2	d0ddc22ef8428a4f4ab8eb6d6feca0d3d24b1c3b65cce3a5f1845c5234fe5428 2fd1cf4919948e77a6cbde320a176ff7b5a25f439d176a3dae7955d31ab794fa	
Linux AMD64 Linux ARM64	1.17.1 1.17.1	e1c11ad4a9752ec19d836f9a67664f701aa0af83f8a5a7e786593d34fb077a80 71d3704c508b654d7ccf8b50ab09980398c59ac4b32b9a506be2571198337a9c	

Platform	Version	SHA-256 checksum	
Linux AMD64 Linux ARM64	1.17.0 1.17.0	bec950646 a347f6637 615d5d6d2 40c8490fb2f1e8fe16 b894527ca 612db6cc5e 7e10efc9cbbf5e6965 c3f2df3ed8283d3fff 4938019d6 7f0d2104d 15cb883230	
Linux AMD64 Linux ARM64	1.16.0 1.16.0	2aff31bd5f7f426020 a5cdda7a5 8aaebde72 e7dae9bc7 8f687978a d19ef487b4 c0ee096fe 6e25123b8 420bdd09a 14d0dbd15 333c017825c6cc815c e68e465d0d	

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.15.5 1.15.5	6dd27c02e9071aff17d41331edc7d4da7432 4c3b2354f85dfcb630079715662f 4098a459676f32a6e43799028e2cc674b558 ac4384fa845947cc6740571a6f3c
Linux AMD64 Linux ARM64	1.15.4 1.15.4	d379121b4b117c2c5e0ddc7b9a964fd9b1366a02b0924f93bb36d5e5edb04842 05263f81ce46e177e17a6e9b9a477f4d03d5129aabe4351d3e509cbf0c6531a5

Platform	Version	SHA-256 checksum	
Linux AMD64 Linux ARM64	1.15.3 1.15.3	3b0a8ebb8 2c1ccf683b06794cc3 0b7106185 f4c9c6e527d98c8f23 86cb9f3303 ed0f2c796b0ac2a8f5 2d2c8f5a0722f7a72e d2dca3811cc726cbe9 c29630c915	
Linux AMD64 Linux ARM64	1.15.2 1.15.2	fdab18524 63cc7590333ff3d0bb 6c088f0da01cbce287 c97a2c448e183efa64 3 96b87405d 56f48d869 7b6da2bbe 8a868b56b fc053c4278afc2456e 9ade810f9f	

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.15.1 1.15.1	405d52793 3943e56b8 84773b987 1b1a63beb 5ee06a245 7e7dd6ead 6c69217737 d57d203d1 b1085968b 4ad9109ad 638a8b069 fa15d3d0b c008eadba 8350df7a0c
Linux AMD64 Linux ARM64	1.15.0 1.15.0	bdd29a252 820265dc8 715ea69bd 15e0cbd17 16397f923e3f70fd74 6775ee82ee e85614cd3 343f02331 90e85981b b5ea18482 214a1d218 cb2287c3b 4e025a06eb

Platform	Version	SHA-256 checksum	
Linux AMD64 Linux ARM64	1.14.4 1.14.4	336d8aad1 d57c41c67 bc1f0252579f29fa16 d71d6d8a5 c5908e04599bcaf28f 3 4389ed3df 5cfd0817d 35a6284d9 3bc6457c2 48826b723 240f92e38 26a32b7e97	
Linux AMD64 Linux ARM64	1.14.3 1.14.3	a498d1919 4c4c79c1efc8a69f6e 512ee9e86 290cff7dc7fb45bd4b 647e5fa629 d06b822e6 33d9d80dc 81597d1c1 7f6df424b542c62dfd 05671bc39fd1c7c0f9 5	

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.14.2 1.14.2	26f019f09dc5f4ce7389e9cf2a852bb2ab5c918740ce13dff5ca68bccea20965 fca50fcee81f14272c1f861ee8d5507d0fdcc67d555056ae78ec18d0707a7966
Linux AMD64 Linux ARM64	1.14.1 1.14.1	2b9512ea9acdc1eb1aa155f7f664f92b4dc96e14d3849fe1f1d3aa51f481b627 f23dafdf0127dfd61af2678aaa67643bf1f0909689487febe681430302a0c8a3
Linux AMD64 Linux ARM64	1.14.0 1.14.0	f884433321762afec889acb78b935a9a6f5fb92fd4bbfa1a54ac037e9f24c351 55c62f54dd96e13c085acf946cd2a841453af00918050331a86b2b83d1f80eaa

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.13.2 1.13.2	86011ffc7ee1a85d4cb411ad807566d7fde37fe9756a9a7e5a2d22205645f2d2 d4b6328867f0cac1698ec4e0862cbdf45af068888ce3c1ed30d780b110c909af
Linux AMD64 Linux ARM64	1.13.1 1.13.1	c100ff7a65f2795ada9193aea0f2944ec67731313a6eb8b841a0ffbf0ebba2c9 3ceaac63796f6ca97e9b5ae7c29bf8c896b20b2616bacde5ea12655c30467164

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.13.0 1.13.0	f6d81b76f 46a7461a2 191a72880 a7ee90a13 ebd4c1b1a 2ab6cbc7b 469613bd93 6d69c2394df58ccf09 40f3495044b90c3f5f eca2650d7af14f2000 a537c4e96e
Linux AMD64 Linux ARM64	1.12.2 1.12.2	c1d771719 41f324519bccf78339 e71a8961b 7bc2a2635 1772c09dc 6fb9dde83f d4b0eee17 c269fcaef7361f7149 76d31cb49 4c68eadf9a9cd8e628 ee14810a7e

Platform	Version	SHA-256 checksum	
Linux AMD64 Linux ARM64	1.12.1 1.12.1	2d698b47d 2a6ba0b43 1c1161bc1fba7bef4d 9041546e6 24086d4b0dfff122a3 f c76fe1bd84ffe92709 21df1d708fc0e49c53 aa7717d29 a90d5cec3cee772f0c 2	
Linux AMD64 Linux ARM64	1.12.0 1.12.0	1112b3952 81ea5d67e 20062429b f1603ba64 56746d6941f2edfa0e e0812b34b6 25e618000 ffc4aceaf002c2fa3e 97d0564d3 76acc34e9 e442cdd76 9f6c805248	

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.11.2 1.11.2	bef68671bc532e4fb529500b62d7af836012 3cd967308 d41ad0ce8f43f7762fb 4f11d7037efa443f442c4edf7ba28774c4fa706fb7622 e4fba645b b3ad3958c9
Linux AMD64 Linux ARM64	1.11.1 1.11.1	809eb7cb8 0d24fbf6ffdd124438d53a90763 2c222e924 913ebd610 44ca949490 057f9e4c9 970aeda4b da0685e7e 02436fd5223fbe81ce c65138551 c63ed77ba0

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.11.0 1.11.0	5172a5556 cf46f9fbc5cf1d35bd 382919fb6b41aca1ec 938db3a75 530060b0cf c9e2da7b0 76dc89dc3 9a962a7dd 9c7d1fd29230a4eec7 eb95f951d 6a179093d0
Linux AMD64 Linux ARM64	1.10.1 1.10.1	9e33622a7 874adfe71 9ab7db75a 1e44f4b5fae3573374 068b501c89f0accfcfe 78d5a7f800fc26ba86 adab5b634 431a91c00 7075e06d6 ce46e5068 7d5156184e

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.10.0 1.10.0	0b7a553d7d2d17c40a62f1a11013bc46fa2c3814f407c11130e15af3fe313769 5ce9e315a4f8f90ff5eed7ab058efc8dbff6593d66d3fc455f1c37e882ec6466
Linux AMD64 Linux ARM64	1.9.1 1.9.1	d0ef4c14fec6c42e70ae55b3e44d17d027132947596e8ef861c0efc3c0e5a871 2d8145011c13f5611fc30f4510785d53e98b911717f6dbe69616af4d4b0df61f

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.9.0 1.9.0	78b377b27 30eb15476 173e40885 454ae191e 953663af3 e0928dddf b8608f4655 985bdc06d 25eccb87c4a81995c8 a2d3c78e1 c02beea30 9a620b2de 4954767591
Linux AMD64 Linux ARM64	1.8.3 1.8.3	54eed5a77 2f68320f3906bec592 0e3a19da9 04abdace1 0f985b878 59015eef89 febd74a397fb0cdd33 56072503f 08465ab87 2d1620d59 a2ab7d83b db076c929d

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.8.2 1.8.2	2e4e3c754e230046349dd975feb48fa953ea5a2de190cbbc17c1c85043936b5a 449a49e222a2bdffe0353435d7b04b0556b35a391c7b9714ce46d1a5382bc3e2
Linux AMD64 Linux ARM64	1.8.1 1.8.1	9ff7958e298d2b228b0c7617f0a9a873254587fc26aee9826c37273650b389e9 6737584fd2c7d24b56777d02846d1737f47d0121344baea217a3e5368fd98fcc

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.8.0 1.8.0	ef32e7fb4ee0af1e47 d6b528b47 293fc7127c7a7539f7 354e84452 626a4c204d 0b82ddc69 1a517bb8fc6ccd67b8 0ca566b11 7a1bb410c 05764c9b7 e3ba76c510
Linux AMD64 Linux ARM64	1.7.3 1.7.3	3fba95d44 aaea55ad0 6d3c7635a 671662c48 347457837 6d3f11e84 474f8de25f 1f4b52e3d 80de87b92 b563a78ba c4a2d898e 7af82db5b 6791d899d 516e97cfbb

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.7.2 1.7.2	c44ba9bf1cfeb3ea2d6d0b15d25816800a5045a438474f2f77c390bac41ae4cb d37c5b1605bf82260da0b0f36311c83b16464327c3fd8169ba4b3a978470c9c
Linux AMD64 Linux ARM64	1.7.1 1.7.1	b0beb602a6ae439d4e307bd99682bc8a419fd7d5e78a278bfc718eb18e00b05e 95ff2d9df2fcd1982dd705df1e763f57a0b499b6fe06801e9a80869e2e464831

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.7.0 1.7.0	a6316c2ecd5fde7091d1099335f45f0e2400b3977c92ee4d72bd1eb359320e61 9751ba5e5c6c6c0aef7d29b1c4adbd4088da3a07bb77eaa7de3f04aa33ad8562
Linux AMD64 Linux ARM64	1.6.3 1.6.3	b6a309e879aaa78d7d8e224eb5214df5fd415244d370885e6c8876db5a4181d2 59ed0b7eb7d1eadadb691f058d32634a03a856ba03ac2ddb8cd3599ceb55cb9a

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.6.2 1.6.2	8d8ba0653 5be614a4d 44b1bd74c 66d1fd487 4ff9ab788ad5e23aa5 229db9c687 2bd7b4a88 b9c6b041a6ff82f7f9 bc116b76cf410bf6eb 896fc8d68e717b55f2 a
Linux AMD64 Linux ARM64	1.6.1 1.6.1	3e3d62dc7 94b31d9d2 de1904592cf42f25e9 f42c30eb90cc53385a 60b42f1a63 ad89f6709 08fb0b48b ca0242f3ac58e7179f 6fabfcc9a2b3fd0e5c 3d79e27539

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.6.0 1.6.0	ffe671c2c1d1c2142a4af056d1c179eaffbc3925f5afaa6f3d655bd495ce5e1c a733c0b00c7225369c68ad47c57846b4546e2c9f47580ab98394baefc765c134
Linux AMD64 Linux ARM64	1.5.5 1.5.5	ebcfbe565631de5bc61b1d55d70a2d15b965f628678a2b60cffd01cd0c3443f1 a8e018ceee3a76dd4271f966015c216438b11ee807fcd970753e786baa335b56

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.5.4 1.5.4	aa8c1ffacc563b87975497f53eddec0b29397a898fac19f4902b8acb7eeb347b c6ba98d441aa88d3d3150449c098cd13ce3baeccee45ad4c9a1326f8bb8f87fc
Linux AMD64 Linux ARM64	1.5.3 1.5.3	d493c23121101c9c3df888e717bf81d7f7b81809754f3492e1ae52f02b089b68 8dfa5c97b3bd45da487706e95d1894290f53b113247bbb89b9fac16dab8184b6

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.5.2 1.5.2	ff6233d7da9f7e963589a0eb8f07bee2ca375360365cb6b6e354585cf1371910 fd31efb6031754b2bc8414d7fe9dd14a067767704145af0559b3500cc437c7ee
Linux AMD64 Linux ARM64	1.5.1 1.5.1	391fcc52117fed79cae6e92a9e225732166a6df2582aa7f6b5230149761f6732 f9bc90d18724f93db0f5ca3b79136adb7b49fa33fa179a5e87b4d512f256b56b

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.5.0 1.5.0	d7b6cb840 53358e462 d76488d01 9140ecd05 ad405217a 60a96b727 fb062880fe 067dcf5c3 02160a5270f89aed3f 941bb0571 dcb8a59f7 5dddb1b77 47c2a82ec7
Linux AMD64 Linux ARM64	1.4.0 1.4.0	c8ca73761afd742e1d eb98b04eb 5714c9c2a 574b652a7 63b18e235 60e66aea24 188d97577 82278653e 65605aaf186feda104 345ba2f9d e438873e568f1ff620 4

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.3.2 1.3.2	57dd5d135 600e84690 706cfe958 60e781499 88d37cf81 429ce97b9 256d179fb4 91526ecdafc6cc3718 fabe75b2693ace5eff b9c0af3327b484b7f5 a154929997
Linux AMD64 Linux ARM64	1.3.1 1.3.1	097ec8390 7c459a36d e11c92d016fffd64f1 c33fd4bcbf2af465e0 979b0d9237 aa93a3d40 2abc4a986 a9ad9d3de8fcca81ee 25a55596a c6dc4502e d1d6819502

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.3.0 1.3.0	21439f92c314daf136832ca6676a65d288768aa69fc6dcd2014a3038b2701eeb 4a41779b0c3b32242eedef288de6c1bf40fda0d4246b32fd0cd8d4e51e58f94b
Linux AMD64 Linux ARM64	1.2.1 1.2.1	e022e95e59f1790949bca8dbbb6478a5d3fb677ccd45aa4ba30ebd91ae86ad65 824acc5bb5b0210954fe9ab089d9461453a4975d34292cc0c676837c3a7279b4

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.2.0 1.2.0	9625b1a8a e1937ca21 79c2535a0ffceca934 138e0b66feac9ba3e3 4ffaa22ec5 7f387e560 b41571fb5 2efd9e620bf2b9e3a0 67ca781e8 8aaa977b2b8acdebf3 5
Linux AMD64 Linux ARM64	1.1.1 1.1.1	6809b7e46 675c66e3a f354c5343 3dc46c4d1 ddaf258e0 5ba15e38e 784ea0285e 6361e59fb 2448c66c4 698ea3397 9ecaaefc2af4420034 aabbbe741 242f60dbdd

Platform	Version	SHA-256 checksum
Linux AMD64 Linux ARM64	1.1.0 1.1.0	f84c88154 13d451490 b38509950 235f88713 c0c61c725 9a4831934 995664bd8f aaffefb5e44195dc55 d5fd3289e 511720f64 c130644cb d58103cf7f36e96f05 8
Linux AMD64 Linux ARM64	1.0.0 1.0.0	cc126e24962f1a6497 cf17679b3 e3b73be68 963c47e39 68a56e73caacf045b5 c 5d5bf97a4acfeaaa73 ad6c91873 8188e0c82 2e475ef37 a334e49d7 7ba907b08a

Amazon Inspector SBOM Generator comprehensive operating system collection

The Amazon Inspector SBOM Generator scans different operating systems to guarantee a robust and detailed analysis of system components. Generating an SBOM helps you understand the

composition of your operating system, so you can identify vulnerabilities in system managed packages. This topic describes key features of different operating system package collections the Amazon Inspector SBOM Generator supports. For information about the operating systems that Amazon Inspector supports, see Supported operating systems and programming languages for Amazon Inspector.

Supported operating system artifacts

The Amazon Inspector SBOM Generator supports the following operating system artifacts:

Platform	Binary	Source	Stream
Alma Linux	N/A	Yes	Yes
Alpine Linux	Yes	Yes	N/A
Amazon Linux	N/A	Yes	N/A
CentOS	N/A	Yes	N/A
Chainguard	Yes	Yes	N/A
Debian	Yes	Yes	N/A
Distroless	Yes	Yes	N/A
Fedora	N/A	Yes	N/A
MinimOS	Yes	Yes	N/A
OpenSUSE	N/A	Yes	N/A
Oracle Linux	N/A	Yes	N/A
Photon OS	N/A	Yes	N/A
RHEL	N/A	Yes	Yes
Rocky Linux	N/A	Yes	Yes
SLES	N/A	Yes	N/A

Platform	Binary	Source	Stream
Ubuntu	Yes	Yes	N/A
Windows	N/A	N/A	N/A

APK-based OS package collection

This section includes the supported platforms and key features for the APK-based OS package collection. For more information, see Alpine Package Keeper on the Alpine Linux website.

Supported platforms

The following are supported platforms.

- Alpine Linux

> **Note**
>
> For APK-based systems, the Amazon Inspector SBOM Generator collects package metadata from the `/lib/apk/db/` file.

Key features

- **Package name collection** – Extracts the name of each installed package
- **Version collection** – Extracts the version of each installed package
- **Source package identification** – Identifies the source package for each installed package

Example

The following snippet is an example of an APK database file.

```
C:Q1JlboSJkrN4qkDcokr4zenpcWEXQ=
P:zlib
V:1.2.13-r1
```

```
A:x86_64
S:54253
I:110592
T:A compression/decompression Library
U:https://zlib.net/
L:Zlib
o:zlib
```

DPKG-based OS package collection

This section includes the supported platforms and key features for the DPKG-based OS package collection. For more information, see Debian Package on the Debian website.

Supported platforms

The following platforms are supported.

- Debian
- Ubuntu

> **Note**
>
> For DPKG-based systems, the Amazon Inspector SBOM Generator collects package metadata from the `/var/lib/dpkg/status` file.

Key features

The following are key features for DPKG-based OS packages.

- **Package name collection** – Extracts the name of each installed package
- **Version collection** – Extracts the version of each installed package
- **Source package identification** – Identifies the source package for each installed package

Example

The following snippet is an example of a `/var/lib/dpkg/` file.

```
Package: zlib1g
Status: install ok installed
Priority: optional
Section: libs
Installed-Size: 168
Maintainer: Mark Brown <broonie@debian.org>
Architecture: amd64
Multi-Arch: same
Source: zlib
Version: 1:1.2.13.dfsg-1
Provides: libz1
Depends: libc6 (>= 2.14)
Breaks: libxml2 (<< 2.7.6.dfsg-2), texlive-binaries (<< 2009-12)
Conflicts: zlib1 (<= 1:1.0.4-7)
Description: compression library - runtime
 zlib is a library implementing the deflate compression method found
 in gzip and PKZIP.  This package includes the shared library.
Homepage: http://zlib.net/
```

RPM-based OS package collection

This section includes the supported platforms and key features for the RPM-based OS package collection. For more information, see RPM Package Manager on the RPM website.

Supported platforms

The following platforms are supported.

- Alma Linux
- Amazon Linux
- CentOS
- Fedora
- OpenSUSE
- Oracle Linux
- PhotonOS
- RedHat Enterprise Linux
- Rocky Linux

- SUSE Linux Enterprise Server

> **Note**
>
> For RPM-based systems, the Amazon Inspector SBOM Generator collects package metadata from the `/var/lib/rpm` file.

Key features

The following are key features for RPM-based OS package collections.

- **Package name collection** – Extracts the name of each installed package
- **Version collection** – Extracts the version of each installed package
- **Source package identification** – Identifies the source package for each installed package
- **Stream support** – Extracts stream metadata of each installed package

Example

The following is an example of an RPM database file snippet.

```
/usr/lib/sysimage/rpm/rpmdb.sqlite
/usr/lib/sysimage/rpm/Packages
/usr/lib/sysimage/rpm/Packages.db
/var/lib/rpm/rpmdb.sqlite
/var/lib/rpm/Packages
/var/lib/rpm/Packages.db
```

Windows OS version collection

Unlike Linux-based operating systems, Windows does not use a package management system for the operating system itself. The Amazon Inspector SBOM Generator collects only the Windows OS version information. For Windows application scanning, use the windows-apps scanner instead. The windows-apps scanner collects information about installed applications on Windows systems. For more information, See Microsoft applications ecosystem collection.

Key features

- **OS version collection** – Extracts the Windows OS version from the Windows Registry. The extracted OS version is used for vulnerability detection for Windows OS.

Registry keys and values

The following Windows Registry keys and values are used to collect OS name and version information.

- **Registry Key**

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion
```

- **Registry Value**
  - ProductName – OS name and edition (e.g., "Windows Server 2025 Datacenter")
  - CurrentMajorVersionNumber – the major version of OS
  - CurrentMinorVersionNumber – The minor version of OS
  - CurrentBuild – The build number of OS
  - UBR – The revision number of OS

Chainguard image package collection

This section includes the supported platforms and key features for Chainguard image package collection. For more information, see Images on the Chainguard website.

Supported platforms

The following platforms are supported

- Wolfi Linux

> **Note**
>
> For Chainguard images, the Amazon Inspector SBOM Generator collects package metadata from the `/lib/apk/db/installed` file.

Key features

The following are key features.

- **Package name collection** – Extracts the name of each installed package
- **Version collection** – Extracts the version of each installed package
- **Source package identification** – Identifies the source package for each installed package

Example

The following snippet is an example of a Chainguard image file.

```
P:wolfi-keys
V:1-r8
A:x86_64
L:MIT
T:Wolfi signing keyring
o:wolfi-keys
```

Distroless image package collection

Distroless containers are container images that exclude package managers, shells, and other utilities in Linux distributions. Distroless containers only include essential dependencies required to run the application and improve performance and security.

> **Note**
>
> For Distroless images, the Amazon Inspector SBOM Generator collects package metadata from the `/var/lib/dpkg/status.d` file. Only Debian and Ubuntu-based distributions are supported. These can be identified by the `NAME` field in the `/etc/os-release` file system, which shows "Debian" or "Ubuntu."

Key features

- **Package name collection** – Extracts the name of each installed package
- **Version collection** – Extracts the version of each installed package

Example

The following is an example of a Distroless image file.

```
Package: tzdata
Version: 2021a-1+deb11u10
Architecture: all
Maintainer: GNU Libc Maintainers <debian-glibc@lists.debian.org>
Installed-Size: 3413
Depends: debconf (>= 0.5) | debconf-2.0
Provides: tzdata-bullseye
Section: localization
Priority: required
Multi-Arch: foreign
Homepage: https://www.iana.org/time-zones
Description: time zone and daylight-saving time data
 This package contains data required for the implementation of
 standard local time for many representative locations around the
 globe. It is updated periodically to reflect changes made by
 political bodies to time zone boundaries, UTC offsets, and
 daylight-saving rules.
```

MinimOS package collection

This section includes the supported platforms and key features for Minimus image package collection. For more information, see the Minimus website.

Supported platforms

The following platforms are supported.

- MinimOS

> **Note**
>
> For Minimus images, the Amazon Inspector SBOM Generator collects package metadata from the `/lib/apk/db/installed` file.

Key features

The following are key features.

- Package name collection – Extracts the name of each installed package
- Version collection – Extracts the name of each installed package
- Source package identification – Identifies the source package for each installed package

The following is a snippet of a Minimus image file.

```
P:ca-certificates-bundle
V:20241121-r1
A:aarch64
L:MPL-2.0 AND MIT
T:
o:ca-certificates
```

Docker Hardened Images package collection

Docker Hardened Images (DHI) are minimal, security-hardened container images. The Amazon Inspector SBOM Generator detects DHI images and namespaces their packages under `dhi` so that vulnerabilities are matched against the hardened distribution.

Detection

DHI version 2 images report `ID=dhi` in the `/etc/os-release` file, with `ID_LIKE=alpine` or `ID_LIKE=debian` indicating the underlying base operating system. The Amazon Inspector SBOM Generator detects DHI images directly from this `ID` value and does not parse the `PRETTY_NAME` field.

Example `/etc/os-release` file

The following is an example of an `/etc/os-release` file for a Debian-based DHI image.

```
ID=dhi
ID_LIKE=debian
```

Package namespace

Every package in a DHI image is namespaced under `dhi`, yielding a Package URL of the form `pkg:deb/dhi/<name>@<version>` or `pkg:apk/dhi/<name>@<version>`. The PURL type (`deb` or `apk`) reflects the underlying base operating system.

Example PURL

The following are example package URLs for packages in DHI images.

```
# Debian-based DHI image
pkg:deb/dhi/<name>@<version>

# Alpine-based DHI image
pkg:apk/dhi/<name>@<version>
```

Hardened-image markers

When the Amazon Inspector SBOM Generator detects `ID=dhi`, it annotates the image with hardened-image markers:

- The operating-system component name is annotated with `<base> (Docker Hardened Image)` – for example, `Debian GNU/Linux (Docker Hardened Image)`.
- The container or operating-system component carries the `amazon:inspector:sbom_generator:hardened_image:vendor` property with a value of `Docker`.

> **Note**
>
> DHI detection applies to container, localhost, and volume scans.

Red Hat Hardened Images package collection

Red Hat Hardened Images are minimal, security-hardened container images built on Hummingbird OS. The Amazon Inspector SBOM Generator detects these images and namespaces their packages under `hummingbird` so that vulnerabilities are matched against the hardened distribution.

Detection

Red Hat Hardened Images report `ID=hummingbird` in the `/etc/os-release` file. The Amazon Inspector SBOM Generator detects these images directly from this `ID` value.

Example `/etc/os-release` file

The following is an example of an `/etc/os-release` file for a Red Hat Hardened Image.

```
ID=hummingbird
```

Package namespace

Every package in a Red Hat Hardened Image is namespaced under `hummingbird`, yielding a Package URL of the form `pkg:rpm/hummingbird/<name>@<version>`.

Example PURL

The following is an example package URL for a package in a Red Hat Hardened Image.

```
pkg:rpm/hummingbird/<name>@<version>
```

Hardened-image markers

When the Amazon Inspector SBOM Generator detects `ID=hummingbird`, it annotates the image with hardened-image markers:

- The container or operating-system component carries the `amazon:inspector:sbom_generator:hardened_image:vendor` property with a value of `Red Hat`.

> **Note**
>
> Red Hat Hardened Images detection applies to container, localhost, and volume scans.

CleanStart hardened images package collection

CleanStart images are minimal, security-hardened container images built on an Alpine base. The Amazon Inspector SBOM Generator detects these images and namespaces their packages under `cleanstart` so that vulnerabilities are matched against the hardened distribution.

Detection

CleanStart images report `ID=clnstrt` in the `/etc/os-release` file. The Amazon Inspector SBOM Generator detects these images directly from this `ID` value.

Example `/etc/os-release` file

The following is an example of an `/etc/os-release` file for a CleanStart image.

```
ID=clnstrt
ID_LIKE=alpine
```

Package namespace

Every package in a CleanStart image is namespaced under `cleanstart`, yielding a Package URL of the form `pkg:apk/cleanstart/<name>@<version>`.

Example PURL

The following is an example package URL for a package in a CleanStart image.

```
pkg:apk/cleanstart/<name>@<version>
```

Hardened-image markers

When the Amazon Inspector SBOM Generator detects `ID=clnstrt`, it annotates the image with hardened-image markers:

- The container or operating-system component carries the `amazon:inspector:sbom_generator:hardened_image:vendor` property with a value of `CleanStart`.

Programming language dependency collection

The Amazon Inspector SBOM Generator supports different programming languages and frameworks, which make up a robust and detailed collection of dependencies. Generating an SBOM helps you understand the composition of your software, so you can identify vulnerabilities and maintain compliance with security standards. The Amazon Inspector SBOM Generator supports the following programming languages and file formats.

Go dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
Go	Go	`go.mod`	N/A	N/A	N/A	N/A	Yes
		`go.sum`	N/A	N/A	N/A	N/A	Yes
		Go Binaries	Yes	N/A	N/A	N/A	Yes
		`GOMODCACHE`	N/A	N/A	N/A	N/A	No

go.mod/go.sum

Use `go.mod` and `go.sum` files to define and lock dependencies in Go projects. The Amazon Inspector SBOM Generator manages these files differently based on the Go toolchain version.

Key features

- Collects dependencies from `go.mod` (if the Go toolchain version is 1.17 or higher)
- Collects dependencies from `go.sum` (if the Go toolchain version is 1.17 or lower)
- Parses `go.mod` to identify all declared dependencies and dependency versions

Example `go.mod` file

The following is an example of `go.mod` file.

```
module example.com/project

go 1.17

require (
github.com/gin-gonic/gin v1.7.2
golang.org/x/crypto v0.0.0-20210616213533-5cf6c0f8e123
)
```

Example `go.sum` file

The following is an example of `go.sum` file.

```
github.com/gin-gonic/gin v1.7.2 h1:VZ7DdRl0sghbA6lVGSkX+UXO2+J0aH7RbsNugG+FA8Q=
github.com/gin-gonic/gin v1.7.2/go.mod h1:ILZ1Ngh2f1pL1ASUj7gGk8lGFeNC8cRTaN2ZhsBNbXU=
golang.org/x/crypto v0.0.0-20210616213533-5cf6c0f8e123 h1:b6rCu+qHze
+BUsmC3CZzH8aNu8LzPZTVsNTo64OypSc=
golang.org/x/crypto v0.0.0-20210616213533-5cf6c0f8e123/go.mod h1:K5Dkpb0Q4ewZW/
EzWlQphgJcUMBCzoWrLfDOVzpTGVQ=
```

> **Note**
>
> Each of these files produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Go Binaries

The Amazon Inspector SBOM Generator extracts dependencies from compiled Go binaries to provide assurance about the code in use.

Note

The Amazon Inspector SBOM Generator supports capturing and evaluating toolchain versions from Go binaries built using the official Go compiler. For more information, see Download and install on the Go website. If you are using the Go toolchain from another vendor, such as Red Hat, evaluation might not be accurate due to potential differences in distribution and metadata availability.

Key features

- Extracts dependency information directly from Go binaries
- Collects dependencies embedded within the binary
- Detects and extracts the Go toolchain version used for compiling the binary.

GOMODCACHE

The Amazon Inspector SBOM Generator scans the Go module cache to collect information about installed dependencies. This cache stores downloaded modules to make sure the same versions are used across different builds.

Key features

- Scans the `GOMODCACHE` directory to identify cached modules
- Extracts detailed metadata, including module names, versions, and source URLs

Example structure

The following is an example of the `GOMODCACHE` structure.

```
~/go/pkg/mod/
### github.com/gin-gonic/gin@v1.7.2
### golang.org/x/crypto@v0.0.0-20210616213533-5cf6c0f8e123
```

Note

This structure produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Java dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
Java	Maven	Compiled Java applications (.jar/.war/.ear)	N/A	N/A	Yes	N/A	Yes
		pom.xml	N/A	N/A	Yes	N/A	Yes

Note

Our vulnerability evaluation feature only supports Maven Central repository. Third-party repositories, such as JBoss Enterprise Maven Repository, are not currently supported.

The Amazon Inspector SBOM Generator performs Java dependency scanning by analyzing compiled Java applications and `pom.xml` files. When scanning compiled applications, the scanner generates SHA–1 hashes for integrity verification, extracts embedded `pom.properties` files, and parses nested `pom.xml` files.

SHA–1 hash collection (for compiled .jar, .war, .ear files)

The Amazon Inspector SBOM Generator tries to collect SHA–1 hashes for all `.ear`, `.jar`, and `.war` files in a project to guarantee the integrity and traceability of compiled Java artifacts.

Key features

- Generates SHA–1 hashes for all compiled Java artifacts

Example artifact

The following is an example of an SHA–1 artifact.

```
{
  "bom-ref": "comp-52",
  "type": "library",
  "name": "jul-to-slf4j",
  "version": "2.0.6",
  "hashes": [
    {
      "alg": "SHA-1",
      "content": ""
    }
  ],
  "purl": "pkg:maven/jul-to-slf4j@2.0.6",
  "properties": [
    {
      "name": "amazon:inspector:sbom_generator:source_path",
      "value": "test-0.0.1-SNAPSHOT.jar/BOOT-INF/lib/jul-to-slf4j-2.0.6.jar"
    }
  ]
}
```

Note

This artifact produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

pom.properties

The `pom.properties` file is used in Maven projects to store project metadata, including package names and package versions. The Amazon Inspector SBOM Generator parses this file to collect project information.

Key features

- Parses and extracts package artifacts, package groups, and package versions

Example `pom.properties` file

The following is an example of a `pom.properties` file.

```
#Generated by Maven
#Tue Mar 16 15:44:02 UTC 2021

version=1.6.0
groupId=net.datafaker
artifactId=datafaker
```

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Excluding nested `pom.xml` parsing

If you want to exclude `pom.xml` parsing when scanning compiled Java applications, use the `--skip-nested-pomxml` argument.

pom.xml

The `pom.xml` file is the core configuration file for Maven projects. It contains information about projects and project dependencies. The Amazon Inspector SBOM Generator parses pom.xml files to collect dependencies, scanning standalone files in repositories and files inside compiled .jar files.

Key features

- Parses and extracts package artifacts, package groups, and package versions from `pom.xml` files.

Supported Maven scopes and tags

Dependencies are collected with the following Maven scopes:

- compile
- provided
- runtime
- test
- system
- import

Dependencies are collected with the following Maven tag: `<optional>true</optional>`.

Example `pom.xml` file with a scope

The following is an example of a `pom.xml` file with a scope.

```
<dependency>
<groupId>jakarta.servlet</groupId>
<artifactId>jakarta.servlet-api</artifactId>
</version>6.0.0</version>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.28</version>
```

```
<scope>runtime</scope>
</dependency>
```

Example `pom.xml` file without a scope

The following is an example of a `pom.xml` file without a scope.

```
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.17.1</version>
</dependency>

<dependency>
<groupId>org.jenkins-ci.plugins</groupId>
<artifactId>plain-credentials</artifactId>
<version>183.va_de8f1dd5a_2b_</version>
</dependency>

<dependency>
<groupId>org.jenkins-ci.plugins</groupId>
<artifactId>jackson2-api</artifactId>
<version>2.15.2-350.v0c2f3f8fc595</version>
</dependency>
```

Note

Each of these files produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

JavaScript dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
Javascript	Node Modules	node_modules/*/package.json	N/A	N/A	Yes	Yes	Yes
	NPM		N/A	Yes	N/A	N/A	No
	PNPM		N/A	Yes	N/A	N/A	No
	YARN		N/A	Yes	N/A	N/A	No
		package-lock.json (v1, v2, and v3) / npm-shrinkwrap.json					
		pnpm-lock.yaml					
		yarn.lock					

package.json

The `package.json` file is a core component of Node.js projects. It contains metadata about installed packages. The Amazon Inspector SBOM Generator scans this file to identify package names and package versions.

Key features

- Parses the JSON file structure to extract package names and versions
- Identifies private packages with private values

Example `package.json` file

The following is an example of a `package.json` file.

```
{
"name": "arrify",
"private": true,
"version": "2.0.1",
"description": "Convert a value to an array",
"license": "MIT",
"repository": "sindresorhus/arrify"
}
```

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

package-lock.json

The `package-lock.json` file is automatically generated by npm to lock exact versions of dependencies installed for a project. It ensures consistency in environments by storing exact versions of all dependencies and their sub-dependencies. This file can distinguish between regular dependencies and development dependencies.

Key features

- Parses the JSON file structure to extract package names and package versions
- Supports dev dependency detection

Example `package-lock.json` file

The following is an example of a `package-lock.json` file.

```
"verror": {
"version": "1.10.0",
"resolved": "https://registry.npmjs.org/verror/-/verror-1.10.0.tgz",
"integrity": "sha1-OhBcoXBTr1XW4nDB+CiGguGNpAA=",
"requires": {
    "assert-plus": "^1.0.0",
    "core-util-is": "1.0.2",
    "extsprintf": "^1.2.0"
}
},
"wrappy": {
"version": "1.0.2",
"resolved": "https://registry.npmjs.org/wrappy/-/wrappy-1.0.2.tgz",
"integrity": "sha1-tSQ9jz7BqjXxNkYFvA0QNuMKtp8=",
"dev": true
},
"yallist": {
"version": "3.0.2",
"resolved": "https://registry.npmjs.org/yallist/-/yallist-3.0.2.tgz",
"integrity": "sha1-hFK0u36Dx8GI2AQcGoN8dz1ti7k="
}
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

npm-shrinkwrap.json

npm automatically generates`package-lock.json` and `npm-shrinkwrap.json` files to lock exact versions of dependencies installed for a project. This guarantees consistency in environments

by storing exact versions of all dependencies and sub-dependencies. The files distinguish between regular dependencies and development dependencies.

Key features

- Parse `package-lock` versions 1 ,2, and 3 of the JSON file structure to extract the package name and version
- Developer dependency detection is supported (`package-lock.json` captures production and development dependencies, allowing tools to identify which packages are used in development environments)
- The `npm-shrinkwrap.json` file is prioritized over the `package-lock.json` file

Example

The following is an example of a `package-lock.json` file.

```
"verror": {
            "version": "1.10.0",
            "resolved": "https://registry.npmjs.org/verror/-/verror-1.10.0.tgz",
            "integrity": "sha1-OhBcoXBTr1XW4nDB+CiGguGNpAA=",
            "requires": {
                "assert-plus": "^1.0.0",
                "core-util-is": "1.0.2",
                "extsprintf": "^1.2.0"
            }
        },
        "wrappy": {
            "version": "1.0.2",
            "resolved": "https://registry.npmjs.org/wrappy/-/wrappy-1.0.2.tgz",
            "integrity": "sha1-tSQ9jz7BqjXxNkYFvA0QNuMKtp8=",
            "dev": true
        },
        "yallist": {
            "version": "3.0.2",
            "resolved": "https://registry.npmjs.org/yallist/-/yallist-3.0.2.tgz",
            "integrity": "sha1-hFK0u36Dx8GI2AQcGoN8dz1ti7k="
}
```

pnpm-yaml.lock

The `pnpm-lock.yaml` file is generated by pnpm to maintain a record of installed dependency versions. It also tracks development dependencies separately.

Key features

- Parses the YAML file structure to extract package names and versions
- Supports dev dependency detection

Example

The following is an example of a `pnpm-lock.yaml` file.

```
lockfileVersion: 5.3
importers:
my-project:
dependencies:
  lodash: 4.17.21
devDependencies:
  jest: 26.6.3
specifiers:
  lodash: ^4.17.21
  jest: ^26.6.3
packages:
/lodash/4.17.21:
resolution:
  integrity: sha512-xyz
engines:
  node: '>=6'
dev: false
/jest/26.6.3:
resolution:
  integrity: sha512-xyz
dev: true
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

yarn.lock

The Amazon Inspector SBOM Generator tries to collect SHA–1 hashes for `.ear`, `.jar`, and `.war` files in a project to guarantee the integrity and traceability of compiled Java artifacts.

Key features

- Generates SHA–1 hashes for all compiled Java artifacts

Example SHA–1 artifact

The following is an example of an SHA–1 artifact.

```
"@ampproject/remapping@npm:^2.2.0":
version: 2.2.0
resolution: "@ampproject/remapping@npm:2.2.0"
dependencies:
"@jridgewell/gen-mapping": ^0.1.0
"@jridgewell/trace-mapping": ^0.3.9
checksum:
 d74d170d06468913921d72430259424b7e4c826b5a7d39ff839a29d547efb97dc577caa8ba3fb5cf023624e9af9d09
languageName: node
linkType: hard

"@babel/code-frame@npm:^7.0.0, @babel/code-frame@npm:^7.12.13, @babel/code-
frame@npm:^7.18.6, @babel/code-frame@npm:^7.21.4":
version: 7.21.4
resolution: "@babel/code-frame@npm:7.21.4"
dependencies:
"@babel/highlight": ^7.18.6
checksum:
 e5390e6ec1ac58dcef01d4f18eaf1fd2f1325528661ff6d4a5de8979588b9f5a8e852a54a91b923846f7a5c681b217
```

```
languageName: node
linkType: hard
```

> **Note**
>
> This artifact produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

.NET dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
.NET	.NET Core	*.deps.json	N/A	N/A	N/A	N/A	Yes
	Nuget	Packages.config	N/A	N/A	N/A	N/A	Yes
	Nuget	packages.lock.json	N/A	N/A	Yes	N/A	Yes
	.NET	.csproj	N/A	N/A	N/A	N/A	Yes

Packages.config

The `Packages.config` file is an XML file used by an older version of Nuget to manage project dependencies. It lists all the packages referenced by the project, including specific versions.

Key features

- Parses XML structure to extract package IDs and versions

Example

The following is an example of a `Packages.config` file.

```
<?xml version="1.0" encoding="utf-8"? >
<packages>
<package id="FluentAssertions" version="5.4.1" targetFramework="net461" />
<package id="Newtonsoft.Json" version="11.0.2" targetFramework="net461" />
<package id="SpecFlow" version="2.4.0" targetFramework="net461" />
<package id="SpecRun.Runner" version="1.8.0" targetFramework="net461" />
<package id="SpecRun.SpecFlow" version="1.8.0" targetFramework="net461" />
<package id="SpecRun.SpecFlow.2-4-0" version="1.8.0" targetFramework="net461" />
<package id="System.ValueTuple" version="4.5.0" targetFramework="net461" />
</packages>
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

*.deps.json

The `*.deps.json` file is generated by .NET Core projects and contains detailed information about all dependencies, including paths, versions, and runtime dependencies. This file makes sure the runtime has necessary information to load correct versions of dependencies.

Key features

- Parses the JSON structure for comprehensive dependency details
- Extracts package names and versions in a `libraries` list.

Example `.deps.json` file

The following is an example of a `.deps.json` file.

```
{
"runtimeTarget": {
    "name": ".NETCoreApp,Version=v7.0",
    "signature": ""
},
"libraries": {
    "sample-Nuget/1.0.0": {
        "type": "project",
        "serviceable": false,
        "sha512": ""
    },
    "Microsoft.EntityFrameworkCore/7.0.5": {
        "type": "package",
        "serviceable": true,
        "sha512": "sha512-
RXbRLHHWP2Z3pq8qcL5nQ6LPeoOyp8hasM5bd0Te8PiQi3RjWQR4tcbdY5XMqQ+oTO9wA8/RLhZRn/
hnxlTDnQ==",
        "path": "microsoft.entityframeworkcore/7.0.5",
        "hashPath": "microsoft.entityframeworkcore.7.0.5.nupkg.sha512"
    },
}
```

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

packages.lock.json

The `packages.lock.json` file is used by newer versions of Nuget to lock exact versions of dependencies for a .NET project to guarantee the same versions are used consistently across different environments.

Key features

- Parses the JSON structure to list locked dependencies
- Supports both direct and transitive dependencies
- Extracts package name and resolved versions

Example `packages.lock.json` file

The following is an example of a `packages.lock.json` file.

```
{
"version": 1,
"dependencies": {
"net7.0": {
  "Microsoft.EntityFrameworkCore": {
    "type": "Direct",
    "requested": "[7.0.5, )",
    "resolved": "7.0.5",
    "contentHash": "RXbRLHHWP2Z3pq8qcL5nQ6LPeoOyp8hasM5bd0Te8PiQi3RjWQR4tcbdY5XMqQ
+oTO9wA8/RLhZRn/hnxlTDnQ==",
    "dependencies": {
      "Microsoft.EntityFrameworkCore.Abstractions": "7.0.5",
      "Microsoft.EntityFrameworkCore.Analyzers": "7.0.5",
      "Microsoft.Extensions.Caching.Memory": "7.0.0",
      "Microsoft.Extensions.DependencyInjection": "7.0.0",
      "Microsoft.Extensions.Logging": "7.0.0"
    }
  },
  "Newtonsoft.Json": {
    "type": "Direct",
    "requested": "[13.0.3, )",
    "resolved": "13.0.3",
    "contentHash": "HrC5BXdl00IP9zeV+0Z848QWPAoCr9P3bDEZguI+gkLcBKAOxix/tLEAAHC
+UvDNPv4a2d18lOReHMOagPa+zQ=="
  },
  "Microsoft.Extensions.Primitives": {
    "type": "Transitive",
    "resolved": "7.0.0",
    "contentHash": "um1KU5kxcRp3CNuI8o/GrZtD4AIOXDk
+RLsytjZ9QPok3ttLUelLKpilVPuaFT3TFjOhSibUAso0odbOaCDj3Q=="
  }
```

```
}
}
}
```

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

.csproj

The `.csproj` file is written in XML and the project file for .NET projects. It includes references to Nuget packages, project properties, and build configurations.

Key features

- Parses XML the structure to extract package references

Example `.csproj` file

The following is an example of a `.csproj` file.

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<TargetFramework>net7.0</TargetFramework>
<RootNamespace>sample_Nuget</RootNamespace>
<ImplicitUsings>enable</ImplicitUsings>
<Nullable>enable</Nullable>
<RestorePackagesWithLockFile>true</RestorePackagesWithLockFile>
</PropertyGroup>
<ItemGroup>
</ItemGroup>
<ItemGroup>
<PackageReference Include="Newtonsoft.Json" Version="13.0.3" />
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.5" />
</ItemGroup>
```

```
</Project>
```

Example `.csproj` file

The following is an example of a `.csproj` file.

```
<PackageReference Include="ExamplePackage" Version="6.*" />
<PackageReferencePackageReference Include="ExamplePackage" Version="(4.1.3,)" />
<PackageReference Include="ExamplePackage" Version="(,5.0)" />
<PackageReference Include="ExamplePackage" Version="[1,3)" />
<PackageReference Include="ExamplePackage" Version="[1.3.2,1.5)" />
```

Note

Each of these files produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

PHP dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
PHP	Composer	`composer.lock`	N/A	N/A	Yes	N/A	Yes
		`/vendor/composer/i`	N/A	N/A	Yes	N/A	Yes

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
		`nstalled.json`					

composer.lock

The `composer.lock` file is automatically generated when running the composer install or composer update commands. This file guarantees the same versions of dependencies are installed in every environment. This provides a consistent and reliable build process.

Key features

- Parses the JSON format for structured data
- Extracts dependency names and versions

Example `composer.lock` file

The following is an example of a `composer.lock` file.

```
{
"packages": [
    {
        "name": "nesbot/carbon",
        "version": "2.53.1",
        // TRUNCATED
    },
    {
        "name": "symfony/deprecation-contracts",
        "version": "v3.2.1",
        // TRUNCATED
    },
    {
        "name": "symfony/polyfill-mbstring",
        "version": "v1.27.0",
        // TRUNCATED
```

```
    }
]
// TRUNCATED
}
```

> **Note**
>
> This produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

/vendor/composer/installed.json

The `/vendor/composer/installed.json` file is located in the `vendor/composer` directory and provides a comprehensive list of all installed packages and package versions.

Key features

- Parses the JSON format for structured data
- Extracts dependency names and version

Example `/vendor/composer/installed.json` file

The following is an example of a `/vendor/composer/installed.json` file.

```
{
"packages": [
    {
        "name": "nesbot/carbon",
        "version": "2.53.1",
        // TRUNCATED
    },
    {
        "name": "symfony/deprecation-contracts",
        "version": "v3.2.1",
        // TRUNCATED
```

```
    },
    {
        "name": "symfony/polyfill-mbstring",
        "version": "v1.27.0",
        // TRUNCATED
    }
]
// TRUNCATED
}
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Python dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
Python	pip	requirements.txt	N/A	N/A	N/A	N/A	Yes
	Poetry	Poetry.lock	N/A	N/A	N/A	N/A	Yes
	Pipenv	Pipfile.lock	N/A	N/A	N/A	N/A	Yes
	uv	uv.lock	N/A	Yes	N/A	N/A	Yes
	Egg/Wheel	.egg-info	N/A	N/A	N/A	N/A	Yes
			N/A	N/A	N/A	N/A	Yes

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
		/PKG-INFO .dist-info/METADATA					

requirements.txt

The `requirements.txt` file is a widely used format in Python projects to specify project dependencies. Each line in this file includes a package with its version constraints. The Amazon Inspector SBOM Generator parses this file to identify and catalog dependencies accurately.

Key features

- Supports version specifiers (== and ~=)
- Supports comments and complex dependency lines

> **Note**
>
> The version specifiers <= and => aren't supported.

Example `requirements.txt` file

The following is an example of a `requirements.txt` file.

```
flask==1.1.2
requests==2.24.0
numpy==1.18.5
foo~=1.2.0
```

```
# Comment about a dependency
scipy. # invalid
```

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Pipfile.lock

Pipenv is a tool bringing the best of all packaging worlds (bundled, pinned, and unpinned). The `Pipfile.lock` locks exact versions of dependencies to facilitate deterministic builds. The Amazon Inspector SBOM Generator reads this file to list dependencies and their resolved versions.

Key features

- Parses the JSON format for dependency resolution
- Supports default and development dependencies

Example `Pipfile.lock` file

The following is an example of a `Pipfile.lock` file.

```
{
"default": {
    "requests": {
        "version": "==2.24.0",
        "hashes": [
            "sha256:cc718bb187e53b8d"
        ]
    }
},
"develop": {
    "blinker": {
        "hashes": [
```

```
            "sha256:1779309f71bf239144b9399d06ae925637cf6634cf6bd131104184531bf67c01",
            "sha256:8f77b09d3bf7c795e969e9486f39c2c5e9c39d4ee07424be2bc594ece9642d83"
        ],
        "markers": "python_version >= '3.8'",
        "version": "==1.8.2"
    }
}
}
```

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Poetry.lock

Poetry is a dependency management and packaging tool for Python. The `Poetry.lock` file locks exact versions of dependencies to facilitate consistent environments. The Amazon Inspector SBOM Generator extracts detailed dependency information from this file.

Key features

- Parses the TOML format for structured data
- Extracts dependency names, and versions

Example `Poetry.lock` file

The following is an example of a `Poetry.lock` file.

```
[[package]]
name = "flask"
version = "1.1.2"
description = "A simple framework for building complex web applications."
category = "main"
optional = false
```

```
python-versions = ">=3.5"
[[package]]
name = "requests"
version = "2.24.0"
description = "Python HTTP for Humans."
category = "main"
optional = false
python-versions = ">=3.5"
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

uv.lock

uv is a fast Python package manager written in Rust. The `uv.lock` file locks exact versions of dependencies to facilitate consistent environments. The Amazon Inspector SBOM Generator extracts detailed dependency information from this file.

Key features

- Parses the TOML formatted `uv.lock` for structured data
- Extracts dependency names, and versions
- Supports development dependencies
- Collects only packages where the source is a registry

Example `uv.lock` file

The following is an example of a `uv.lock` file.

```
version = 1
requires-python = ">=3.12"
```

```
[[package]]
name = "flask"
version = "3.1.0"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "blinker" },
    { name = "click" },
    { name = "itsdangerous" },
    { name = "jinja2" },
    { name = "markupsafe" },
    { name = "werkzeug" },
]

[[package]]
name = "pytest"
version = "8.3.4"
source = { registry = "https://pypi.org/simple" }
dependencies = [
    { name = "iniconfig" },
    { name = "packaging" },
    { name = "pluggy" },
]

[package.dev-dependencies]
dev = [
    { name = "pytest" },
]
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Egg/Wheel

For globally installed Python packages, the Amazon Inspector SBOM Generator supports parsing metadata files found in the `.egg-info/PKG-INFO` and `.dist-info/METADATA` directories. These files provide detailed metadata about installed packages.

Key features

- Extracts package name, and version
- Supports both egg and wheel formats

> **Expanded default scan paths (Sbomgen 1.13.0 and later)**
>
> In Amazon Inspector SBOM Generator release 1.13.0 and later, the scanner's default localhost scan paths were expanded to include user-level `pip` installs (`pip install --user`) and installs produced by the `uv tool install` command. This enables detection of Python packages that are installed outside system-wide `site-packages`, including frameworks and tools such as CrewAI, Aider, GPT Engineer, and Open Interpreter, whose recommended install methods produce installs in these locations (for example, `uv tool install crewai`).
> Added default scan paths:
>
> - Linux – `/home/*/.local/lib/python*/site-packages/`, `/home/*/.local/share/uv/tools/*/lib/python*/site-packages/` (with `/root/` equivalents for containers running as root).
> - Windows – `%APPDATA%\Python\Python*\Lib\site-packages\`, `%LOCALAPPDATA%\uv\tools\*\Lib\site-packages\`.
>
> For non-default Python install prefixes on any platform, use the `--path` argument to point the scanner at the install location.

Example `PKG-INFO/METADATA` file

The following is an example of a `PKG-INFO/METADATA` file.

```
Metadata-Version: 1.2
Name: Flask
Version: 1.1.2
Summary: A simple framework for building complex web applications.
Home-page: https://palletsprojects.com/p/flask/
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Ruby dependency scanning

Programming language	Package manager	Supported artifacts	Toolchaing support	Development dependencies	Transitive dependencies	Private flag	Recursively
Ruby	Bundler	Gemfile.lock	N/A	N/A	Yes	N/A	Yes
		.gemspec	N/A	N/A	N/A	N/A	Yes
		globall installed Gems	N/A	N/A	N/A	N/A	Yes

Gemfile.lock

The Gemfile.lock file locks exact versions of all dependencies to make sure the same versions are used in every environment.

Key features

- Parses the Gemfile.lock file to identity dependencies and dependency versions
- Extracts detailed package names and package versions

Example Gemfile.lock file

The following is an example of a Gemfile.lock file.

```
GEM
remote: https://rubygems.org/
specs:
ast (2.4.2)
awesome_print (1.9.2)
diff-lcs (1.5.0)
json (2.6.3)
parallel (1.22.1)
parser (3.2.2.0)
nokogiri (1.16.6-aarch64-linux)
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

.gemspec

The `.gemspec` file is a RubyGem file containing metadata about a gem. The Amazon Inspector SBOM Generator parses this file to collect detailed information about a gem.

Key features

- Parses and extracts the gem name and gem version

Note

Reference specification is not supported.

Example `.gemspec` file

The following is an example of a `.gemspec` file.

```
Gem::Specification.new do |s|
s.name        = "generategem"
s.version     = "2.0.0"
s.date        = "2020-06-12"
s.summary     = "generategem"
s.description = "A Gemspec Builder"
s.email       = "edersondeveloper@gmail.com"
s.files       = ["lib/generategem.rb"]
s.homepage    = "https://github.com/edersonferreira/generategem"
s.license     = "MIT"
s.executables = ["generategem"]
s.add_dependency('colorize', '~> 0.8.1')
end
```

```
# Not supported

Gem::Specification.new do |s|
s.name        = &class1
s.version     = &foo.bar.version
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Globally installed gems

The Amazon Inspector SBOM Generator supports scanning globally installed gems, which are located in standard directories, such as `/usr/local/lib/ruby/gems/<ruby_version>/gems/` in Amazon EC2/Amazon ECR and `ruby/gems/<ruby_version>/gems/` in Lambda. This makes sure all globally installed dependencies are identified and cataloged.

Key features

- Identifies and scans all globally installed gems in standard directories

- Extracts metadata and version information for each globally installed gem

Example directory structure

The following is an example of a directory structure.

```
.
### /usr/local/lib/ruby/3.5.0/gems/
### actrivesupport-6.1.4
### concurrent-ruby-1.1.9
### i18n-1.8.10
```

> **Note**
>
> This structure produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Rust dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
Rust	Cargo.toml	Cargo.toml	N/A	N/A	N/A	N/A	Yes
		Cargo.lock	N/A	N/A	Yes	N/A	Yes
		Rust binary (built	Yes	N/A	N/A	N/A	Yes

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
		with cargo-auditable)					

Cargo.toml

The `Cargo.toml` file is the manifest file for Rust projects.

Key features

- Parses and extracts the `Cargo.toml` file to identify the project package name and version.

Example `Cargo.toml` file

The following is an example of a `Cargo.toml` file.

```
[package]
name = "wait-timeout"
version = "0.2.0"
description = "A crate to wait on a child process with a timeout specified across Unix
 and\nWindows platforms.\n"
homepage = "https://github.com/alexcrichton/wait-timeout"
documentation = "https://docs.rs/wait-timeout"
readme = "README.md"
categories = ["os"]
license = "MIT/Apache-2.0"
repository = "https://github.com/alexcrichton/wait-timeout"
[target."cfg(unix)".dependencies.libc]
version = "0.2"
[badges.appveyor]
repository = "alexcrichton/wait-timeout"
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Cargo.lock

The `Cargo.lock` file locks dependency versions to make sure the same versions are used whenever a project is built.

Key features

- Parses the `Cargo.lock` file to identify all dependencies and dependency versions.

Example `Cargo.lock` file

The following is an example of a `Cargo.lock` file.

```
# This file is automatically @generated by Cargo.
# It is not intended for manual editing.
[[package]]
name = "adler32"
version = "1.0.3"
source = "registry+https://github.com/rust-lang/crates.io-index"

[[package]]
name = "aho-corasick"
version = "0.7.4"
source = "registry+https://github.com/rust-lang/crates.io-index"
```

Note

This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can

be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Rust binaries with cargo-auditable

The Amazon Inspector SBOM Generator collects dependencies from Rust binaries built with the `cargo-auditable` library. This provides additional dependency information by enabling dependency extraction from compiled binaries.

Key features

- Extracts dependency information directly from Rust binaries built with the `cargo-auditable` library
- Retrieves metadata and version information for dependencies included in the binaries

> **Note**
>
> This file produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Swift dependency scanning

Programming language	Package manager	Supported artifacts	Toolchain support	Development dependencies	Transitive dependencies	Private flag	Recursively
Swift	Swift Package Manager	Package.resolved (v1, v2, and v3)	N/A	N/A	N/A	N/A	Yes

Package.resolved

The `Package.resolved` file is generated by Swift Package Manager to lock exact versions of dependencies for a Swift project. The Amazon Inspector SBOM Generator parses this file to collect dependency names, versions, and source repository information. Schema versions 1, 2, and 3 of the `Package.resolved` format are supported.

> **Note**
>
> The Amazon Inspector SBOM Generator does not generate the `Package.resolved` file. If your project does not already contain a `Package.resolved` file, you must generate it before scanning by running `swift package resolve` in your project directory.

Supported artifacts

- `Package.resolved` (schema version 1)
- `Package.resolved` (schema version 2)
- `Package.resolved` (schema version 3)

Key features

- Extracts dependency names and locked versions from pinned packages
- Derives the package namespace from the source repository URL (for example, `github.com/Alamofire`)
- Skips branch-only pins that don't have a resolved version
- Filters out local source control and file system dependencies
- Excludes `Package.resolved` files inside `.build/` directories to prevent collecting dependencies from the Swift Package Manager build cache

> **Note**
>
> The `Package.resolved` file is platform-independent. The Amazon Inspector SBOM Generator discovers this file recursively on Linux, macOS, and Windows.

Example `Package.resolved` file (v2)

The following is an example of a version 2 `Package.resolved` file.

```
{
  "pins" : [
    {
      "identity" : "alamofire",
      "kind" : "remoteSourceControl",
      "location" : "https://github.com/Alamofire/Alamofire.git",
      "state" : {
        "revision" : "a1b2c3...",
        "version" : "5.8.1"
      }
    },
    {
      "identity" : "swift-argument-parser",
      "kind" : "remoteSourceControl",
      "location" : "https://github.com/apple/swift-argument-parser.git",
      "state" : {
        "revision" : "d4e5f6...",
        "version" : "1.3.0"
      }
    }
  ],
  "version" : 2
}
```

Example `Package.resolved` file (v1)

The following is an example of a version 1 `Package.resolved` file.

```
{
  "object": {
    "pins": [
      {
        "package": "Alamofire",
        "repositoryURL": "https://github.com/Alamofire/Alamofire.git",
        "state": {
          "branch": null,
```

```
          "revision": "a1b2c3...",
          "version": "5.8.1"
        }
      }
    ]
  },
  "version": 1
}
```

Example PURL

The following is an example package URL for a Swift Package Manager dependency.

```
pkg:swift/github.com/Alamofire/Alamofire@5.8.1
```

> **Note**
>
> Each of these files produces an output that contains a package URL. This URL can be used to specify information about software packages when generating a software bill of materials and can be included in the ScanSbom API. For more information, see package-url on the GitHub Website.

Unsupported artifacts

This section describes unsupported artifacts.

Java

The Amazon Inspector SBOM Generator generator only supports vulnerability detection for dependencies sourced from the mainstream Maven repository. Private or custom Maven repositories, such as Red Hat Maven and Jenkins, aren't supported. For accurate vulnerability detection, make sure Java dependencies are pulled from the mainstream Maven repository. Dependencies from other repositories won't be covered in vulnerability scans.

JavaScript

esbuild bundles

For esbuild minified bundles, the Amazon Inspector SBOM Generator doesn't support dependency scanning for projects using esbuild. Source maps generated by esbuild don't include sufficient metadata(dependency names and versions) required for accurate Sbomgen generation. For reliable results, scan the original project files, such as the `node_modules/directory` and `package-lock.json`, prior to the bundling process.

package.json

The Amazon Inspector SBOM Generator doesn't support scanning the root-level package.json file for dependency information. This file only specifies package names and version ranges, but doesn't include fully resolved package versions. For accurate scanning results, use `package.json` or other lock files, such as `yarn.lock` and `pnpm.lock`, that include resolved versions.

Dotnet

When using floating versions or version ranges in `PackageReference`, it becomes more challenging to determine the exact package version used in a project without performing package resolution. Floating versions and version ranges allow developers to specify a range of acceptable package versions rather than a fixed version.

Go binaries

The Amazon Inspector SBOM Generator doesn't scan Go binaries that are built with build flags configured to exclude the build ID. These build flags prevent Amazon Inspector SBOM Generator from accurately mapping the binary to its original source. Unclear Go binaries aren't supported due to the inability to extract package information. For accurate dependency scanning, make sure that Go binaries are built with default settings, including the build ID.

Rust binaries

The Amazon Inspector SBOM Generator only scans Rust binaries if the binaries are built using the cargo-auditable library. Rust binaries not utilizing this library lack necessary metadata for accurate dependency extraction. The Amazon Inspector SBOM Generator extract the compiled Rust toolchain version starting from Rust 1.7.3, but only for binaries in a Linux environment. For comprehensive scanning, build Rust binaries on Linux using cargo-auditable.

> **Note**
>
> Vulnerability detection for the Rust toolchain itself isn't supported, even if the toolchain version is extracted.

Amazon Inspector SBOM Generator comprehensive ecosystem collection

The Amazon Inspector SBOM Generator is a tool for creating a software bill of materials (SBOM) and performing vulnerability scanning for supported packages from operating systems and programming languages. It supports the scanning of various ecosystems beyond core operating systems, ensuring a robust and detailed analysis of infrastructure components. By generating an SBOM, you can understand the composition of modern technology stacks, identify vulnerabilities in ecosystem components, and gain visibility into third-party software.

Supported ecosystems

The ecosystem collection extends SBOM generation beyond packages installed through OS package managers. This is done through the collection of applications deployed through alternative methods, such as manual installation. These application collectors run under the `extra-ecosystems` scanner group, which is applied by default for container and localhost scans. For other scan types, add a scanner with the `--additional-scanners` option. For more information, see Using the Amazon Inspector SBOM Generator. The Amazon Inspector SBOM Generator supports scanning for the following ecosystems:

Ecosystems	Applications
7-Zip	7-Zip archiver (version 21.07 and higher)
AI agent skills	Claude Cursor Fast-Agent OpenClaw

Ecosystems	Applications
	Cross-client
AI IDE extensions	Continue Cody Cline
Amazon Q Developer	Amazon Q Developer CLI Amazon Q VS Code extension Amazon Q JetBrains plugin
Anthropic	Claude Code CLI Claude Desktop
Anysphere	Cursor
Apache	Apache Cassandra Apache httpd Apache Struts Apache tomcat
Atlassian	Jira Core Confluence Jira Software Jira Service Management
Codeium	Windsurf

Ecosystems	Applications
Conda	Miniconda environments Anaconda environments Miniforge environments Mambaforge environments
Curl	Curl Libcurl
Docker binaries	docker dockerd containerd runc
Elasticsearch	Elasticsearch
Generic AI/ML Models	`.gguf` – GGUF `.safetensors` – Safetensors `.onnx` – ONNX `.pt` – PyTorch `.pth` – PyTorch `.h5` – Keras / HDF5
GitHub	GitHub Copilot CLI GitHub Copilot VS Code extension GitHub Copilot JetBrains plugin
Google	Chrome

Ecosystems	Applications
HuggingFace	HuggingFace CLI Models Cache
Jan	Jan
Java	JDK JRE Amazon Corretto
Jenkins	Jenkins (version 2.400.* and higher)
Kiro	Kiro CLI
LM Studio	LM Studio
MariaDB and MySQL	MariaDB Server (10.6+, 11.x, 12.x) Oracle MySQL Server (8.0, 8.4, 9.4+) MySQL NDB Cluster
Metabase	Metabase

Ecosystems	Applications
Microsoft applications	PowerShell NuGet CLI Visual Studio Code Microsoft Edge SharePoint Server Microsoft Defender Exchange Server Visual Studio .NET Core Runtime .NET Framework ASP.NET Core Runtime Microsoft Teams Outlook for Windows Microsoft Office Microsoft 365
Microsoft SQL Server	Microsoft SQL Server
MongoDB	MongoDB Server (7.0+, 8.0+)
Mozilla	Firefox Firefox ESR Thunderbird
Nginx	Nginx

Ecosystems	Applications
Node	Node
Node.js	node
Nomic AI	GPT4All
Ollama	Ollama Ollama Model Collector
OpenSSH	OpenSSH (versions 9 and 10)
OpenSSL	OpenSSL
Oracle	Oracle Database Server
PHP	PHP (version 8.1 and higher)
Redis	Redis (version 7.2 and higher)
TabbyML	Tabby
WordPress	core plugin theme
Zed Industries	Zed

7-Zip ecosystem collection

Supported applications

- 7-Zip archiver (version 21.07 and higher)

Key features

- Examines 7-Zip binaries to extract the embedded version information.

> **Note**
>
> Specifically, it searches for the product version value in the binary.

Supported platforms – Windows

- `C:\Program Files\7-Zip\7z.exe`
- `C:\Program Files\7-Zip\7za.exe`
- `C:\Program Files\7-Zip\7zz.exe`
- `C:\Program Files\7-Zip\7zr.exe`
- `C:\Program Files (x86)\7-Zip\7z.exe`
- `C:\Program Files (x86)\7-Zip\7za.exe`
- `C:\Program Files (x86)\7-Zip\7zz.exe`
- `C:\Program Files (x86)\7-Zip\7zr.exe`

Example PURL

The following is an example package URL for 7-Zip.

```
pkg:generic/7zip/7zip@25.01
```

AI agent skills ecosystem collection

Supported applications

- Claude
- OpenClaw
- Cursor
- Fast-Agent
- Cross-client

Key features

- Detects `SKILL.md` files in default skill directories for supported AI agent skills.

- Supports the cross-client standard path (`~/.agents/skills/`) and agent-specific paths (for example, `~/.claude/skills/`).
- Skills are not versioned. Each skill is identified by the SHA-256 hash of its `SKILL.md` file.

Supported platforms

For localhost scans, the Amazon Inspector SBOM Generator scans the following default paths. For container, directory, and archive scans, `SKILL.md` files are detected anywhere in the filesystem.

Linux

- `/home/<user>/.agents/skills/`
- `/home/<user>/<agent>/skills/`

macOS

- `/Users/<user>/.agents/skills/`
- `/Users/<user>/<agent>/skills/`

Windows

- `C:\Users\<user>\.agents\skills\`
- `C:\Users\<user>\<agent>\skills\`

Example PURL

The following is an example package URL for an AI agent skill.

```
# Claude skill
pkg:generic/agentskills/claude/web-design-guidelines

# Cross-client skill
pkg:generic/agentskills/cross-client/web-design-guidelines
```

AI IDE extensions ecosystem collection

The `ai-ide-extensions` collector is a single consolidated scanner that detects AI coding assistant extensions installed in Visual Studio Code and JetBrains IDEs. Detected extensions are

reported as CycloneDX components with the `application` type and a `generic` Package URL type.

Supported applications

- Continue
- Cody (by Sourcegraph)
- Cline

Key features

- Detects the extension across both supported IDE families. For Visual Studio Code extensions, the version is read from the extension's `package.json` file. For JetBrains plugins, the version is read from the `plugin.xml` file.
- Each supported extension is identified by its IDE extension ID (Visual Studio Code) or plugin directory (JetBrains):
  - Continue – Visual Studio Code extension ID `continue.continue-*`, JetBrains plugin directory `continue`
  - Cody – Visual Studio Code extension ID `sourcegraph.cody-ai-*`
  - Cline – Visual Studio Code extension ID `saoudrizwan.claude-dev-*`

Supported platforms

Cross-platform. The Amazon Inspector SBOM Generator scans wherever the IDE stores its extensions.

Scanner name

`ai-ide-extensions`

Example PURL

The following are example package URLs for the supported AI IDE extensions.

```
# Continue
pkg:generic/continue-dev/continue@<version>

# Cody
```

```
pkg:generic/sourcegraph/cody@<version>

# Cline
pkg:generic/cline/cline@<version>
```

Amazon Q Developer ecosystem collection

Supported applications

- Amazon Q Developer CLI
- Amazon Q VS Code extension
- Amazon Q JetBrains plugin

Key features

- Examines the `q` CLI binary for the embedded `AmazonQ-For-CLI` identity string and extracts the version from AWS tooling user-agent metadata.
- For IDE extensions, parses `package.json` (VS Code) and `META-INF/plugin.xml` (JetBrains) to extract the version.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux and macOS

- `~/.local/bin/q`
- `/usr/local/bin/q`
- `~/.vscode/extensions/amazonwebservices.amazon-q-vscode-*/`
- `<IDE>/plugins/amazon-q/META-INF/plugin.xml`

Windows

- `%APPDATA%\npm\q.cmd`
- `%USERPROFILE%\.vscode\extensions\amazonwebservices.amazon-q-vscode-*\`

- `%APPDATA%\JetBrains\<IDE>\plugins\amazon-q\`

Example PURL

The following is an example package URL for Amazon Q Developer.

```
Sample PURL: pkg:generic/amazon/amazon-q@1.19.7?distro=linux
```

Apache ecosystem collection

This section provides details about Apache Cassandra, Apache httpd, Apache Struts, and Apache tomcat applications.

Apache Cassandra

Supported applications

- Apache Cassandra

Key features

- Discovers `apache-cassandra-<specific.version>.jar` files and unpacks them to extract the version string from the `Implementation-Version` entry inside the `META-INF/MANIFEST.MF` file.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `/usr/share/cassandra/lib/`
- `/opt/cassandra/lib/`

macOS

- `/opt/homebrew/Cellar/cassandra/`

- /usr/local/Cellar/cassandra/

Windows

- C:\Program Files\cassandra\lib\
- C:\Program Files\apache-cassandra\lib\
- C:\Program Files (x86)\cassandra\lib\
- C:\Program Files (x86)\apache-cassandra\lib\

Example apache-cassandra-<specific.version>.jar/META-INF/MANIFEST.MF file

The following is an example of content inside an apache-cassandra-<specific.version>.jar/META-INF/MANIFEST.MF file.

```
//truncated

Manifest-Version: 1.0
Implementation-Title: Cassandra
Implementation-Version: 4.1.3
Implementation-Vendor: Apache

//truncated
```

Example PURL

The following is an example package URL for an Apache Cassandra application.

```
Sample PURL: pkg:generic/apache/cassandra@4.1.3
```

Apache httpd

Supported applications

- Apache httpd

> **Note**
>
> Vulnerability evaluation only applies to Apache httpd version 2.0 and higher.

Key features

- Parses the `/include/ap_release.h` file to extract installation macros, which contain major identifier strings, minor identifier strings, and patch identifier strings.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Unix

- `/usr/local/apache2/include/`

Windows

- `C:\Apache24\include\`
- `C:\Program Files\Apache24\include\`
- `C:\Program Files (x86)\Apache24\include\`

Example `ap_release.h` file

The following is an example of content inside an `ap_release.h` file.

```
//truncated

#define AP_SERVER_BASEVENDOR "Apache Software Foundation"
#define AP_SERVER_BASEPROJECT "Apache HTTP Server"
#define AP_SERVER_BASEPRODUCT "Apache"

#define AP_SERVER_MAJORVERSION_NUMBER 2
#define AP_SERVER_MINORVERSION_NUMBER 4
#define AP_SERVER_PATCHLEVEL_NUMBER   1
```

```
#define AP_SERVER_DEVBUILD_BOOLEAN    0

//truncated
```

Example PURL

The following is an example package URL for an Apache httpd application.

```
Sample PURL: pkg:generic/apache/httpd@2.4.1
```

Apache Struts

Supported applications

- Apache Struts

Key features

- Discovers `struts2-core-<specific.version>.jar` files (typically deployed inside a web application's `WEB-INF/lib/` directory on a Java application server) and unpacks them to extract the version string from the `Implementation-Version` entry inside the `META-INF/MANIFEST.MF` file.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common Java application server paths across platforms:

Linux

- `/opt/tomcat/`
- `/usr/share/tomcat/`
- `/var/lib/tomcat/`
- `/usr/local/tomcat/`
- `/opt/wildfly/`

- /opt/jboss/
- /opt/jetty/

Windows

- C:\Program Files\Apache Software Foundation\
- C:\Program Files (x86)\Apache Software Foundation\

Example struts2-core-<specific.version>.jar/META-INF/MANIFEST.MF file

The following is an example of content inside a struts2-core-<specific.version>.jar/META-INF/MANIFEST.MF file.

```
//truncated

Manifest-Version: 1.0
Implementation-Title: Struts 2 Core
Implementation-Version: 6.4.0
Implementation-Vendor: Apache Software Foundation

//truncated
```

Example PURL

The following is an example package URL for an Apache Struts application.

```
Sample PURL: pkg:generic/apache/struts@6.4.0
```

Apache tomcat

Supported applications

- Apache tomcat

> **Note**
>
> Vulnerability evaluation only applies to Apache tomcat version 9.0 and higher.

Key features

- Unpacks the `catalina.jar` file to extract installation macros inside the `META-INF/MANIFEST.MF` file, which contains the version string.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `/opt/tomcat/lib/`
- `/usr/share/tomcat/lib/`
- `/var/lib/tomcat/lib/`

macOS

- `/Library/Tomcat/lib/`
- `/usr/local/tomcat/lib/`

Windows

- `C:\Program Files\Apache Software Foundation\`
- `C:\Program Files (x86)\Apache Software Foundation\`

Example `catalina.jar/META-INF/MANIFEST.MF` file

The following is an example of content inside a `catalina.jar/META-INF/MANIFEST.MF` file.

```
//truncated
```

```
Implementation-Title: Apache Tomcat
Implementation-Vendor: Apache Software Foundation
Implementation-Version: 10.1.31

//truncated
```

Example PURL

The following is an example package URL for an Apache tomcat application.

```
Sample PURL: pkg:generic/apache/tomcat@10.1.31
```

Atlassian ecosystem collection

This section provides details about Atlassian server products and applications.

Atlassian Server Products

Supported applications

- Jira Core
- Confluence

Key features

- Jira Core – Parses Maven POM properties from `atlassian-jira-webapp` to extract version information.
- Confluence – Parses Maven POM properties from `confluence-webapp` to extract version information.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths:

Linux

- `/opt/atlassian/jira/atlassian-jira/META-INF/maven/com.atlassian.jira/atlassian-jira-webapp/pom.properties`
- `/opt/atlassian/confluence/confluence/META-INF/maven/com.atlassian.confluence/confluence-webapp/pom.properties`

Example PURL

The following are example package URLs for Atlassian server products.

```
// Jira Core
pkg:generic/atlassian/jira-core@10.0.1?distro=linux

// Confluence
pkg:generic/atlassian/confluence@9.2.7?distro=linux
```

Atlassian Applications

Supported applications

- Jira Software
- Jira Service Management

Key features

- Jira Software – Detects via `jira-software-application` JAR and extracts version from Maven POM properties.
- Jira Service Management – Detects via `jira-servicedesk-application` JAR and extracts version from Maven POM properties.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths:

Linux

- `/opt/atlassian/jira/atlassian-jira/WEB-INF/application-installation/jira-software-application/jira-software-application-*.jar`

- `/opt/atlassian/jira/atlassian-jira/WEB-INF/application-installation/jira-servicedesk-application/jira-servicedesk-application-*.jar`

Example PURL

The following are example package URLs for Atlassian applications.

```
// Jira Software
pkg:generic/atlassian/jira-software@10.3.9?distro=linux

// Jira Service Management
pkg:generic/atlassian/jira-service-management@10.3.9?distro=linux
```

Claude Code ecosystem collection

Supported applications

- Claude Code CLI (Anthropic)

Key features

- Claude Code is distributed as an npm package (`@anthropic-ai/claude-code`). The scanner discovers the `claude` CLI binary and resolves `package.json` relative to it using the standard npm `<prefix>/bin/<binary>` to `<prefix>/lib/node_modules/@anthropic-ai/claude-code/package.json` layout.
- Version is read from the `version` field of `package.json`.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux and macOS

- `/usr/bin/claude`
- `/usr/local/bin/claude`
- `~/.local/bin/claude`
- `~/.npm-global/bin/claude`

Windows

- `%APPDATA%\npm\claude.cmd`

Example PURL

The following is an example package URL for Claude Code.

```
Sample PURL: pkg:generic/anthropic/claude-code@1.0.5?distro=linux
```

Conda ecosystem collection

Supported applications

- Miniconda environments
- Anaconda environments
- Miniforge environments
- Mambaforge environments

Key features

- Parses `conda-meta/*.json` package records written by conda and mamba in each conda environment.
- Extracts the package name, version, build string, subdir (target platform), and archive type (`conda` or `tar.bz2`) per CEP 26.
- Generates conda Package URLs.
- Filters out macOS resource-fork files (`._*`) and skips records with an empty name or version.

Supported platforms

For localhost scans, the Amazon Inspector SBOM Generator scans conda-meta directories at the following installer-default and per-user paths. For container, directory, and archive scans, the Amazon Inspector SBOM Generator detects `conda-meta/*.json` records anywhere in the tree.

Linux

- `/opt/conda`

- /opt/miniconda3, /opt/anaconda3, /opt/miniforge3
- /root/miniconda3, /root/anaconda3, /root/miniforge3
- /home/<user>/miniconda3, /home/<user>/anaconda3, /home/<user>/miniforge3
- ~/.conda/envs/<env>

macOS

- /opt/miniconda3, /opt/anaconda3
- /opt/homebrew/anaconda3, /usr/local/anaconda3
- /opt/homebrew/Caskroom/miniconda/base, /opt/homebrew/Caskroom/miniforge/base
- /Users/<user>/miniconda3, /Users/<user>/anaconda3, /Users/<user>/miniforge3
- ~/.conda/envs/<env>

Windows

- C:\ProgramData\miniconda3, C:\ProgramData\anaconda3, C:\ProgramData\miniforge3
- C:\Users\<user>\miniconda3, C:\Users\<user>\anaconda3, C:\Users\<user>\miniforge3
- C:\Users\<user>\.conda\envs\<env>

Example `conda-meta` record path

The following is an example path for a conda package record.

```
/opt/miniforge3/conda-meta/numpy-1.24.0-py311h64a7726_0.json
```

Example PURL

The following are example package URLs for a platform-specific conda package and a `noarch` conda package.

```
pkg:conda/numpy@1.24.0?build=py311h64a7726_0&subdir=linux-64&type=conda
```

```
pkg:conda/flask@3.0.0?build=pyhd8ed1ab_0&subdir=noarch&type=conda
```

Curl ecosystem collection

This section provides details about Curl and Libcurl applications.

Curl

Supported applications

- Curl

Supported platforms

- Unix – Linux and macOS
  - `/usr/local/bin/curl`

Key features – Curl

- Examines curl binaries to extract the embedded version information.

> **Note**
>
> Specifically, it searches for version strings in the binary executable `.rodata` section (for ELF binaries on Linux), `.rdata` section (for PE binaries on Windows), or `__cstring` section (for Mach-O binaries on macOS).

Curl version string

The following is an example of a version string embedded in a Curl binary:

```
curl/8.14.1
```

Version `8.14.1` is extracted from the string to identify the Curl version.

Example PURL (Curl)

The following is an example package URL for a Curl binary.

```
Sample PURL: pkg:generic/curl/curl@8.14.1
```

Libcurl

Supported applications

- Libcurl

Supported platforms

- Unix – Linux and macOS
  - `/usr/local/include/curl/curlver.h`

Key features – Libcurl

- Examines `curlver.h` to extract embedded version information for Libcurl.

> **Note**
>
> Specifically, it extracts the version from the defined `LIBCURL_VERSION_MAJOR`, `LIBCURL_VERSION_MINOR`, and `LIBCURL_VERSION_PATCH` variables.

Libcurl version string

The following is an example of the version variables in a `curlver.h` file:

```
#define LIBCURL_VERSION_MAJOR 8
#define LIBCURL_VERSION_MINOR 14
#define LIBCURL_VERSION_PATCH 1
```

Version `8.14.1` is extracted from these lines to identify the Libcurl version.

Example PURL (Libcurl)

The following is an example package URL for a Libcurl version file.

```
Sample PURL: pkg:generic/curl/libcurl@8.14.1
```

Cursor ecosystem collection

Supported applications

- Cursor (by Anysphere)

Key features

- Examines installations of Cursor, an Electron-based AI code editor by Anysphere. Extracts the version from the application's `resources/app/package.json` Electron manifest.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `/usr/share/cursor/`
- `/usr/bin/cursor`
- `~/.local/bin/cursor`

Windows

- `%LOCALAPPDATA%\Programs\cursor\`
- `<SystemDrive>\Program Files\cursor\`

Example PURL

The following is an example package URL for Cursor.

```
Sample PURL: pkg:generic/cursor/cursor@0.42.3?distro=linux
```

Docker binaries ecosystem collection

Supported applications

- docker (docker version 20.10.18+)

- dockerd (docker version 20.10.18+)
- containerd (docker version 20.10.18+)
- runc (docker version 29.0.0+)

> **Note**
>
> Vulnerability evaluation only applies to Docker versions 25.0.0+. Earlier versions of Docker binaries may be collected, but they are no longer supported by Docker.

Key features

- Examines Docker static binaries to extract version information.

Supported platforms

The Amazon Inspector SBOM Generator scans for binaries in the installation paths recommended in Docker's static binary installation documentation:

Linux

- `/usr/bin/`

macOS

- `/usr/local/bin/`

Windows

- `C:\Program Files\Docker\`

Example PURL

The following is an example package URL for Docker.

```
pkg:generic/docker/docker@29.4.0
pkg:generic/docker/dockerd@29.4.0
```

```
pkg:generic/docker/containerd@2.2.2
pkg:generic/docker/runc@1.3.5
```

Elasticsearch ecosystem collection

Supported applications

- Elasticsearch

> **Note**
>
> Vulnerability evaluation only applies to Elasticsearch version 7.17.0.

Key features

- **Version** – Unpacks the `elasticsearch-<specific.version>.jar` file to extract installation macros inside `META-INF/MANIFEST.MF` files, which contain the Elasticsearch version string.

Supported platforms

- **Linux** – `/etc/elasticsearch/lib/`, `/opt/elasticsearch/lib/`, and `/usr/share/elasticsearch/lib/`
- **macOS** – `/usr/local/var/lib/elasticsearch/lib/`
- **Windows** – `C:\elasticsearch\`, `C:\Program Files (x86)\Elastic\elasticsearch\lib\`, and `C:\Program Files\Elastic\elasticsearch\lib\`

Example `elasticsearch-<specific.version>.jar/META-INF/MANIFEST.MF` file

The following is an example of an `elasticsearch-<specific.version>.jar/META-INF/MANIFEST.MF` file.

```
//truncated

Manifest-Version: 1.0
Module-Origin: git@github.com:elastic/elasticsearch.git
X-Compile-Elasticsearch-Version: 8.19.0-SNAPSHOT
X-Compile-Lucene-Version: 9.12.1
```

```
X-Compile-Elasticsearch-Snapshot: true

//truncated
```

Example PURL

The following is an example package URL for an `elasticsearch-<specific.version>.jar/META-INF/MANIFEST.MF` file.

```
pkg:generic/elastic/elasticsearch@8.19.0-SNAPSHOT
```

Generic AI/ML Models ecosystem collection

Supported applications

- Locally stored AI/ML model files across common frameworks and tools

Key features

- Detects AI/ML model files based on file extension: `.gguf`, `.safetensors`, `.onnx`, `.pt`, `.pth`, and `.h5`.
- Scans common model directories used by popular AI/ML frameworks and local inference tools.
- Detected files are generated as components with the `machine-learning-model` component type.

> **Note**
>
> No PURL is generated for components detected by this collector.

Supported platforms

The Amazon Inspector SBOM Generator scans for model files in common paths across platforms:

Linux and macOS

- `~/.keras/models`
- `~/.cache/lm-studio/models`

- ~/.local/share/nomic.ai/GPT4All
- ~/.jan/models
- ~/.cache/llama.cpp
- ~/.tabby/models
- ~/.local/share/localai/models
- ~/text-generation-webui/models
- ~/ComfyUI/models
- ~/stable-diffusion-webui/models

GitHub Copilot ecosystem collection

Supported applications

- GitHub Copilot CLI
- GitHub Copilot VS Code extension
- GitHub Copilot JetBrains plugin

Key features

- Detects Copilot across three install surfaces: npm-distributed CLI (resolves `package.json` from the binary path via the npm layout), VS Code extension (parses `package.json` in `github.copilot-<version>` directories; excludes the separate `github.copilot-chat-*` extension), and JetBrains plugin (parses `META-INF/plugin.xml` for the `<version>` element).

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux and macOS

- /usr/bin/github-copilot-cli
- /usr/local/bin/github-copilot-cli
- ~/.local/bin/github-copilot-cli
- ~/.vscode/extensions/github.copilot-*/

- `<IDE>/plugins/github-copilot/META-INF/plugin.xml`

Windows

- `%APPDATA%\npm\github-copilot-cli.cmd`
- `%USERPROFILE%\.vscode\extensions\github.copilot-*\`
- `%APPDATA%\JetBrains\<IDE>\plugins\github-copilot\`

Example PURL

The following is an example package URL for GitHub Copilot.

```
Sample PURL: pkg:generic/github/copilot@0.1.36?distro=linux
```

Google ecosystem collection

Supported applications

- Google Chrome
- Puppeteer (supports the puppeteer library; puppeteer-core is not included)

> **Note**
>
> Puppeteer supports the puppeteer library. Puppeteer core is not included.

Supported artifacts

Amazon Inspector collects Google Chrome information from the following:

- The `chrome/VERSION` file (build source)
- The `chrome.exe` file (Windows Chrome installation)
- The `puppeteer` file (installation)

For each of the supported artifacts, the Sbomgen parses and collects either the chrome file or the puppeteer file. For puppeteer installations, the corresponding Chromium version is collected

based on the puppeteer version. For more information, see Supported browsers on the Puppeteer website.

When any of the following environment variables is set to `true`, evaluation is skipped, and the `skip_chromium_download=true` qualifier is added to the Puppeteer package URL.

- `PUPPETEER_SKIP_CHROMIUM_DOWNLOAD`
- `PUPPETEER_CHROME_SKIP_DOWNLOAD`
- `PUPPETEER_SKIP_DOWNLOAD`

Example `chrome/VERSION` version file

The following is an example of the `chrome/VERSION` version file.

```
MAJOR=130
MINOR=0
BUILD=6723
PATCH=58
```

Example PURL

The following is an example package URL for a `chrome/VERSION` version file.

```
Sample PURL: pkg:generic/google/chrome@130.0.6723.58
```

Example `puppeteer` version file

The following is an example of the `puppeteer` version file.

```
{
"name": "puppeteer",
"version": "23.9.0",
"description": "A high-level API to control headless Chrome over the DevTools
 Protocol",
"keywords": [
  "puppeteer",
  "chrome",
  "headless",
  "automation"
```

```
]
}
```

Example PURL

The following is an example package URL for a `puppeteer` version file.

```
Sample PURL: pkg:generic/google/puppeteer@23.9.0
```

Example PURL

The following is an example package URL with skip qualifier for a `puppeteer` version file.

```
pkg:generic/google/puppeteer@22.15.0?distro=linux&skip_chromium_download=true
```

Homebrew ecosystem collection

Supported applications

- Homebrew package manager

Key features

- Detects Homebrew-managed packages by reading each formula's `INSTALL_RECEIPT.json` under the Homebrew `Cellar` directory.
- Scans the default `Cellar` locations (`/usr/local/Cellar`, `/opt/homebrew/Cellar`, and `/home/linuxbrew/.linuxbrew/Cellar`), as well as the location named by the `HOMEBREW_CELLAR` environment variable.
- Generates Homebrew Package URLs with the `brew` type.

Supported platforms

- macOS
- Linux

Example path

The following is an example of a Homebrew package install receipt path.

```
/opt/homebrew/Cellar/<name>/<version>/INSTALL_RECEIPT.json
```

Example PURL

The following is an example package URL for a Homebrew-managed package.

```
pkg:brew/<name>@<version>
```

HuggingFace ecosystem collection

Supported applications

- HuggingFace `hf` CLI

Key features

- Extracts locally cached AI/ML models installed by HuggingFace.
- Generates HuggingFace Package URLs.
- Models downloaded using `hf download --local-dir` are not currently supported.

Example path

The following is an example of a cached HuggingFace model path.

```
/home/ec2-user/.cache/huggingface/hub/models--MiniMaxAI--MiniMax-M2.5/snapshots/<hash>
```

Example PURL

The following is an example package URL for a HuggingFace model. The component type is `machine-learning-model`.

```
pkg:huggingface/MiniMaxAI/MiniMax-M2.5@<hash>
```

Java ecosystem collection

Supported applications

- Oracle JDK

- Oracle JRE
- Amazon Corretto

Key features

- Extracts the string of the Java installation.
- Identifies the directory path that contains the Java runtime.
- Identifies the vendor as Oracle JDK, Oracle JRE, or Amazon Corretto.

The Amazon Inspector SBOM Generator scans for Java installations across the following installation paths and platforms:

- macOS: `/Library/Java/JavaVirtualMachines`
- Linux 32-bit: `/usr/lib/jvm`
- Linux 64-bit: `/usr/lib64/jvm`
- Linux (generic): `/usr/java` and `/opt/java`

Example Java version information

The following are examples of Java release information for Amazon Corretto and Oracle JDK.

```
// Amazon Corretto
IMPLEMENTOR="Amazon.com Inc."
IMPLEMENTOR_VERSION="Corretto-17.0.11.9.1"
JAVA_RUNTIME_VERSION="17.0.11+9-LTS"
JAVA_VERSION="17.0.11"
JAVA_VERSION_DATE="2024-04-16"
LIBC="default"
MODULES="java.base java.compiler java.datatransfer java.xml java.prefs java.desktop
 java.instrument java.logging java.management java.security.sasl java.naming
 java.rmi java.management.rmi java.net.http java.scripting java.security.jgss
 java.transaction.xa java.sql java.sql.rowset java.xml.crypto java.se java.smartcardio
 jdk.accessibility jdk.internal.jvmstat jdk.attach jdk.charsets jdk.compiler
 jdk.crypto.ec jdk.crypto.cryptoki jdk.dynalink jdk.internal.ed jdk.editpad
 jdk.hotspot.agent jdk.httpserver jdk.incubator.foreign jdk.incubator.vector
 jdk.internal.le jdk.internal.opt jdk.internal.vm.ci jdk.internal.vm.compiler
 jdk.internal.vm.compiler.management jdk.jartool jdk.javadoc jdk.jcmd jdk.management
```

```
 jdk.management.agent jdk.jconsole jdk.jdeps jdk.jdwp.agent jdk.jdi jdk.jfr jdk.jlink
 jdk.jpackage jdk.jshell jdk.jsobject jdk.jstatd jdk.localedata jdk.management.jfr
 jdk.naming.dns jdk.naming.rmi jdk.net jdk.nio.mapmode jdk.random jdk.sctp
 jdk.security.auth jdk.security.jgss jdk.unsupported jdk.unsupported.desktop
 jdk.xml.dom jdk.zipfs"
OS_ARCH="x86_64"
OS_NAME="Darwin"
SOURCE=".:git:7917f11551e8+"

// JDK
IMPLEMENTOR="Oracle Corporation"
JAVA_VERSION="19"
JAVA_VERSION_DATE="2022-09-20"
LIBC="default"
MODULES="java.base java.compiler java.datatransfer java.xml java.prefs java.desktop
 java.instrument java.logging java.management java.security.sasl java.naming
 java.rmi java.management.rmi java.net.http java.scripting java.security.jgss
 java.transaction.xa java.sql java.sql.rowset java.xml.crypto java.se java.smartcardio
 jdk.accessibility jdk.internal.jvmstat jdk.attach jdk.charsets jdk.zipfs jdk.compiler
 jdk.crypto.ec jdk.crypto.cryptoki jdk.dynalink jdk.internal.ed jdk.editpad
 jdk.hotspot.agent jdk.httpserver jdk.incubator.concurrent jdk.incubator.vector
 jdk.internal.le jdk.internal.opt jdk.internal.vm.ci jdk.internal.vm.compiler
 jdk.internal.vm.compiler.management jdk.jartool jdk.javadoc jdk.jcmd jdk.management
 jdk.management.agent jdk.jconsole jdk.jdeps jdk.jdwp.agent jdk.jdi jdk.jfr jdk.jlink
 jdk.jpackage jdk.jshell jdk.jsobject jdk.jstatd jdk.localedata jdk.management.jfr
 jdk.naming.dns jdk.naming.rmi jdk.net jdk.nio.mapmode jdk.random jdk.sctp
 jdk.security.auth jdk.security.jgss jdk.unsupported jdk.unsupported.desktop
 jdk.xml.dom"
OS_ARCH="x86_64"
OS_NAME="Darwin"
SOURCE=".:git:53b4a11304b0 open:git:967a28c3d85f"
```

Example PURL

The following are example package URLs for Java releases.

```
Sample PURL:
# Amazon Corretto
pkg:generic/amazon/amazon-corretto@21.0.3
# Oracle JDK
pkg:generic/oracle/jdk@11.0.16
# Oracle JRE
```

```
pkg:generic/oracle/jre@20
```

Jenkins ecosystem collection

Supported applications

- Jenkins Core

> **Note**
>
> Vulnerability evaluation applies to Jenkins version 2.400.* and higher.

Key features

- Extracts version information from the `jenkins.war` file by reading the `META-INF/MANIFEST.MF` file, which contains the Jenkins version string.

The Amazon Inspector SBOM Generator looks for Jenkins installations in common installation paths across platforms:

Linux

- `/usr/share/jenkins/jenkins.war`
- `/usr/share/java/jenkins.war`

macOS

- `/opt/homebrew/opt/jenkins-lts/libexec/jenkins.war`

Windows

- `C:\Program Files\Jenkins\Jenkins.war`
- `C:\Program Files (x86)\Jenkins\Jenkins.war`

Example files

The following are examples of `jenkins.war/META-INF/MANIFEST.MF` files for different releases.

```
Manifest-Version: 1.0
Created-By: Maven WAR Plugin 3.4.0
Build-Jdk-Spec: 21
Implementation-Title: Jenkins war
Main-Class: executable.Main
Implementation-Version: 2.516.2
Jenkins-Version: 2.516.2
```

```
Manifest-Version: 1.0
Jenkins-Version: 2.414.1
Implementation-Title: Jenkins
Implementation-Version: 2.414.1
Built-By: kohsuke
Created-By: Apache Maven 3.8.6
```

Sample PURLs

The following are package URLs for version 2.516.2 of the Jenkins LTS release and version 2.414 of the Jenkins automation server release.

```
LTS: pkg:generic/jenkins/jenkins-core-lts@2.516.2
Regular: pkg:generic/jenkins/jenkins-core@2.414
```

Kiro CLI ecosystem collection

Supported applications

- Kiro CLI (Amazon AI coding assistant)

Key features

- Examines the `kiro-cli` binary (or `kiro-cli.appimage`) to extract embedded version information.
- Looks for version strings in the binary executable `.rodata` section (for ELF binaries on Linux) or `__cstring` section (for Mach-O binaries on macOS). Version is extracted from the AWS tooling user agent string pattern `Version<X.Y.Z>/`.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `~/.local/bin/kiro-cli`
- `/usr/bin/kiro-cli`
- `~/Applications/kiro-cli.appimage`

macOS

- `/Applications/Kiro CLI.app/Contents/MacOS/kiro-cli`
- `~/.local/bin/kiro-cli`

Example PURL

The following is an example package URL for Kiro CLI.

```
Sample PURL: pkg:generic/amazon/kiro@1.29.5?distro=linux
```

MariaDB and MySQL ecosystem collection

MariaDB

Supported applications

- MariaDB Server (10.6+, 11.x, 12.x)

Key features

- Extracts version information from database server binaries and header files using database-specific patterns.
- Identifies the directory path containing the database server installation.
- Automatically distinguishes between MariaDB and MySQL installations using data-driven file type detection.

The Amazon Inspector SBOM Generator looks for the MariaDB installation in common installation paths across platforms:

Linux

- `/usr/bin/mariadbd`
- `/usr/sbin/mariadbd`
- `/usr/local/bin/mariadbd`

macOS

- `/opt/homebrew/Cellar/mariadb/include/mysql/mariadb_version.h` (MariaDB)

Windows

- `C:\Program Files (x86)\MariaDB\include\mysql\mariadb_version.h` (MariaDB)
- `C:\Program Files\MariaDB\include\mysql\mariadb_version.h` (MariaDB)

Example PURL

The following is an example package URL for a MariaDB server.

```
# MariaDB Server

pkg:generic/mysql/mariadb-server@10.11.8
```

MySQL

Supported applications

- Oracle MySQL Server (8.0, 8.4, 9.4+)
- MySQL NDB Cluster

Key features

- Extracts version information from database server binaries and header files using database-specific patterns.

- Identifies the directory path containing the database server installation.
- Automatically distinguishes between MySQL and MariaDB installations using data-driven file type detection.
- Identifies MySQL NDB Cluster builds of the server and reports them as `mysql-cluster` with the NDB Cluster version.

The Amazon Inspector SBOM Generator looks for the MySQL installation in common installation paths across platforms:

Linux

- `/usr/local/bin/mysqld`
- `/usr/bin/mysqld`
- `/usr/sbin/mysqld`

macOS

- `/usr/local/mysql/include/mysql_version.h` (MySQL)

Windows

- `C:\Program Files\MySQL\MySQL Server\include\mysql_version.h` (MySQL)
- `C:\Program Files (x86)\MySQL\MySQL Server\include\mysql_version.h` (MySQL)
- `C:\mysql-cluster-*-winx64\bin\mysqld.exe` (MySQL NDB Cluster)

Example PURL

The following are example package URLs for a MySQL server and a MySQL NDB Cluster server.

```
# Oracle MySQL Server

pkg:generic/mysql/mysql-server@8.0.43

# MySQL NDB Cluster

pkg:generic/mysql/mysql-cluster@7.6.25
```

Microsoft applications ecosystem collection

The following Microsoft applications are inventoried by the Amazon Inspector SBOM Generator. Due to limitations in the Microsoft CVRF API, detections in the InspectorScan API are only supported for versions of these applications released in 2021 or later. Findings are mapped to Microsoft KBs or CVEs (where applicable).

Supported Microsoft applications (2021+)

- PowerShell
- NuGet CLI
- Visual Studio Code
- Microsoft Edge
- SharePoint Server
- Microsoft Defender
- Exchange Server
- Visual Studio
- .NET Core Runtime
- .NET Framework
- ASP.NET Core Runtime
- Microsoft Teams
- Outlook for Windows
- Microsoft Office
- Microsoft 365

Key features

- PowerShell – Examines the `pwsh.exe` file to extract the embedded version information.
- NuGet CLI – Examines the `nuget.exe` file to extract the embedded version information.
- Visual Studio Code – Examines the `Code.exe` file to extract the embedded version information.
- Microsoft Edge – Examines the `msedge.exe` file to extract the embedded version information.
- SharePoint Server – Examines the `Microsoft.SharePoint.dll` file to extract the embedded version information.

- Microsoft Defender – Examines the `MsMpEng.exe` file to extract the embedded version information.
- Exchange Server – Examines the `Exsetup.exe` file to extract the embedded version information.
- Visual Studio – Parses the `state.json` file to retrieve the version string from the `catalogInfo.productDisplayVersion` field.
- .NET Core Runtime – Searches for the `Microsoft.NETCore.App.deps.json` file in installation paths and extracts the version string from the following file path pattern.

```
Microsoft.NETCore.App/<VERSION>/Microsoft.NETCore.App.deps.json
```

- .NET Framework – Parses the Windows Registry and reads file metadata to detect installed .NET Framework versions. The scanner checks the following registry keys, registry values, and files.
  - **Registry Key** (<VERSION_SUB_KEY> represents the .NET Framework version, such as v2.0.50727, v3.5, or v4\Full)

    ```
    HKLM\SOFTWARE\Microsoft\NET Framework Setup\NDP\<VERSION_SUB_KEY>
    ```

    ```
    HKLM\SOFTWARE\Wow6432Node\Microsoft\NET Framework Setup\NDP\<VERSION_SUB_KEY>
    ```

  - **Registry Value**
    - Install – Indicates whether the .NET Framework version is installed.
    - Version – Installed .NET Framework version (version 4.0 or lower)
    - Release – A REG_DWORD value that maps to the installed .NET Framework version (version 4.5 or later)
  - **DLL Files**

    The scanner extracts the file version from `mscorlib.dll` and `System.dll`. If these files exist, they are added to the SBOM as nested file components. For .NET Framework version 4.5 or later, the largest file version among these files is reported as the version.
- ASP.NET Core Runtime – Searches for the `Microsoft.AspNetCore.App.deps.json` file in installation paths and extracts the version string from the following file path pattern.

```
Microsoft.AspNetCore.App/<VERSION>/Microsoft.AspNetCore.App.deps.json
```

- Outlook for Windows – Parses the Windows Registry and extracts the version from the following registry key.

```
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft
\Windows\CurrentVersion\AppModel\PackageRepository\Packages
\Microsoft.OutlookForWindows_<VERSION>_<ARCH>__8wekyb3d8bbwe
```

- Microsoft Teams – Parses the Windows Registry and extracts the version from the following registry key.

```
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion
\AppModel\PackageRepository\Packages\MSTeams_<VERSION>_<ARCH>__8wekyb3d8bbwe
```

- Microsoft Office 365 / Microsoft 365 – Parses the Windows Registry and extracts the version from the following registry key and value.
  - Registry Key

    ```
    HKLM\SOFTWARE\Microsoft\Office\ClickToRun\Configuration
    ```

  - Registry Value
    - VersionToReport – Microsoft Office version
    - ProductReleaseIds – List of product IDs. This is used to identify installed Office products. For more information about product IDs, see product IDs on the Microsoft website.
- Microsoft Office Suite – Collects each installed Office application by examining the following executable files:
  - `EXCEL.EXE` – Microsoft Excel
  - `WINWORD.EXE` – Microsoft Word
  - `POWERPNT.EXE` – Microsoft PowerPoint
  - `OUTLOOK.EXE` – Microsoft Outlook

  The version number in the Windows Registry is used as the authoritative version number for each installed Office application.

Example `state.json` file

The following is an example of a `state.json` file that is used to collect the installed Visual Studio version.

```
{
    "icon": {
        "mimeType": "image/svg+xml",
        "fileName": "product.svg"
    },
    "updateDate": "2025-11-06T05:05:35.6517471Z",
    "installDate": "2025-11-06T05:05:35.6527436Z",
    "enginePath": "C:\\Program Files (x86)\\Microsoft Visual Studio\\Installer\
\resources\\app\\ServiceHub\\Services\\Microsoft.VisualStudio.Setup.Service",
    "installationName": "VisualStudio/17.14.19+36623.8",
    "catalogInfo": {
        "id": "VisualStudio/17.14.19+36623.8",
        "buildBranch": "d17.14",
        "buildVersion": "17.14.36623.8",
        "localBuild": "build-lab",
        "manifestName": "VisualStudio",
        "manifestType": "installer",
        "productDisplayVersion": "17.14.19",
// truncated
```

Example PURL

The following are example package URLs for each Microsoft application.

```
// PowerShell
Sample PURL: pkg:generic/microsoft/powershell@7.5.3

// NuGet CLI
Sample PURL: pkg:generic/microsoft/nuget@6.14.0

// Visual Studio Code
Sample PURL: pkg:generic/microsoft/visualstudiocode@1.104.2

// Microsoft Edge
Sample PURL: pkg:generic/microsoft/edge@140.0.3485.94

// SharePoint Server
Sample PURL: pkg:generic/microsoft/sharepoint@23.38.219.1

// Microsoft Defender
Sample PURL: pkg:generic/microsoft/defender@4.18.23110.3

// Exchange Server
```

```
Sample PURL: pkg:generic/microsoft/exchangeserver@15.2.2562.17

// Visual Studio
Sample PURL: pkg:generic/microsoft/visualstudio@17.14.19

// .NET Core Runtime
Sample PURL: pkg:generic/microsoft/dotnet@8.0.18

// .NET Framework
Sample PURL: pkg:generic/microsoft/dotnet-framework-v4.8.1@4.8.9320.0

// ASP.NET Core Runtime
Sample PURL: pkg:generic/microsoft/aspdotnet@8.0.18

// Microsoft Teams
Sample PURL: pkg:generic/microsoft/teams@25241.203.3947.4411

// Outlook for Windows
Sample PURL: pkg:generic/microsoft/outlookforwindows@1.2025.916.400

// Microsoft 365 / Office 365
Sample PURL: pkg:generic/microsoft/office@16.0.19127.20264?
product_ids=O365HomePremRetail

// Microsoft Word
Sample PURL: pkg:generic/microsoft/word@16.0.19127.20264

// Microsoft Excel
Sample PURL: pkg:generic/microsoft/excel@16.0.19127.20264

// Microsoft PowerPoint
Sample PURL: pkg:generic/microsoft/powerpoint@16.0.19127.20264

// Microsoft Outlook
Sample PURL: pkg:generic/microsoft/outlook@16.0.19127.20264
```

Microsoft SQL Server ecosystem collection

Supported applications

- Microsoft SQL Server

Key features

- Reads from the Windows registry to discover installed Microsoft SQL Server instances and extract version information.
- Discovers instances through a two-step process: reads the `InstalledInstances` value, resolves each instance path from the `Instance Names\SQL` subkey, then reads setup information from each instance's `Setup` subkey.
- Collects instance name, base version, patch level, edition, service pack (if present), and the registry key path.
- The component version and PURL use the patch level (full build number).

Supported platforms – Windows

The Amazon Inspector SBOM Generator reads from the following Windows registry key to discover installed instances:

```
HKLM\SOFTWARE\Microsoft\Microsoft SQL Server
```

The scanner reads the `InstalledInstances` value to enumerate instances, resolves each instance path from the `Instance Names\SQL` subkey, then reads setup information from each instance's `Setup` subkey.

Example PURL

The following is an example package URL for a Microsoft SQL Server instance.

```
pkg:generic/microsoft/sqlserver@16.0.1000.6
```

MongoDB ecosystem collection

Supported applications

- MongoDB Server (7.0+, 8.0+)

Key features

- Examines mongod binaries to extract embedded version information.

> **Note**
>
> The mongod binary can exceed 200 MB in size. To scan for MongoDB, the Amazon Inspector SBOM Generator file size limit must be configured to allow files over 200 MB.

The Amazon Inspector SBOM Generator looks for MongoDB installations in common installation paths across platforms:

Linux

- `/usr/bin/mongod`
- `/usr/local/bin/mongod`

macOS

- `/usr/local/bin/mongod`
- `/opt/homebrew/bin/mongod`

Windows

- `C:\Program Files\MongoDB\Server\bin\mongod.exe`

Example PURL

The following is an example package URL for MongoDB Server.

```
pkg:generic/mongodb/mongodb-server@8.2.4?platform=linux
```

Mozilla ecosystem collection

Supported applications

- Firefox
- Firefox ESR
- Thunderbird

Key features

- Parses the `application.ini` manifest that ships with Mozilla applications. The parser reads only the `[App]` section and stops at the next section header, so keys from other sections, such as `[Gecko]`, are ignored.
- Identifies the product from the `RemotingName` field, which resolves to `firefox`, `firefox-esr`, or `thunderbird`. `RemotingName` is used instead of `Name` because the `Name` field is `Firefox` for both the release and ESR channels.
- Extracts the version verbatim from the `Version` field. Both `RemotingName` (matching a known product) and `Version` are required. If either is missing, or the `RemotingName` value is not a known Mozilla product, no component is reported. This excludes non-Mozilla Gecko applications that also ship an `application.ini` file.
- Detected applications are reported as CycloneDX components with the `application` type.
- Records the `SourceRepository` value as the CycloneDX property `amazon:inspector:sbom_generator:mozilla:source_repository` rather than as a package URL qualifier, so it doesn't affect vulnerability identification.

Supported platforms

The Amazon Inspector SBOM Generator discovers `application.ini` files by name with the following default paths across platforms:

Linux

- `/usr/lib/firefox/`, `/usr/lib/firefox-esr/`, `/usr/lib/thunderbird/`
- `/usr/lib64/firefox/`, `/usr/lib64/thunderbird/`
- `/opt/firefox/`, `/opt/thunderbird/`
- `/usr/local/lib/firefox/`, `/usr/local/lib/thunderbird/`

macOS

- `/Applications/Firefox.app/Contents/Resources/`
- `/Applications/Thunderbird.app/Contents/Resources/`

Windows

- `C:\Program Files\Mozilla Firefox\`, `C:\Program Files (x86)\Mozilla Firefox\`
- `C:\Program Files\Mozilla Thunderbird\`, `C:\Program Files (x86)\Mozilla Thunderbird\`

Limitations

- Installations in non-default paths aren't on the default localhost scan list. To scan them, use the `--path` option to run a directory scan.
- Snap and Flatpak packages of Firefox and Thunderbird aren't supported.
- Installations from an OS package manager are detected if they contain an `application.ini` file. These installations are also reported by the operating system package collectors.

Example `application.ini` file

The following is an example of content inside an `application.ini` file for a Firefox ESR installation.

```
//truncated

[App]
Vendor=Mozilla
Name=Firefox
RemotingName=firefox-esr
Version=140.14.0
SourceRepository=https://hg.mozilla.org/releases/mozilla-esr140

//truncated
```

The product `firefox-esr` is taken from the `RemotingName` field, and version `140.14.0` is taken from the `Version` field.

Example PURL

The following are example package URLs for Mozilla applications.

```
// Firefox
```

```
pkg:generic/mozilla/firefox@152.0.6?distro=darwin

// Firefox ESR
pkg:generic/mozilla/firefox-esr@140.14.0?distro=linux

// Thunderbird
pkg:generic/mozilla/thunderbird@153.0.3?distro=darwin
```

Nginx ecosystem collection

Supported applications

- Nginx

Supported platforms

The following are supported platforms.

Linux

- `/usr/sbin/nginx`
- `/usr/local/nginx`
- `/usr/local/etc/nginx`
- `/usr/local/nginx/nginx`
- `/usr/local/nginx/sbin/nginx`
- `/etc/nginx/nginx`

Windows

- `C:\nginx\nginx.exe`
- `C:\nginx-x.y.z\nginx.exe` (where `x.y.z` is an arbitrary version)

macOS

- `/usr/local/etc/nginx/nginx`

Key features

This collection examines binaries to extract embedded version information. It searches for version strings in the binary executable `.rodata` section (for ELF binaries on Linux), `.rdata` section (for PE binaries on Windows), or `__cstring` section (for Mach-O binaries on macOS).

Example version string

The following is an example of a version string embedded in an Nginx binary.

```
nginx version: nginx/1.27.5
```

Version `1.27.5` is extracted to identify the Nginx version.

Example PURL

The following is an example package URL for Nginx.

```
Sample PURL: pkg:generic/nginx/nginx@1.27.5
```

Node.js runtime collection

Supported applications

- node runtime binary for Node.js

Supported platforms

The following are supported platforms. In these paths, * represents an arbitrary version.

Linux

- `/usr/local/bin/node`
- `/usr/bin/node`
- `/nodejs/bin/node`
- `~/.nvm/versions/node/*/bin/node`
- `~/.local/share/fnm/node-versions/*/installation/bin/node`
- `~/.asdf/installs/nodejs/*/bin/node`
- `~/.local/share/mise/installs/node/*/bin/node`

- `~/.volta/tools/image/node/*/bin/node`

Windows

- `C:\Program Files\nodejs\node.exe`
- `C:\Program Files (x86)\nodejs\node.exe`
- `%APPDATA%\fnm\node-versions\*\installation\node.exe`

macOS

- `/opt/homebrew/Cellar/node/*/bin/node`

Key features

This collection examines binaries to extract embedded version information. It searches for version strings in the binary executable `.rodata` section (for ELF binaries on Linux), `.rdata` section (for PE binaries on Windows), or `__cstring` section (for Mach-O binaries on macOS).

Example version string

The following is an example of a version string embedded in a Node.js runtime binary.

```
node.js/v24.11.1
```

Version `24.11.1` is extracted to identify the Node.js runtime version.

Example PURL

The following is an example package URL for Node.js.

```
Sample PURL: pkg:generic/nodejs/node@24.11.1
```

Ollama ecosystem collection

Supported applications

- Ollama (local LLM runtime)

Key features

- Examines the `ollama` binary (or `ollama.exe` on Windows) to extract embedded version information.
- Ollama is a Go binary with the version injected at build time via `-ldflags -X`, which appears in the binary's data section as a tilde-prefixed string (for example, `~0.21.0`). On Homebrew-built macOS binaries, the version may alternatively be present in the Go module build info as `ollama \tv<version>`.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `/usr/local/bin/ollama`
- `/usr/bin/ollama`
- `~/.local/bin/ollama`

macOS

- `/Applications/Ollama.app/Contents/MacOS/ollama`
- `/usr/local/bin/ollama`
- `/opt/homebrew/bin/ollama`

Windows

- `%LOCALAPPDATA%\Programs\Ollama\ollama.exe`
- `%ProgramFiles%\Ollama\ollama.exe`

Example version string

The following is an example of a version string embedded in an Ollama binary.

```
~0.21.0
```

Version `0.21.0` is extracted to identify the Ollama version.

Example PURL

The following is an example package URL for Ollama.

```
Sample PURL: pkg:generic/ollama/ollama@0.21.0?distro=linux
```

Ollama Model Collector ecosystem collection

Supported applications

- Ollama CLI

Key features

- Extracts locally cached AI/ML models installed by Ollama.
- Generates Ollama Package URLs.

Example path

The following is an example of a cached Ollama model path.

```
/usr/share/ollama/.ollama/models/manifests/registry.ollama.ai/library/gemma4/latest
```

Example PURL

The following is an example package URL for an Ollama model. The component type is `machine-learning-model`.

```
pkg:ollama/gemma4@<hash>
```

OpenSSH ecosystem collection

Supported applications

- OpenSSH (version 9)
- OpenSSH (version 10)

Supported platforms – Linux and macOS

- `/usr/sbin/sshd`
- `/usr/local/sbin/sshd`

Supported platforms – Windows

- `C:\Windows\System32\OpenSSH\sshd.exe`
- `C:\Program Files\OpenSSH\sshd.exe`
- `C:\Program Files (x86)\OpenSSH\sshd.exe`
- `C:\OpenSSH\sshd.exe`

Key features

- Examines `sshd` binaries to extract embedded version information.
- Looks for version strings in the binary executable `.rodata` section (for ELF binaries on Linux), `__cstring` section (for Mach-O binaries on macOS), or `.rdata` section (for PE binaries on Windows).

Example version string

The following is an example of a version string embedded in an OpenSSH binary.

```
OpenSSH_9.9p2
```

Version `9.9p2` is extracted to identify the OpenSSH version.

Example PURL

The following is an example package URL for OpenSSH.

```
Sample PURL: pkg:generic/openssh/openssh@9.9p2
```

OpenSSL ecosystem collection

Supported applications

Support for OpenSSL libraries and development packages is limited to software built with official OpenSSL 3.0.0 releases and later. The software also must follow semantic versioning. Custom or forked OpenSSL variants and versions lower than 3.0.0 are not supported.

The Amazon Inspector SBOM Generator extracts key package information for each installed OpenSSL instance.

Key features

- Extracts the base semantic version (SemVer) string from the OpenSSL header file.
- Identifies the directory path containing the OpenSSL installation.

The Amazon Inspector SBOM Generator looks for OpenSSL installations by scanning for the `openssl.h` file in common installation paths across platforms.

Example installation paths for Linux and Unix

The following are example installation paths for Linux and Unix.

```
/usr/local/include/openssl/opensslv.h
/usr/local/ssl/include/openssl/opensslv.h
/usr/local/openssl/include/openssl/opensslv.h
/usr/local/opt/openssl/include/openssl/opensslv.h
/usr/include/openssl/opensslv.h
```

The Amazon Inspector SBOM Generator extracts version information by parsing the `openssl.h` file and looking for the version definitions.

```
# define OPENSSL_VERSION_MAJOR  3
# define OPENSSL_VERSION_MINOR  4
# define OPENSSL_VERSION_PATCH  0
```

Example PURL

The following is an example package URL for the OpenSSL version.

```
Sample PURL: pkg:generic/openssl/openssl@3.4.0
```

Oracle Database Server collection

Supported applications

- Oracle Database

Supported platforms – Linux

- `/opt/oracle`
- `/u01/app/oracle`

> **Note**
>
> Vulnerability evaluation applies only to Oracle Database Server version 19 and higher.

Key features

- Examines Oracle binaries to extract embedded version information.
- Looks for version strings in the binary executable `.rodata` section (for ELF binaries on Linux).
- Version information follows a specific format that includes the RDBMS version string.

Example version string

The following is an example of a version string embedded in an Oracle Database binary:

```
RDBMS_23.7.0.25.01DBRU_LINUX.X64_240304
```

Version `23.7.0.25.01` is extracted to identify the Oracle Database version.

Example PURL

The following is an example package URL for Oracle Database.

```
Sample PURL: pkg:generic/oracle/database@23.7.0.25.01
```

PHP ecosystem collection

Supported applications

- PHP (version 8.1 and higher)

Key features

- Extracts version information from PHP binary executables using embedded version strings.
- Identifies the directory path containing the PHP binary.
- Automatically detects both standard PHP binaries and versioned installations, such as `php8.1`, `php8.2`, and `php8.3`.

The Amazon Inspector SBOM Generator looks for PHP installations in common installation paths across platforms:

Linux

- `/usr/bin/php8.1` through `/usr/bin/php8.9`
- `/usr/sbin/php8.1` through `/usr/sbin/php8.9`
- `/usr/local/bin/php`, `/usr/bin/php`, `/usr/sbin/php`
- `/usr/local/bin/php8.1` through `/usr/local/bin/php8.9` (versioned binaries)

macOS

- `/opt/homebrew/bin/php`
- `/usr/bin/php`
- `/usr/local/bin/php`

Windows

- `C:\php\php.exe`
- `C:\php8.1\php.exe` through `C:\php8.9\php.exe` (versioned directories)

Example PHP version extraction

The Amazon Inspector SBOM Generator extracts version information from PHP binaries by searching for embedded version strings using the following pattern.

```
X-Powered-By: PHP/8.4.12
```

Version `8.4.12` is extracted from this pattern to identify the PHP version.

Example PURL

The following is an example package URL for PHP.

```
pkg:generic/php/php@8.4.12
```

Redis ecosystem collection

Supported applications

- Redis (version 7.2 and higher)

Key features

- Extracts version information from Redis `redis-server` binary executables using embedded version strings.
- Searches for version strings in the binary executable `.rodata` section (for ELF binaries on Linux) or `__cstring` section (for Mach-O binaries on macOS).

The Amazon Inspector SBOM Generator looks for Redis installations in common installation paths across platforms:

Linux

- `/usr/bin/redis-server`
- `/usr/local/bin/redis-server`

macOS

- `/opt/homebrew/bin/redis-server`

- `/usr/local/bin/redis-server`

Example version string

The following is an example of a version string embedded in a Redis binary.

```
redis-7.2.6
```

Version `7.2.6` is extracted to identify the Redis version.

Example PURL

The following is an example package URL for Redis.

```
pkg:generic/redis/redis@7.2.6
```

Standalone AI applications ecosystem collection

The Amazon Inspector SBOM Generator detects standalone AI desktop and local LLM applications that are installed outside a package manager. Detected applications are reported as CycloneDX components with the `application` type and a `generic` Package URL type. The version-extraction method differs per application, as described in the following table.

Application	Scanner name	Version source	Example PURL
Tabby (TabbyML)	`tabby-installation`	Binary constant section (OTEL `service.version`)	`pkg:generic/tabbyml/tabby@<version>`
Claude Desktop	`claude-desktop-installation`	Windows PE `ProductVersion` (`StringFileInfo`)	`pkg:generic/anthropic/claude-desktop@<version>`
LM Studio	`lm-studio-installation`	Electron `resources/app/package.json`	`pkg:generic/lmstudio/lm-studio@<version>`

Application	Scanner name	Version source	Example PURL
Jan	`jan-installation`	Binary anchor `<version> jan.ai.app` (Tauri)	`pkg:generic/janhq/jan@<version>`
GPT4All	`gpt4all-installation`	Binary anchor: the application-name string `GPT4All` followed by a null byte and `<version>` (Qt)	`pkg:generic/nomic-ai/gpt4all@<version>`

Supported platforms

Windows and Linux (application-only scope).

> **Note**
>
> Installations in non-default install prefixes require the `--path` argument. A build that lacks the expected version anchor yields no component.

Windsurf ecosystem collection

Supported applications

- Windsurf (by Codeium)

Key features

- Examines installations of Windsurf, an Electron-based AI code editor by Codeium. Extracts the version from the application's `resources/app/package.json` Electron manifest.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `/usr/share/windsurf/`
- `/usr/bin/windsurf`

Windows

- `%LOCALAPPDATA%\Programs\windsurf\`
- `<SystemDrive>\Program Files\Windsurf\`

Example PURL

The following is an example package URL for Windsurf.

```
Sample PURL: pkg:generic/codeium/windsurf@1.10.5?distro=linux
```

WordPress ecosystem collection

Supported components

- WordPress core
- WordPress plugins
- WordPress themes

Key features

- WordPress core – Parses the `/wp-includes/version.php` file to extract the version value from the `$wp_version` variable.
- WordPress plugins – Parses the `/wp-content/plugins/<WordPress Plugin>/readme.txt` file or `/wp-content/plugins/<WordPress Plugin>/readme.md` file to extract the `Stable tag` value as the version string.
- WordPress themes – Parses the `/wp-content/themes/<WordPress Theme>/style.css` file to extract the version from the version metadata.

Example `version.php` file

The following is an example of a WordPress core `version.php` file.

```
// truncated

/**
* The WordPress version string.
*
* Holds the current version number for WordPress core. Used to bust caches
* and to enable development mode for scripts when running from the /src directory.
*
* @global string $wp_version
*/
$wp_version = '6.5.5';

// truncated
```

Example PURL

The following is an example package URL for WordPress core.

```
Sample PURL: pkg:generic/wordpress/core/wordpress@6.5.5
```

Example `readme.txt` file

The following is an example of a WordPress plugin `readme.txt` file.

```
=== Plugin Name ===
Contributors: (this should be a list of wordpress.org userid's)
Donate link: https://example.com/
Tags: tag1, tag2
Requires at least: 4.7
Tested up to: 5.4
Stable tag: 4.3
Requires PHP: 7.0
License: GPLv2 or later
License URI: https://www.gnu.org/licenses/gpl-2.0.html
```

```
// truncated
```

Example PURL

The following is an example package URL for a WordPress plugin.

```
Sample PURL: pkg:generic/wordpress/plugin/exclusive-addons-for-elementor@1.0.0
```

Example `style.css` file

The following is an example of a WordPress theme `style.css` file.

```
/*
Author: the WordPress team
Author URI: https://wordpress.org
Description: Twenty Twenty-Four is designed to be flexible, versatile and applicable
 to any website. Its collection of templates and patterns tailor to different needs,
 such as presenting a business, blogging and writing or showcasing work. A multitude
 of possibilities open up with just a few adjustments to color and typography. Twenty
 Twenty-Four comes with style variations and full page designs to help speed up the
 site building process, is fully compatible with the site editor, and takes advantage
 of new design tools introduced in WordPress 6.4.
Requires at least: 6.4
Tested up to: 6.5
Requires PHP: 7.0
Version: 1.2
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Text Domain: twentytwentyfour
Tags: one-column, custom-colors, custom-menu, custom-logo, editor-style, featured-
images, full-site-editing, block-patterns, rtl-language-support, sticky-post,
 threaded-comments, translation-ready, wide-blocks, block-styles, style-variations,
 accessibility-ready, blog, portfolio, news
*/
```

Example PURL

The following is an example package URL for a WordPress theme.

```
Sample PURL: pkg:generic/wordpress/theme/avada@1.0.0
```

Zed ecosystem collection

Supported applications

- Zed (by Zed Industries)

Key features

- Examines the standalone Zed binary distributed by Zed Industries.
- Zed is a Rust binary with no manifest. The version is extracted from a constant string embedded in the binary.

Supported platforms

The Amazon Inspector SBOM Generator scans for installations in common installation paths across platforms:

Linux

- `/usr/bin/zed`
- `/usr/local/bin/zed`
- `~/.local/bin/zed`

Windows

- `%LOCALAPPDATA%\Zed\`
- `<SystemDrive>\Program Files\Zed\`

Example PURL

The following is an example package URL for Zed.

```
Sample PURL: pkg:generic/zed-industries/zed@1.2.6?distro=linux
```

Amazon Inspector SBOM Generator SSL/TLS certificate scans

This section describes how to use the Amazon Inspector SBOM Generator to inventory SSL/TLS certificates. The Sbomgen inventories SSL/TLS certificates by searching for certificates in predefined locations as well as directories provided by the user. The feature is intended to enable users to inventory SSL/TLS certificates as well as identify expired certificates. CA certificates will also appear in the output inventory.

Using Sbomgen certificate scans

You can enable SSL/TLS certificate inventory collection using the `--scanners certificates` argument. Certificate scans can be combined with any of the other scanners. By default, certificate scans are not enabled.

The Sbomgen searches different locations for certificates depending on the artifact being scanned. In all cases, the Sbomgen attempts to extract certificates in files with the following extensions.

```
.pem
.crt
.der
.p7b
.p7m
.p7s
.p12
.pfx
```

The localhost artifact type

If the certificate scanner is enabled and the artifact type is localhost, the Sbomgen recursively looks for certificates in `/etc/*/ssl`, `/opt/*/ssl/certs`, `/usr/local/*/ssl`, and `/var/lib/*/certs`, where `*` is not empty. User-provided directories will be searched recursively, regardless of what directories are named. Typically, CA/system certificates are not placed in these paths. These certificates are often in folders named `pki`, `ca-certs`, or `CA`. They also may appear in the default localhost scan paths.

Directory and container artifacts

When scanning directory or container artifacts, the Sbomgen searches for certificates located anywhere on the artifact.

Example certificate scan commands

The following contains example certificate scan commands. One generates an SBOM that only contains certificates in a local directory. Another generates an SBOM that contains certificates and Alpine, Debian, and Rhel packages in a local directory. Another generates an SBOM that contains certificates found in common certificate locations.

```
# generate SBOM only containing certificates in a local directory
./inspector-sbomgen directory --path ./project/ --scanners certificates

# generate SBOM only containing certificates and Alpine, Debian, and Rhel OS packages
 in a local directory
./inspector-sbomgen directory --path ./project/ --scanners certificates,dpkg,alpine-
apk,rhel-rpm

# generate SBOM only containing certificates, taken from common localhost certificate
 locations
./inspector-sbomgen localhost --scanners certificates
```

Example file component

The following contains two example certificate finding components. When a certificate expires, you can view an extra property that identifies the expiration date.

```
{
      "bom-ref": "comp-2",
      "type": "file",
      "name": "certificate:expired.pem",
      "properties": [
            {
                "name": "amazon:inspector:sbom_generator:certificate_finding:IN-
CERTIFICATE-001",
                "value": "expired:2015-06-06T11:59:59Z"
            },
            {
                "name": "amazon:inspector:sbom_generator:source_path",
                "value": "/etc/ssl/expired.pem"
            }
      ]
},
{
      "bom-ref": "comp-3",
      "type": "file",
      "name": "certificate:unexpired.pem",
```

```
      "properties": [
            {
                "name": "amazon:inspector:sbom_generator:source_path",
                "value": "/etc/ssl/unexpired.pem"
            }
      ]
}
```

Example vulnerability response component

Running the Amazon Inspector SBOM Generator with the `--scan-sbom` flag sends the resulting SBOM to Amazon Inspector for vulnerability scanning. The following is an example of a certificate finding for a vulnerability response component.

```
{
    "advisories": [
        {
            "url": "https://aws.amazon.com/inspector/"
        },
        {
            "url": "https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/
sec_protect_data_transit_encrypt.html"
        }
    ],
    "affects": [
        {
            "ref": "comp-2"
        }
    ],
    "analysis": {
        "state": "in_triage"
    },
    "bom-ref": "vuln-1",
    "created": "2025-04-17T18:48:20Z",
    "cwes": [
        324,
        298
    ],
    "description": "Expired Certificate: The associated certificate(s) are no longer
 valid. Replace certificate in order to reduce risk.",
    "id": "IN-CERTIFICATE-001",
    "properties": [
        {
```

```
            "name": "amazon:inspector:sbom_scanner:priority",
            "value": "standard"
        },
        {
            "name": "amazon:inspector:sbom_scanner:priority_intelligence",
            "value": "unverified"
        }
    ],
    "published": "2025-04-17T18:48:20Z",
    "ratings": [
        {
            "method": "other",
            "severity": "medium",
            "source": {
                "name": "AMAZON_INSPECTOR",
                "url": "https://aws.amazon.com/inspector/"
            }
        }
    ],
    "source": {
        "name": "AMAZON_INSPECTOR",
        "url": "https://aws.amazon.com/inspector/"
    },
    "updated": "2025-04-17T18:48:20Z"
}
```


Amazon Inspector SBOM Generator license collection

The Amazon Inspector SBOM Generator helps track license information in a software bill of materials (SBOM). It collects license information from supported packages across operating systems and programming languages. With standardized license expressions in your generated SBOM, you can understand your licensing obligations.

Collect license information

Example command

The following example shows how to collect license information from a directory.

```
./inspector-sbomgen directory --path /path/to/your/directory/ --collect-licenses
```

SBOM component example

The following example shows a component entry in the generated SBOM.

```
"components": [
    {
      "bom-ref": "comp-2",
      "type": "application",
      "name": "sample-js-pkg",
      "version": "1.2.3",
      "licenses": [
        {
          "expression": "Apache-2.0 AND (MIT OR GPL-2.0-only)"
        }
      ],
      "purl": "pkg:npm/sample-js-pkg@1.2.3",
    }
  ]
```

Note

When a package nests child components – for example DPKG and RPM binary packages under a source package – each child reports its own license rather than inheriting the parent's license or being omitted. Nested-component license reporting applies to container, localhost, and volume scans run with `--collect-licenses`.

Supported packages

The following programming languages and operating system packages are supported for license collection.

Target	Package manager	License information source	Type
Alma Linux	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages	OS

Target	Package manager	License information source	Type
		• /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	
Amazon Linux	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS

Target	Package manager	License information source	Type
CentOS	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS
Fedora	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS

Target	Package manager	License information source	Type
OpenSUSE	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS
Oracle Linux	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS

Target	Package manager	License information source	Type
Photon OS	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS
RHEL	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS

Target	Package manager	License information source	Type
Rocky Linux	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS
SLES	RPM	• /usr/lib/sysimage/rpm/rpmdb.sqlite • /usr/lib/sysimage/rpm/Packages • /usr/lib/sysimage/rpm/Packages.db • /var/lib/rpm/rpmdb.sqlite • /var/lib/rpm/Packages • /var/lib/rpm/Packages.db	OS
Alpine Linux	APK	`/lib/apk/db/installed`	OS
Chainguard	APK	`/lib/apk/db/installed`	OS

Target	Package manager	License information source	Type
Debian	DPKG	`/usr/share/doc/*/copyright`	OS
Ubuntu	DPKG	`/usr/share/doc/*/copyright`	OS
Node.js	Javascript	`node_modules/*/package.json`	Programing language
PHP	Composer package	• `composer.lock` • `/vendor/composer/installed.json`	Programing language
Go	Go	`LICENSE`	Programing language
Python	Python/Egg/Wheel	• `.dist-info/METADATA` • `.egg-info` • `.egg-info/PKG-INFO`	Programing language
Ruby	RubyGem	`*.gemspec`	Programing language
Rust	crate	`Cargo.toml`	Programming language

License expression standardization

The SPDX license expressions format provides accurate representation of licensing terms found in open source software. The Amazon Inspector SBOM Generator standardizes all license information into SPDX license expressions through rules described in this section. The rules provide consistency and compatibility across licensing information.

SPDX short form identifier mapping

All license names are mapped to SPDX short form identifiers. For example, `MIT License` is shortened to `MIT`.

Multiple license combination

You can combine more than one license with the `AND` operator. The following is an example command showing how to format your command.

```
MIT AND Apache-2.0
```

Custom license prefix

Custom licenses are prefixed with `LicenseRef`, such as `LicenseRef-CompanyPrivate`.

Custom exception prefix

Custom exceptions are prefixed with `AdditionRef-`, such as `AdditionRef-CustomException`.

Non-license values

Values that do not name a license are ignored rather than recorded. This includes the SPDX `NOASSERTION` and `NONE` assertions and the full text of a license.

What is a package URL?

A package URL or PURL is a standardized format used to identify software packages, components, and libraries across different package management systems. The format makes it easier to track, analyze, and manage dependencies in software projects, particularly when generating a Software Bill of Materials (SBOMs).

PURL structure

The PURL structure is similar to a URL and is composed of multiple components:

- pkg – The literal prefix

- `type`– The package type
- `namespace` – The grouping
- `name` – The package name
- `version` – The package version
- `qualifiers` – Extra key-value pairs
- `subpath` – The filepath in the package

Example PURL

The following is an example of how a PURL might look.

```
pkg:<type>/<namespace>/<name>@<version>?<qualifiers>#<subpath>
```

The generic PURL

A generic PURL is used to represent software packages and components that don't fit into established package ecosystems, such as npm, pypi, or maven. It identifies software components and captures metadata that might not align with specific package management systems. A generic PURL is useful for a variety of software projects, from compiled binaries to platforms, such as Apache and WordPress. Its allows it to be applied across a wide range of use cases, including compiled binaries, web platforms, and custom software distributions.

Key use cases

- Supports compiled binaries and is useful for Go and Rust
- Supports web platforms, such as Apache and WordPress, where a package might not be associated with traditional package managers.
- Supports custom legacy software by allowing organizations to reference internally developed software or systems lacking formal packages.

Example format

The following is an example of the generic PURL format.

```
pkg:generic/<namespace>/<name>@<version>?<qualifiers>
```

Additional examples of the generic PURL format

The following are additional examples of the generic PURL format.

Compiled Go binary

The following represents the `inspector-sbomgen binary` compiled with a Go.

```
pkg:generic/inspector-sbomgen?go_toolchain=1.22.5
```

Compiled Rust binary

The following represents the `myrustapp` binary compiled with Rust.

```
pkg:generic/myrustapp?rust_toolchain=1.71.0
```

Apache project

The following refers to an http project under the Apache namespace.

```
pkg:generic/apache/httpd@1.0.0
```

WordPress software

The following refers to a core WordPress software.

```
pkg:generic/wordpress/core/wordpress@6.0.0
```

WordPress theme

The following refers to a custom WordPress theme.

```
pkg:generic/wordpress/theme/mytheme@1.0.0
```

WordPress plugin

The following refers to a custom WordPress plugin.

```
pkg:generic/wordpress/plugin/myplugin@1.0.0
```

Handling unresolved or non-standard version references in the Amazon Inspector SBOM Generator

The Amazon Inspector SBOM Generator locates and parses supported artifacts within a system by identifying dependencies directly from source files. It's not a package manager and does not resolve version ranges, infer versions based on dynamic references, or handle registry lookups. It collects dependencies only as they're defined in project source artifacts. In many cases, dependencies in package manifests, such as `package.json`, `pom.xml`, or `requirements.txt`, are specified using unresolved or range-based versions. This topic includes examples of how these dependencies might look.

Recommendations

The Amazon Inspector SBOM Generator extracts dependencies from source artifacts, but doesn't resolve or interpret version ranges or dynamic references. For more accurate vulnerability scanning and SBOMs, we recommend using resolved, semantic version identifiers in project dependencies.

Java

For Java, Maven projects can use version ranges to define dependencies in the `pom.xml` file.

```
<dependency>
    <groupId>org.inspector</groupId>
    <artifactId>inspector-api</artifactId>
    <version>(,1.0]</version>
</dependency>
```

The range specifies that any version up to and including 1.0 is acceptable. However, if a version is not a resolved version, the Amazon Inspector SBOM Generator won't collect it because it cannot be mapped to a specific release.

JavaScript

For JavaScript, the `package.json` file can include version ranges that resemble the following:

```
"dependencies": {
    "ky": "^1.2.0",
    "registry-auth-token": "^5.0.2",
    "registry-url": "^6.0.1",
    "semver": "^7.6.0"
}
```

The ^ operator specifies that any version greater than or equal to the specified version is acceptable. However, if the specified version is not a resolved version, the Amazon Inspector SBOM Generator won't collect it becaue doing so can lead to false positives during vulnerability detection.

Python

For Python, the `requirements.txt` file can include entries with a boolean expression.

```
requests>=1.0.0
```

The >= operator specifies that any version greater than or equal to `1.0.0` is acceptable. Because this particular expression doesn't specify an exact version, the Amazon Inspector SBOM Generator cannot reliably collect a version for vulnerability analysis.

The Amazon Inspector SBOM Generator doesn't support non-standard or ambiguous version identifiers, such as beta, latest, or snapshot.

```
pkg:maven/org.example.com/testmaven@1.0.2%20Beta-RC-1_Release
```

> **Note**
>
> The use of a non-standard suffix, such as Beta-RC-1_Release, isn't compliant with standard semantic versioning and cannot be assessed for vulnerabilities within the Amazon Inspector detection engine.

Using CycloneDX namespaces with Amazon Inspector

Amazon Inspector provides you with CycloneDX namespaces and property names that you can use with SBOMs. This section describes all of the custom key/value properties that might be added to

components in CycloneDX SBOMs. For more information, see CycloneDX property taxonomy on the GitHub website.

`amazon:inspector:sbom_scanner` namespace taxonomy

The Amazon Inspector Scan API uses the `amazon:inspector:sbom_scanner` namespace and has the following properties:

Property	Description
`amazon:inspector:sbom_scanner:cisa_kev_date_added`	Indicates when the vulnerability was added to the CISA Known Exploited Vulnerabilities catalog.
`amazon:inspector:sbom_scanner:cisa_kev_date_due`	Indicates when the vulnerability fix is due according to the CISA Known Exploited Vulnerabilities catalog.
`amazon:inspector:sbom_scanner:critical_vulnerabilities`	Count of the total number of critical severity vulnerabilities found in the SBOM.
`amazon:inspector:sbom_scanner:exploit_available`	Indicates if an exploit is available for the given vulnerability.
`amazon:inspector:sbom_scanner:exploit_last_seen_in_public`	Indicates when an exploit was last seen in public for the given vulnerability.
`amazon:inspector:sbom_scanner:fixed_version: component_bom_ref`	Provides the fixed version of the indicated component for the given vulnerability.
`amazon:inspector:sbom_scanner:high_vulnerabilities`	Count of the total number of high severity vulnerabilities found in the SBOM.
`amazon:inspector:sbom_scanner:info`	Provides scan context for a given component, for example: "Component scanned: no vulnerabilities found."

Property	Description
amazon:inspector:sbom_scanner:is_malicious	Indicates if OpenSSF identifies affected components as malicious.
amazon:inspector:sbom_scanner:low_vulnerabilities	Count of the total number of low severity vulnerabilities found in the SBOM.
amazon:inspector:sbom_scanner:medium_vulnerabilities	Count of the total number of medium severity vulnerabilities found in the SBOM.
amazon:inspector:sbom_scanner:path	The path to the file that yields the subject package information.
amazon:inspector:sbom_scanner:priority	The recommended priority for fixing a given vulnerability. The values in descending order are "IMMEDIATE", "URGENT", "MODERATE", and "STANDARD".
amazon:inspector:sbom_scanner:priority_intelligence	The quality of intelligence used to determine the priority for a given vulnerability. The values include "VERIFIED" or "UNVERIFIED".
amazon:inspector:sbom_scanner:warning	Provides context for a why a given component was not scanned, for example: "Component skipped: no purl provided."

amazon:inspector:sbom_generator namespace taxonomy

The Amazon Inspector SBOM Generator uses the amazon:inspector:sbom_generator namespace and has the following properties:

Property	Description
amazon:inspector:sbom_generator:cpu_architecture	The CPU architecture of the system being inventoried (x86_64).

Property	Description
amazon:inspector:sbom_generator:ec2:instance_id	The Amazon EC2 instance ID.
amazon:inspector:sbom_generator:ec2:instance_type	The Amazon EC2 Instance type
amazon:inspector:sbom_generator:live_patching_enabled	A boolean value indicating whether live patching is enabled on Amazon EC2 Amazon Linux.
amazon:inspector:sbom_generator:live_patched_cves	A list of CVEs patched through live patching on Amazon EC2 Amazon Linux.
amazon:inspector:sbom_generator:dockerfile_finding: *inspector_finding_id*	Indicates that an Amazon Inspector finding in a component is related to Dockerfile checks.
amazon:inspector:sbom_generator:image_id	The hash belonging to the container image config file (also known as the Image ID).
amazon:inspector:sbom_generator:image_arch	The architecture of the container image.
amazon:inspector:sbom_generator:image_author	The author of the container image.
amazon:inspector:sbom_generator:image:cmd: *count*	An absolute directory within the container image defined in default CMD configured at image build time.
amazon:inspector:sbom_generator:image:entrypoint: *count*	An absolute directory within the container image defined in default ENTRYPOINT configured at image build time.
amazon:inspector:sbom_generator:image:workdir	The WORKDIR directory of the container image configured at image build time.

Property	Description
amazon:inspector:sbom_generator:image:base_layer	The diff ID of the container image's bottom-most layer, which identifies the image's base layer for downstream consumers.
amazon:inspector:sbom_generator:image:cis:docker: *check*	The pass or fail result of a supported CIS Docker Benchmark check (4.1, 4.6, 4.7, 4.9, or 4.10). Recorded only when the experimental --enable-cis-docker argument is used. For more information, see Evaluate CIS Docker Benchmark checks.
amazon:inspector:sbom_generator:image:cis:docker:checks_passed	A boolean value indicating whether every supported CIS Docker Benchmark check passed. Recorded only when the experimental --enable-cis-docker argument is used.
amazon:inspector:sbom_generator:image_docker_version	The docker version used to build the container image.
amazon:inspector:sbom_generator:is_duplicate_package	Indicates that the subject package was found by more than one file scanner.
amazon:inspector:sbom_generator:duplicate_purl	Indicates the duplicated package PURL found by another scanner.
amazon:inspector:sbom_generator:kernel_name	The kernel name of the system being inventoried.
amazon:inspector:sbom_generator:kernel_version	The kernel version of the system being inventoried.
amazon:inspector:sbom_generator:kernel_component	A boolean value indicating whether a subject package is a kernel component
amazon:inspector:sbom_generator:running_kernel	A boolean value that indicates if a subject package is the running kernel

Property	Description
amazon:inspector:sbom_generator:layer_diff_id	The hash of the uncompressed container image layer.
amazon:inspector:sbom_generator:replaced_by	The value that replaces the current Go module.
amazon:inspector:sbom_generator:os_hostname	The hostname of the system being inventoried.
amazon:inspector:sbom_generator:host:process_id: *pid*:path	The executable path of a running process that was matched to the component during a localhost scan. Emitted only when you enable the `--collect-processes` option.
amazon:inspector:sbom_generator:host:process_id: *pid*:started_at	The start time (RFC 3339, UTC) of a running process that was matched to the component during a localhost scan. Emitted only when you enable the `--collect-processes` option.
amazon:inspector:sbom_generator:source_file_scanner	The scanner that found the file that contains package information, for example: `/var/lib/dpkg/status` .
amazon:inspector:sbom_generator:source_package_collector	The collector that extracted the package name and version from a specific file.
amazon:inspector:sbom_generator:source_path	The path to the file that the subject package information was extracted from.
amazon:inspector:sbom_generator:file_size_bytes	Indicates file size of a given artifact.
amazon:inspector:sbom_generator:unresolved_version	Indicates a version string that has not been resolved by package manager..

Property	Description
amazon:inspector:sbom_generator:experimental:transitive_dependency	Indicates indirect dependencies from a package manager.
amazon:inspector:sbom_generator:subscription:enabled	A boolean value indicating whether a subscription is enabled, such as RHEL EUS/E4S or Ubuntu Pro.
amazon:inspector:sbom_generator:subscription:name	The name of the active subscription (e.g., EUS, E4S, Pro).
amazon:inspector:sbom_generator:subscription:locked_version	The version locked by the active RHEL subscription (RHEL EUS/E4S only).
amazon:inspector:sbom_generator:metadata:host:hostname	The hostname of the scanned system.
amazon:inspector:sbom_generator:metadata:host:kernel_name	The kernel name of the operating system (e.g., Linux, Darwin, Windows_NT).
amazon:inspector:sbom_generator:metadata:host:kernel_version	The kernel version string of the operating system.
amazon:inspector:sbom_generator:metadata:host:cpu_architecture	The CPU architecture of the system (e.g., x86_64, arm64).
amazon:inspector:sbom_generator:metadata:host:bootdisk_id	Unique identifier of the boot disk.
amazon:inspector:sbom_generator:metadata:host:boot_id	Unique identifier for the current boot session.
amazon:inspector:sbom_generator:metadata:host:boot_time	System boot time in ISO 8601 format.

Property	Description
amazon:inspector:sbom_generator:metadata:host:system_id	Persistent system identifier (machine-id on Linux, MachineGuid on Windows).
amazon:inspector:sbom_generator:metadata:host:system_serial	Hardware serial number from system firmware.
amazon:inspector:sbom_generator:metadata:host:network_interfaces: *name*:hardware	MAC address of the network interface.
amazon:inspector:sbom_generator:metadata:host:network_interfaces: *name*:ipv4	IPv4 address(es) assigned to the interface.
amazon:inspector:sbom_generator:metadata:host:network_interfaces: *name*:ipv6	IPv6 address(es) assigned to the interface.
amazon:inspector:sbom_generator:metadata:host:sbomgen_tag: *key*	Custom user-defined tags passed via the --tag CLI argument.
amazon:inspector:sbom_generator:metadata:imds:provider	The cloud provider detected via IMDS (aws, azure, gcp).
amazon:inspector:sbom_generator:metadata:imds:instance_id	The Amazon EC2 instance ID or Azure VM name.
amazon:inspector:sbom_generator:metadata:imds:instance_type	The instance type (e.g., t3.micro, Standard_D2s_v3).
amazon:inspector:sbom_generator:metadata:imds:instance_location	The region/location of the instance.

Property	Description
amazon:inspector:sbom_generator:metadata:imds:instance_partition	The cloud partition (aws, aws-cn, aws-us-gov for AWS, or AzurePublicCloud for Azure).
amazon:inspector:sbom_generator:metadata:imds:account_id	The AWS account ID of the Amazon EC2 instance, obtained from the instance identity document (AWS only).
amazon:inspector:sbom_generator:metadata:imds:resource_type	The type of cloud resource being scanned (e.g., aws_ec2_instance).
amazon:inspector:sbom_generator:metadata:imds:instance_managed_id	Amazon EC2 Systems Manager managed instance ID (AWS only).
amazon:inspector:sbom_generator:metadata:imds:tenant_id	The Microsoft Entra ID tenant the instance's subscription belongs to (Azure only, populated when a managed identity is assigned to the VM).
amazon:inspector:sbom_generator:metadata:imds:resource_group	The Azure resource group the instance belongs to (Azure only).
amazon:inspector:sbom_generator:metadata:imds:subscription_id	The Azure subscription ID associated with the instance (Azure only).
amazon:inspector:sbom_generator:metadata:imds:vm_id	Azure VM unique identifier (Azure only).
amazon:inspector:sbom_generator:metadata:imds:project_id	The Google Cloud project ID the instance belongs to (GCP only).

Property	Description
`amazon:inspector:sbom_generator:metadata:imds:numeric_project_id`	The numeric Google Cloud project ID the instance belongs to (GCP only).
`amazon:inspector:sbom_generator:metadata:imds:instance_name`	The instance name as reported by the GCP metadata service (GCP only).
`amazon:inspector:sbom_generator:metadata:host:open_port: port:protocol`	Indicates an open port of a runtime resource (i.e. EC2)
`amazon:inspector:sbom_generator:hardened_image:vendor`	The vendor of a hardened container image

Plugin developer guide

This guide explains how to extend Amazon Inspector SBOM Generator (inspector-sbomgen) with custom Lua plugins. Plugins let you add support for new package ecosystems without modifying sbomgen's source code or having to re-compile.

For the complete function catalog, see the the section called “Plugin API reference”. For guidance on writing tests, see the the section called “Plugin testing guide”.

Overview

Sbomgen plugins are written in Lua, and follow a two step pipeline:

- **Discovery** — scan the artifact's file list and report which files are relevant to your ecosystem.
- **Collection** — parse each discovered file and push package findings into the SBOM.

Plugin Event Model

Discovery plugins need a way to tell collection plugins that files containing package metadata were discovered in the artifact under inventory. To facilitate this data sharing, discovery plugins define

an **event name** and return a list of file paths. Collection plugins **subscribe** to that event and receive each matched file path. This decouples file detection from parsing — you can have one discovery plugin feed multiple collectors (fan-out pattern). However, each discovery plugin must have a unique event name, and each collection plugin must have a unique collector name. See the section called “Plugin Collision Rules” for details.

Developers may recognize this as the **observer** design pattern.

This design lets a single discovery plugin trigger multiple independent analyses in a performant manner. For example, one discovery plugin can locate every `requirements.txt` in an artifact, then feed:

- A **package collector** that parses each line into SBOM findings (`name==version`).
- A **secrets collector** that flags lines containing API keys or tokens accidentally pinned as versions.
- A **policy collector** that reports unpinned or wildcard version specifiers.

Each collector runs independently against the same file list without re-walking the artifact's file system. This also enables plugin authors to add new collection plugins that subscribes to existing events without having to change the corresponding discovery plugin.

Quick Start: Creating New Plugins

The fastest way to create a new plugin is using the built-in scaffolding command:

```
inspector-sbomgen plugin new
```

The command prompts for a plugin name and project directory. Pressing Enter accepts the default shown in brackets:

```
Plugin name (identifies the software ecosystem your plugin will inventory, e.g. debian-
dpkg, rhel-rpm, python-pip, cmake) [my-custom-ecosystem]: cmake
Project directory [my-sbomgen-plugins]:
```

You can also pass arguments directly:

```
inspector-sbomgen plugin new --name cmake --path /tmp/custom-plugins
```

Starting with a working example

If you want to experiment with plugins before writing your own logic, create a new plugin using the `--with-example` flag:

```
inspector-sbomgen plugin new --with-example
```

This generates a fully working plugin project with a sample lockfile parser, test data, and passing tests. The example plugin discovers `example.lock` files, parses `name==version` entries, and pushes packages into the SBOM. You can run the tests immediately to see the plugin system in action, then modify the code to target your actual ecosystem.

For advanced scaffolding that includes all optional override functions (`get_scanner_name`, `get_event_name`, `get_scanner_groups`, multi-event discovery, etc.), use the `--with-overrides` flag (more on this later):

```
inspector-sbomgen plugin new --with-overrides
```

Completing Your Plugin

After scaffolding, the generated plugin files contain `TODO` markers indicating where to add your ecosystem-specific logic. Work through these markers to turn the scaffold into a working plugin:

- Replace all `TODO` markers with your actual values
- Update the file pattern in `discover()` to match your target files
- Implement parsing logic in `collect()` and call `sbomgen.push_package()` for each package

Test Your Plugin

There are two ways to test your plugins:

1. Built-in test harness — Use `inspector-sbomgen plugin test` to validate plugin logic during development. This runs your `init_test.lua` test files without needing a real artifact to scan:

```
inspector-sbomgen plugin test --path ./my-plugins
```

See the the section called “Plugin testing guide” for details on writing test files and using the `testing.*` API.

2. End-to-end scan — Invoke your plugin using standard sbomgen commands to verify it works against real artifacts. For this approach, you need to provide both an artifact containing the files your plugin targets (e.g., a directory with `requirements.txt` or equivalent) and the path to your plugin directory:

```
inspector-sbomgen directory \
    --path /path/to/test/dir \
    --plugin-dir ./my-plugins \
    --disable-native-scanners \
    -o sbom.json
```

The `--disable-native-scanners` flag ensures only your Lua plugins run, making it easier to test without output from the built-in (native) scanners.

IDE Setup

Sbomgen provides code completion, type checking, and inline documentation for the entire `sbomgen.*` API in VS Code.

VS Code with Lua Language Server

- Install the Lua extension: sumneko.lua
- Open any `.lua` file in your plugin project

That's it! The `plugin new` command generates `.vscode/settings.json` and `library/sbomgen.lua` which are automatically detected by the Lua Language Server. You'll immediately get:

- Code completion for all `sbomgen.*` functions
- Parameter hints with types
- Hover documentation
- Type checking

Plugin Directory Structure

Sbomgen discovery and collection plugins must adhere to the following directory structure:

```
{plugin-dir}/
```

```
### discovery/
#   ### {platform}/
#       ### {category}/
#           ### {ecosystem}/
#               ### init.lua          # REQUIRED entrypoint
### collection/
    ### {platform}/
        ### {category}/
            ### {ecosystem}/
                ### init.lua          # REQUIRED entrypoint
```

These directory names carry semantic meaning — sbomgen uses them to derive default metadata for your plugin, including the scanner name, event name, scanner groups, and platform filtering. This reduces the amount of boilerplate developers would otherwise have to write. Choosing the correct values ensures your plugin integrates properly with sbomgen's scanner selection and execution model.

The sections below explore the directory structure in greater detail, providing guidance on semantic meaning and conventions.

Platform

The platform directory controls which operating systems your plugin runs on.

Value	When to use
`cross-platform`	Plugin works on any OS (most plugins)
`linux`	Linux-specific detection logic
`windows`	Windows-specific detection logic
`macos`	macOS-specific detection logic

Category

The category directory determines the default scanner groups assigned to your plugin, which controls whether it runs by default or requires explicit opt-in. See the section called “Scanner Selection” for how groups affect execution.

Value	Default groups	When to use
`proglang`	`programming-language-packages` , `pkg-scanner`	Programming language packages (pip, npm, maven, etc.)
`os`	`os`, `pkg-scanner`	OS package managers (dpkg, rpm, apk, etc.)
`extra-ecosystems`	`extra-ecosystems` , `pkg-scanner`	Applications and runtimes (nginx, curl, wordpress, etc.)

If you use a category name that doesn't match any of the above, the category name itself is used as the group.

Ecosystem

A name for the specific package ecosystem (e.g., `python-pip`, `python-poetry`, `debian-dpkg`, `curl`). Hyphenated names are a common convention but not a strict requirement.

The scanner name and collector name are derived directly from the ecosystem directory name.

Event name pairing

Discovery and collection plugins at the same directory path are automatically paired. For example, a discovery plugin at `discovery/cross-platform/proglang/python-pip/` automatically pairs with `collection/cross-platform/proglang/python-pip/`.

You can override this by defining `get_event_name()` and `subscribe_to_event()` in your plugins.

Discovery Plugins

A discovery plugin only requires the `discover()` function. All other functions are optional — defaults are derived from the directory path.

Most discovery plugins work by locating files whose names or paths identify a specific ecosystem — for example, `requirements.txt` for Python pip, `package.json` for npm, or `Cargo.lock` for

Rust cargo. The `sbomgen.find_files_by_*` functions perform this matching outside the Lua VM, which makes them significantly faster than iterating the full file list in Lua:

```
-- REQUIRED: Scans the artifact and returns a table of file paths.
function discover()
    return sbomgen.find_files_by_name({"requirements.txt"})
end
```

`discover()` must return a Lua table (array) of strings. If no files are found, return an empty table `{}`.

Common discovery patterns

Goal	Recommended function
Match one or more exact filenames	`sbomgen.find_files_by_name({names})`
Match filenames case-insensitively	`sbomgen.find_files_by_name_icase({names})`
Match by path suffix (e.g., `/pom.properties`)	`sbomgen.find_files_by_suffix({suffixes})`
Match by full-path regex	`sbomgen.find_files_by_path_regex({patterns})`
Glob-style basename match (e.g., `*.lock`)	`sbomgen.glob_find_files(pattern)`

When your logic requires post-filtering — for example, keeping files matching a suffix but excluding build-output directories — combine a `find_files_by_*` call with a Lua loop:

```
function discover()
    local found = {}
    for _, f in ipairs(sbomgen.find_files_by_suffix({".conda-meta.json"})) do
        if not f:match("[/\\]%.cache[/\\]") then
            table.insert(found, f)
        end
    end
    return found
```

```
end
```

Avoid `sbomgen.get_file_list()` in discovery unless no other matcher fits — it copies every path into the Lua VM and can take several seconds on large artifacts. See the the section called "Plugin API reference" for details.

Multi-event discovery

By default, all files returned by `discover()` are published to a single event (from `get_event_name()`). If your scanner needs to route different files to different collectors, return a keyed table instead:

```
function discover()
    return {
        EventNameFoundCurl = sbomgen.find_files_by_name({"curl", "curl.exe"}),
        EventNameFoundLibcurl = sbomgen.find_files_by_name({"curlver.h"}),
    }
end
```

When `discover()` returns a table with string keys, each key is treated as a separate event name and its value (a table of file paths) is published to that event. Collection plugins subscribe to specific events via `subscribe_to_event()` as usual.

This is backward compatible — returning a sequential table `{"file1", "file2"}` still works as single-event mode. The detection is automatic: tables with any string keys are multi-event, tables with only integer keys (or empty) are single-event.

When using multi-event, `get_event_name()` is not used for publishing (the event names come from the returned table keys). However, it is still called during plugin loading for collision detection, so it should return a unique value or be omitted to use the default.

Optional discovery functions

All of these have sane defaults derived from the directory path. Define them only if you need to override:

Function	Default	Override when...
`get_scanner_name()`	`{ecosystem}` (e.g., `python-pip`)	You want a custom scanner name

Function	Default	Override when...
`get_scanner_description()`	`"Lua discovery plugin: {ecosystem}"`	You want a custom description
`get_scanner_groups()`	Derived from category directory	You need non-standard groups
`get_event_name()`	Derived from directory path	You need custom event routing
`get_localhost_scan_paths()`	None	Your plugin needs specific paths scanned during `localhost` scans
`get_fileless_events()`	None	A collector reads a source other than the file system, so its event must fire even when `discover()` matched nothing

Localhost scan paths

When sbomgen runs a `localhost` scan, it walks user-specified directories plus any default paths declared by scanners. By default, Lua discovery plugins do not contribute any paths, so files outside the user-specified directories won't appear in the file list.

Define `get_localhost_scan_paths()` to return directories or file paths that the localhost walker should include:

```
function get_localhost_scan_paths()
    return {
        "/usr/bin",
        "/usr/local/bin",
    }
end
```

The returned paths are appended to the walker's scan list only during `localhost` scans — they have no effect on `container`, `directory`, or `archive` scans.

Platform-specific scan paths

When the files you care about live at different locations on Windows, macOS, and Linux, branch on `sbomgen.get_platform()` and return the appropriate paths for the host:

```
function get_localhost_scan_paths()
    local platform = sbomgen.get_platform()

    if platform == sbomgen.platform.WINDOWS then
        local drive = sbomgen.get_system_drive()
        return {
            drive .. "/Program Files/MyApp/myapp.exe",
            drive .. "/Program Files (x86)/MyApp/myapp.exe",
        }
    end

    if platform == sbomgen.platform.DARWIN then
        return {"/Applications/MyApp.app/Contents/MacOS/myapp"}
    end

    -- Linux
    return {
        "/usr/bin/myapp",
        "/usr/local/bin/myapp",
    }
end
```

On Windows, use `sbomgen.get_system_drive()` to resolve the system drive letter (e.g., `"C:"`) rather than hard-coding it. For paths derived from environment variables such as `LOCALAPPDATA` or `PROGRAMFILES`, iterate `sbomgen.get_env_vars()` and look up the value by key. See the the section called “Plugin API reference” for details.

Collection Plugins

A collection plugin only requires the `collect()` function. All other functions are optional.

`collect(file_path)` is called once per file discovered by the paired discovery plugin. The typical pattern is:

- **Read** the file's contents using `sbomgen.read_file()` (for small files loaded into memory) or `sbomgen.open_file()` (for large files read line-by-line).

- **Parse** the contents — string matching for simple manifests, `sbomgen.json_decode()` for JSON, `sbomgen.xml_decode()` for XML, or `sbomgen.search_binary()` for compiled binaries.
- **Publish** each discovered package by calling `sbomgen.push_package()` with the package's metadata.

```
-- REQUIRED: Called once per discovered file.
-- Parse the file and call sbomgen.push_package() for each package found.
function collect(file_path)
    local content, err = sbomgen.read_file(file_path)
    if err or not content then return end

    for line in content:gmatch("[^\r\n]+") do
        local name, version = line:match("^([%w%-%_%.]+)==(.+)$")
        if name and version then
            sbomgen.push_package({
                name = name,
                version = version,
                purl_type = "pypi",
                component_type = sbomgen.component_types.LIBRARY,
            })
        end
    end
end
```

`collect()` does not return a value. Every `push_package()` call requires `name`, `purl_type`, and `component_type`. See the the section called “Plugin API reference” for all supported fields.

Attaching metadata to components

Sbomgen supports two ways to attach metadata to a package component: **PURL qualifiers** and **CycloneDX properties**. They serve different purposes, and the choice between them has implications for how Amazon Inspector identifies vulnerabilities in the resulting SBOM.

Mechanism	Where it appears	Use for
`qualifiers`	Inside the package URL (e.g., `pkg:deb/debian/curl@7.88.1?arch=amd64`)	Data that is part of the package's identity

Mechanism	Where it appears	Use for
`properties`	In the SBOM's `components[].properties` array	Descriptive metadata that does not change how the package is identified

Recommendation: prefer CycloneDX properties (under your own namespace) for custom metadata. Properties do not alter a component's identity, so they cannot impact Amazon Inspector's vulnerability identification. Reserve PURL qualifiers for cases where your ecosystem's PURL type requires them.

PURL qualifiers

Some PURL qualifiers have semantic meaning to Amazon Inspector and influence vulnerability identification. For example, on `deb` components Inspector uses qualifiers such as `arch` and `distro` to select the correct vulnerability feed; on `generic` components for compiled binaries, qualifiers such as `go_toolchain` or `rust_toolchain` identify the toolchain used. Setting a qualifier Inspector does not recognize, or omitting one it expects, can cause vulnerabilities to be missed or misattributed.

See What is a package URL? in the Amazon Inspector user guide for the qualifier conventions Inspector recognizes per PURL type.

Set qualifiers via the `qualifiers` table on `sbomgen.push_package()`:

```
sbomgen.push_package({
    name = "curl",
    version = "7.88.1",
    purl_type = "deb",
    namespace = "debian",
    component_type = sbomgen.component_types.LIBRARY,
    qualifiers = {
        arch = "amd64",
        distro = "debian-12",
    },
})
```

Only set qualifiers when they align with Inspector's expectations for the PURL type. If you need to record metadata that is not part of the package's identity, use CycloneDX properties instead.

CycloneDX properties

CycloneDX properties are key-value annotations that appear in the SBOM's `components[].properties` array. They describe a component without affecting how it is identified, so they are the safe choice for plugin-defined metadata.

The `amazon:inspector:*` namespaces are reserved for Amazon Inspector. Specifically:

- `amazon:inspector:sbom_generator:*` — reserved for sbomgen and its built-in scanners.
- `amazon:inspector:sbom_scanner:*` — reserved for the Amazon Inspector Scan API.

Plugin authors must not emit keys inside these reserved namespaces. Writing into them can interfere with Inspector's behavior and may be overwritten. For the complete list of reserved keys, see Using CycloneDX namespaces with Amazon Inspector.

Use your own namespace (typically your organization or plugin identifier) when defining properties:

```
sbomgen.push_package({
    name = "requests",
    version = "2.28.1",
    purl_type = "pypi",
    component_type = sbomgen.component_types.LIBRARY,
    properties = {
        ["acme:python:manifest_path"] = file_path,
        ["acme:python:pinned"] = "true",
        ["acme:python:source"] = "requirements.txt",
    },
})
```

Key naming rules

Property keys are processed by sbomgen as follows:

- A key that **contains a colon** is used verbatim in the SBOM. Always include at least one colon in your keys so you control the namespace.
- A key that **does not contain a colon** is automatically prefixed with `amazon:inspector:sbom_generator:` — placing it inside the reserved Inspector namespace. Avoid this shape for custom properties.

```
properties = {
    ["acme:my_plugin:detected_via"] = "lockfile",  -- used as-is (recommended)
    detected_via                    = "lockfile",  -- becomes
 "amazon:inspector:sbom_generator:detected_via" (avoid)
}
```

The `sbomgen.properties.*` constants exist so that official scanners emit consistent keys inside the reserved namespace. They are not extension points for custom plugins — use your own namespace instead.

Properties and qualifiers on child components

Nested `children` are independent components. Each child has its own `properties` and `qualifiers` tables; metadata set on the parent does not propagate to children. Set values explicitly on each child that needs them.

Optional collection functions

Function	Default	Override when...
`get_collector_name()`	`{ecosystem}` (e.g., `python-pip`)	You want a custom collector name
`get_collector_description()`	empty string	You want a description
`subscribe_to_event()`	Derived from directory path	You need custom event routing
`finalize()`	None	You group findings across an event's files instead of emitting one per file

Running Your Plugins

For plugins to produce package metadata, sbomgen must be given an artifact to scan that contains the files your plugin targets (e.g., a directory with `requirements.txt`, `package.json`, or equivalent package manifest files).

Basic usage

```
inspector-sbomgen <artifact type> <arguments> --plugin-dir /path/to/plugins
```

Example:

```
inspector-sbomgen directory --path /target -o /tmp/sbom.json --plugin-dir /path/to/
plugins
```

With native scanners disabled (Lua-only mode)

```
inspector-sbomgen directory --path /target --plugin-dir /path/to/plugins --disable-
native-scanners -o sbom.json
```

With verbose logging

```
inspector-sbomgen directory --path /target --plugin-dir /path/to/plugins --verbose -o
 sbom.json
```

Listing Available Scanners

Use `list-scanners` to see every scanner available to sbomgen. This includes the built-in native scanners, any official Lua plugins bundled with sbomgen, and any custom Lua plugins you've supplied via `--plugin-dir`:

```
inspector-sbomgen list-scanners --plugin-dir /path/to/plugins
```

```
#################################################################################################
#    SCANNER NAME     # SOURCE #            GROUPS             #         DESCRIPTION
      #
#################################################################################################
# curl                # custom # extra-ecosystems              # Discovers curl version
      #
#                     #        # pkg-scanner                   # header files
 (curlver.h)    #
#################################################################################################
# python-requirements # custom # pkg-scanner                   # Discovers
 requirements*.txt #
```

```
#                    #        # programming-language-packages # files for Python pip
      #
#                    #        #                                # packages
      #
#################################################################################################
```

The SOURCE column shows where each scanner comes from:

Source	Meaning
`native`	Built-in scanner bundled with sbomgen
`official`	Lua plugins bundled with sbomgen
`custom`	User-provided Lua plugin loaded via `--plugin-dir`

Running `list-scanners` without `--plugin-dir` still includes both `native` and `official` scanners — those are always available. The `--plugin-dir` flag adds your `custom` scanners to the listing.

Components from custom plugins are marked with CycloneDX `scope: "optional"`

Components emitted by user-supplied Lua plugins loaded via `--plugin-dir` are tagged with the CycloneDX `scope: "optional"` field in the resulting SBOM. This distinguishes them from components produced by built-in native scanners and official Lua plugins bundled with sbomgen, which leave `scope` unset (consumers default a missing scope to `"required"` per the CycloneDX specification). The `scope` value applies to every component the plugin emits, including nested children, and you do not need to set it yourself in `sbomgen.push_package()`.

To list only Lua scanners without native scanners:

```
inspector-sbomgen list-scanners --plugin-dir /path/to/plugins --disable-native-scanners
```

Scanner Selection

Lua discovery plugins participate in the same scanner selection model as the built-in native scanners. By default, sbomgen runs all scanners whose groups match the default scanner groups for the artifact type. You can override this with three flags:

Run only specific scanners

Use `--scanners` to run only the named scanners. All other scanners are excluded:

```
inspector-sbomgen directory --path /target \
    --plugin-dir /path/to/plugins \
    --scanners python-requirements \
    -o sbom.json
```

This runs only the `python-requirements` scanner. You can pass multiple scanner names separated by commas, or pass a scanner group name (e.g., `programming-language-packages`) to enable every scanner that belongs to that group.

Exclude specific scanners

Use `--skip-scanners` to exclude named scanners while running everything else:

```
inspector-sbomgen directory --path /target \
    --plugin-dir /path/to/plugins \
    --skip-scanners python-poetry \
    -o sbom.json
```

This runs every default scanner except `python-poetry`. Like `--scanners`, this flag also accepts group names, so passing `--skip-scanners programming-language-packages` disables every scanner in that group.

> **Note**
>
> `--scanners` and `--skip-scanners` are mutually exclusive. Passing both produces an error.

Add scanners from non-default groups

The default scanner set depends on the artifact type being scanned (see the matrix in the section called “How groups affect selection” below). A scanner whose groups are not part of the default set for the artifact type will not run unless you opt it in. Use `--additional-scanners` to append scanners to the default set without replacing it:

```
inspector-sbomgen directory --path /target \
    --plugin-dir /path/to/plugins \
    --additional-scanners my-extra-scanner \
    -o sbom.json
```

This runs every default scanner for the artifact type, plus `my-extra-scanner`. The flag accepts a comma-separated list of scanner names or group names, and stacks with the default set rather than replacing it. Use `list-scanners` to check which groups a scanner belongs to.

How groups affect selection

The `get_scanner_groups()` function in your discovery plugin determines which groups the scanner belongs to. Whether a scanner runs by default depends on both its groups and the artifact type being scanned. The matrix below shows which groups are included in the default scanner set for each artifact type:

Group	directory / archive	container	localhost	volume	binary
os	—	✓	✓	✓	—
programming-language-packages	✓	✓	✓	✓	—
binary	✓	✓	—	—	✓
extra-ecosystems	—	✓	✓	✓	—

Group	directory / archive	container	localhost	volume	binary
dockerfile	✓	✓	—	—	—
custom	✓	✓	✓	✓	✓
certificate	—	—	—	—	—
machine-learning	—	—	—	—	—
pkg-scanner	—	—	—	—	—

A ✓ means every scanner in that group runs by default for that artifact type. A — means the group is not in the default set, so its scanners only run if explicitly selected via `--scanners` or `--additional-scanners`.

Notable details:

- **`custom`** is always in the default set — custom plugins loaded via `--plugin-dir` automatically receive the `custom` group, so they run by default regardless of artifact type.
- **`extra-ecosystems`** is default for `container`, `localhost`, and `volume` scans, but not for `directory`, `archive`, or `binary` scans. For those types you must pass `--additional-scanners` (by name or by the `extra-ecosystems` group) to include them.
- **`pkg-scanner`** is informational — it marks a scanner as a package collector for display in `list-scanners`, but does not by itself cause the scanner to run. Pair it with an execution group (e.g., `programming-language-packages`) in `get_scanner_groups()`.

For example, a plugin that returns `{sbomgen.groups.EXTRA_ECOSYSTEMS, sbomgen.groups.PACKAGE_COLLECTOR}` will run by default on container, localhost, and volume scans, but will require `--additional-scanners` (or `--scanners`) on directory, archive, and binary scans.

Plugin Collision Rules

Sbomgen enforces unique metadata across all loaded plugins to prevent silent overwrites and ensure SBOM integrity. When a collision is detected, the later plugin is **skipped** and a warning is logged.

What is checked

Metadata	Scope	On collision
Discovery event name (`get_event_name`)	All discovery plugins	Second plugin skipped
Scanner name (`get_scanner_name`)	All discovery plugins	Second plugin skipped
Collector name (`get_collector_name`)	All collection plugins	Second plugin skipped

What is allowed

Multiple collection plugins **can** subscribe to the same event via `subscribe_to_event()`. This is the intended fan-out pattern — one discovery plugin can feed multiple collectors that each do different things (e.g., one extracts packages, another detects secrets).

Avoiding collisions

If two plugins use the same scanner name, event name, or collector name, the second one loaded is skipped. To resolve collisions, rename the conflicting metadata by defining the appropriate override function in your plugin (`get_scanner_name()`, `get_event_name()`, or `get_collector_name()`).

Collision warning example

```
[custom:python-pip] SKIPPED: discovery event name "EventNameFoundPythonRequirements"
is already registered by [official:python-pip]. Each discovery plugin must have a
unique event name. Rename get_event_name() in your plugin to use a unique name.
```

The warning tells you which plugin was skipped, what collided, which plugin already owns that name, and which function to change.

Debugging

Console logging

Plugins can emit messages to sbomgen's console output using the following functions:

Function	Level	Visible by default?
`sbomgen.log_debug(message)`	DEBUG	No — requires `--verbose`
`sbomgen.log_info(message)`	INFO	Yes
`sbomgen.log_warn(message)`	WARN	Yes
`sbomgen.log_error(message)`	ERROR	Yes

All log output from a plugin is automatically prefixed with the plugin's source and path (e.g., `[custom:python-pip]`), so messages from different plugins are easy to distinguish. `log_info`, `log_warn`, and `log_error` always print; `log_debug` only prints when sbomgen is invoked with `--verbose`.

```
function discover()
    sbomgen.log_info("starting discovery")
    local files = sbomgen.find_files_by_name({"requirements.txt"})
    sbomgen.log_debug(string.format("matched %d files", #files))
    if #files == 0 then
        sbomgen.log_warn("no requirements.txt files found")
    end
    return files
end
```

Breakpoints

Use `sbomgen.breakpoint()` to pause plugin execution and block until you press Enter. This acts as a crude debugger — combine it with log statements to inspect state at specific points.

```
function discover()
    local files = sbomgen.find_files_by_name({"requirements.txt"})

    sbomgen.log_info(string.format("about to inspect %d files", #files))
    sbomgen.breakpoint("before file inspection — press Enter to continue")

    local found = {}
    for _, f in ipairs(files) do
        if not f:match("[/\\]tests[/\\]") then
            table.insert(found, f)
        end
    end

    sbomgen.log_info(string.format("kept %d files after filtering", #found))
    sbomgen.breakpoint("after filtering — press Enter to continue")

    return found
end
```

The breakpoint message is printed to stderr. Execution pauses until you press Enter, giving you time to review log output.

Common issues

Symptom	Cause	Fix
Plugin not loaded	Missing `init.lua`	Ensure entrypoint exists at the correct directory depth
"missing required function"	Typo in function name	Check that `get_scanner_name` , `get_scanner_description` , `get_scanner_groups` , `discover`, `get_event_name` , `get_local`

Symptom	Cause	Fix
		`host_scan_paths` , `get_collector_name` , `collect`, `subscribe_to_event` are defined
Collection plugin never called	Event name mismatch	Verify `get_event_name()` and `subscribe_to_event()` return the same string
No packages in SBOM	`push_package` not called or required fields missing	Ensure `name`, `purl_type` , and `component_type` are set in every `push_package` call (including children). Use `sbomgen.component_types.*` constants.
Runtime error in plugin	Lua error during execution	Check sbomgen output for warning messages with the error details
"SKIPPED: discovery event name ... is already registered"	Another plugin uses the same event name	Rename `get_event_name()` to a unique value
"SKIPPED: scanner name ... is already registered"	Another plugin uses the same scanner name	Rename `get_scanner_name()` to a unique value
"SKIPPED: collector name ... is already registered"	Another plugin uses the same collector name	Rename `get_collector_name()` to a unique value

API Reference

The complete function catalog is maintained in a companion document:

→ **the section called “Plugin API reference”**

The API reference covers every `sbomgen.*` function (file I/O, binary utilities, package output, regex, structured parsing, Windows registry, logging, debugging), the `testing.*` API available in test files, all built-in constants (`properties`, `groups`, `component_types`, `platform`), and the plugin lifecycle globals.

Error Handling

API functions that can fail return two values: `value, err`. On success, `err` is `nil`. On failure, the first value is `nil` and `err` is an error string.

```
local content, err = sbomgen.read_file(path)
if err then
    sbomgen.log_error("failed to read " .. path .. ": " .. err)
    return
end
-- content is safe to use here
```

If a plugin raises an unhandled Lua error, sbomgen logs a warning and continues with the next file or plugin. Other plugins are not affected.

Sandbox Restrictions

Plugins run in a sandboxed Lua VM with limited standard library access:

Library	Available	Notes
`base`	✓	`dofile`, `loadfile`, `loadstring` are removed
`string`	✓	Full string manipulation
`table`	✓	Full table manipulation
`math`	✓	Full math library
`package`	✓	`require()` restricted to plugin directory
`io`	✗	Use `sbomgen.*` I/O functions instead

Library	Available	Notes
`os`	✗	Blocked for security
`debug`	✗	Blocked to prevent VM introspection
`coroutine`	✗	Not loaded

Direct filesystem access via `io.open` or `os.execute` is not available. All file operations must go through the `sbomgen` API, which ensures consistent behavior across artifact types.

The `sbomgen` file functions confine reads to the artifact under inventory: a path that resolves outside the artifact root (for example via `../` traversal) is rejected and returns an error. The `localhost` artifact type is the exception — it inventories the host itself, so reads are not confined to a narrower root.

`require()` can load modules only from within the plugin's own directory tree. Parent-directory traversal such as `require("../shared")` is blocked.

SBOM contents are not sanitized

The read boundary above governs what a plugin can read, not what it writes. Sbomgen does not inspect or filter the data a plugin emits into the SBOM, and does not detect or redact secrets, credentials, or other sensitive values. Whatever a plugin places into a finding appears in the output SBOM and travels wherever that SBOM is published, so only emit data derived from the artifact you intend to inventory.

Sharing Code Between Plugins

You can use `require()` to load helper modules from within your plugin's directory:

```
my-ecosystem/
### init.lua
### helpers.lua
```

```
-- helpers.lua
local M = {}
function M.parse_version(s)
```

```
    return string.match(s, "(%d+%.%d+%.%d+)")
end
return M
```

```
-- init.lua
local helpers = require("helpers")

function subscribe_to_event() return "MyEvent" end

function collect(file_path)
    local content, err = sbomgen.read_file(file_path)
    if err then return end
    local version = helpers.parse_version(content)
    -- ...
end
```

Subdirectories with `init.lua` are also supported:

```
my-ecosystem/
### init.lua
### parsers/
    ### init.lua
```

```
local parsers = require("parsers")
```

`require()` is restricted to your plugin's directory. You cannot load modules from other plugins or system paths. Third-party Lua libraries (e.g., from LuaRocks) are not supported — only local helper modules within the plugin directory can be loaded.

Plugin testing guide

Plugin authors write and test Lua plugins entirely in Lua — no Go toolchain required. Tests are co-located with the plugin under `init_test.lua` and run via the `inspector-sbomgen plugin test` command.

For general plugin authoring, see the the section called “Plugin developer guide”. For the complete function catalog (including the `testing.*` API), see the the section called “Plugin API reference”.

```
-- init_test.lua (next to init.lua)
```

```
function test_discovers_curl_version()
    local result = testing.scan_directory("_testdata/include/curl")
    testing.assert_equals(1, #result.findings)
    testing.assert_equals("libcurl", result.findings[1].name)
    testing.assert_equals("8.14.1", result.findings[1].version)
end
```

```
# Run every test under a plugin directory
inspector-sbomgen plugin test --path ./my-plugins

# Verbose output — show each test name and result
inspector-sbomgen plugin test --path ./my-plugins -v
```

Quick Start

1. Create a test file

Place `init_test.lua` next to your plugin's `init.lua`:

```
my-plugin/
### discovery/cross-platform/extra-ecosystems/curl/
#   ### init.lua
#   ### init_test.lua
#   ### _testdata/
#       ### include/curl/curlver.h
```

2. Write test functions

Any global function starting with `test_` is discovered and executed:

```
function test_finds_libcurl()
    local result = testing.scan_directory("_testdata/include/curl")
    testing.assert_equals(1, #result.findings)
    testing.assert_equals("libcurl", result.findings[1].name)
end

function test_no_findings_for_empty_dir()
    local result = testing.scan_directory("_testdata/empty")
    testing.assert_equals(0, #result.findings)
end
```

3. Run tests

```
inspector-sbomgen plugin test --path ./my-plugin
```

Directory Layout

Plugin structure

Test files and test data are co-located with the plugin:

```
my-plugins/
### discovery/
#   ### cross-platform/
#       ### extra-ecosystems/
#           ### curl/
#               ### init.lua          # plugin source
#               ### init_test.lua     # test file
#               ### _testdata/        # test fixtures
#                   ### include/curl/curlver.h
#                   ### binaries/unix/curl
### collection/
#   ### cross-platform/
#       ### extra-ecosystems/
#           ### curl-installation/
#               ### init.lua
#               ### init_test.lua
```

Test file naming

- Default: `init_test.lua` next to the plugin's `init.lua`
- Multiple test files per plugin: any file matching `*_test.lua` is discovered
- Examples: `init_test.lua`, `parsing_test.lua`, `discovery_test.lua`

Test data: `_testdata/`

Test data lives in `_testdata/` next to the plugin. The leading underscore is a convention that keeps fixtures visually separate from plugin source; the `plugin test` command does not descend into `_testdata/` when searching for `*_test.lua` files, so fixtures are never mistaken for test files.

Test files reference fixtures with relative paths:

```
local result = testing.scan_directory("_testdata/include/curl")
```

Paths are resolved relative to the directory containing the test file.

The `testing.*` API

Scan Functions

Each scan function creates an artifact, runs the plugin's discovery → collection pipeline, and returns findings. The test author never manually creates artifacts, event buses, or registries.

```
-- Scan a directory of test fixtures (most common)
local result = testing.scan_directory("_testdata/curl")

-- Alias for scan_directory (archives use the same backend)
local result = testing.scan_archive("_testdata/app.tar.gz")

-- Scan as a localhost artifact
local result = testing.scan_localhost("_testdata/curl")

-- Scan a compiled binary
local result = testing.scan_binary("_testdata/binaries/curl")

-- Scan a mounted volume
local result = testing.scan_volume("_testdata/volume-root")

-- Scan a container image tarball
local result = testing.scan_container("_testdata/images/alpine.tar")
```

Each scan function:

- Creates a fresh artifact for each call (no state leaks between calls)
- Loads only the current plugin's discovery + collection pair
- Returns a result table

Result Table

```
result.findings              -- array of finding tables
```

```
result.findings[1].name      -- package name (string)
result.findings[1].version   -- package version (string)
result.findings[1].purl      -- package URL (string)
result.findings[1].component_type -- component type (string)
result.findings[1].properties    -- table<string, string>
result.findings[1].children      -- array of nested finding tables (same shape)
```

Assertions

```
-- Equality
testing.assert_equals(expected, actual, message?)
testing.assert_not_equals(expected, actual, message?)

-- Truthiness
testing.assert_true(value, message?)
testing.assert_false(value, message?)

-- Nil checks
testing.assert_nil(value, message?)
testing.assert_not_nil(value, message?)

-- String
testing.assert_contains(haystack, needle, message?)
testing.assert_matches(string, pattern, message?)

-- Tables
testing.assert_length(table, expected_length, message?)

-- Control flow
testing.fail(message)    -- immediately fail the current test
testing.skip(message)    -- skip the current test (not a failure)
```

Standard `sbomgen.*` API

The full `sbomgen.*` API (file I/O, regex, system info, logging, etc.) is available in test files, same as in production plugins. However, `sbomgen.*` functions that require an artifact (e.g., `sbomgen.read_file()`) only work inside a `testing.scan_*` callback — they are not available at the top level of a test function.

Running Tests

```
# Run all tests under a plugin directory
```

```
inspector-sbomgen plugin test --path ./my-plugins

# Filter by regex pattern
inspector-sbomgen plugin test --path ./my-plugins --run curl

# Verbose output (show each test name and result)
inspector-sbomgen plugin test --path ./my-plugins -v

# Stop on the first failing test
inspector-sbomgen plugin test --path ./my-plugins --fail-fast
```

The `--path` flag accepts a plugin root directory (containing `discovery/` and/or `collection/`) or a single ecosystem directory (auto-detected). The command exits non-zero if any test fails.

Output format

With `-v`, each test prints a `=== RUN` line and a result line (`--- PASS`, `--- FAIL`, or `--- SKIP`). Without `-v`, only failing tests print. A summary line is printed at the end:

```
=== RUN   curl/discovery/init_test/test_discovers_libcurl_header
--- PASS: curl/discovery/init_test/test_discovers_libcurl_header (0.04s)
=== RUN   curl/discovery/init_test/test_discovers_curl_binary_unix
--- PASS: curl/discovery/init_test/test_discovers_curl_binary_unix (0.04s)
=== RUN   curl/discovery/init_test/test_no_findings_for_unrelated_files
--- PASS: curl/discovery/init_test/test_no_findings_for_unrelated_files (0.04s)
ok    3 tests passed
```

Testing Plugin Helpers

The test file is loaded into the same Lua VM as the plugin's `init.lua`. Global functions defined in the plugin are callable from tests. To test helper functions, expose them as globals or in a module table:

```
-- init.lua
M = {}
function M.parse_version(raw)
    return string.match(raw, "(%d+%.%d+%.%d+)")
end

function collect(file_path)
    local ver = M.parse_version(...)
    -- ...
```

```
end
```

```
-- init_test.lua
function test_parse_version_extracts_semver()
    testing.assert_equals("1.2.3", M.parse_version("curl/1.2.3"))
end

function test_parse_version_returns_nil_for_garbage()
    testing.assert_nil(M.parse_version("not-a-version"))
end
```

Functions declared `local` in `init.lua` are not visible to the test file. This is standard Lua scoping.

Behavior and Invariants

Shared VM, shared state

Test functions within a single file share a Lua VM. A global variable set in one `test_*` function is visible to subsequent functions. If two functions define the same global, the second overwrites the first. Each test should be self-contained and not depend on state from other tests.

Non-deterministic execution order

Test functions are discovered by iterating Lua's global table, which uses hash-based ordering. **Tests are not guaranteed to run in the order they are defined.** Do not write tests that depend on execution order.

Fresh artifact per scan call

Each call to `testing.scan_directory()` (or any scan function) creates a completely new artifact. There is no state carried between scan calls within a test or across tests.

Plugin loading

Plugins are loaded once per test run, not once per test file. The test runner loads all plugins from the provided filesystem, then matches each test file to its corresponding plugin VM by phase, platform, category, and ecosystem.

Assertion behavior

When an assertion fails, the failure is recorded but the test function continues executing — subsequent assertions and statements still run. If more than one assertion fails in the same test,

the **most recent** failure is the message reported in the summary; earlier failures are overwritten. To stop a test on its first failure, return from the function after the failing assertion (or use `testing.fail()` inside a conditional).

Limitations

- **`local` functions are not testable.** Only global functions from `init.lua` are visible to the test file. Expose helpers via a module table if they need testing.
- **No `sbomgen.*` file I/O outside scan calls.** Functions like `sbomgen.read_file()` require an artifact context, which only exists inside `testing.scan_*` calls.
- **No lifecycle hooks.** There is no `before_each`, `after_each`, `setup`, or `teardown`. Each test function manages its own state.
- **No test timeout.** A test function that loops forever will hang the runner.
- **No coverage reporting.** There is no way to measure which lines of `init.lua` were exercised.
- **No benchmarks.** The test framework does not support performance benchmarks.

Developer Responsibilities

When writing a new Lua plugin

- Create `init_test.lua` next to your `init.lua`
- Create `_testdata/` with minimal fixtures that exercise your plugin's logic
- Write `test_*` functions covering: successful detection, version extraction, edge cases, and no-match scenarios
- Run tests locally before submitting

Test data guidelines

- Keep fixtures minimal — use the smallest file that exercises the behavior
- Avoid committing large binaries to `_testdata/` when a small text fixture would suffice
- Each plugin's `_testdata/` should be self-contained — no references to files outside the plugin directory. At runtime, `sbomgen.*` reads are confined to the artifact under inventory, so a plugin that depends on files elsewhere on the host will not work outside `localhost` scans.

- Use representative, non-sensitive fixtures. Sbomgen does not redact SBOM contents, so never commit real secrets or credentials to `_testdata/`.

Plugin API reference

Complete API reference for inspector-sbomgen Lua plugins. For a guide on writing plugins, see the section called “Plugin developer guide”. For testing, see the section called “Plugin testing guide”.

Overview

All runtime-provided functions are accessed through the global `sbomgen` table (file I/O, regex, logging, constants, etc.). In addition, each plugin defines a small set of top-level global functions (`discover`, `collect`, `get_scanner_name`, `subscribe_to_event`, and so on) that sbomgen calls at defined points in the plugin lifecycle. These are documented in the section called “Plugin Lifecycle Globals”.

Inside `*_test.lua` files, sbomgen additionally exposes a `testing` global that lets test authors drive the discovery→collection pipeline and make assertions. See the section called “Testing API”.

Sandbox Restrictions

Plugins run in a sandboxed Lua VM with restricted standard library access. The following Lua standard library modules are **available**:

Module	Notes
`base`	Core functions (`print`, `type`, `tostring`, `tonumber`, `pairs`, `ipairs`, `pcall`, `error`, `select`, unpack, `rawget`, `rawset`, etc.). `dofile`, `loadfile`, and `loadstring` are **removed**.
`string`	Full string manipulation (`string.match` , `string.find` , `string.format` , `string.gsub` , etc.)

Module	Notes
`table`	Full table manipulation (`table.insert` , `table.remove` , `table.sort` , `table.concat` , etc.)
`math`	Full math library (`math.floor` , `math.max`, `math.min`, etc.)
`package`	`require()` is available but restricted to modules within the plugin's own directory tree. Parent-directory traversal (`require("../shared")`) is blocked. `package.cpath` and `package.path` are cleared.

The following standard library modules are **explicitly disallowed** for security and stability:

Module	Reason
`io`	Direct filesystem access is blocked. All file operations must go through `sbomgen.*` functions, which route through the artifact interface for consistent behavior across artifact types (directory, container, volume, etc.) and confine reads to the artifact under inventory (see the section called “File access boundary”).
`os`	System-level operations (`os.execute` , `os.remove` , `os.rename` , `os.getenv` , etc.) are blocked to prevent plugins from modifying the host system.
`debug`	The debug library is blocked to prevent inspection or modification of the Lua VM internals.

Module	Reason
`coroutine`	Coroutines are not loaded.

These modules are not in the VM's allowlist and cannot be accessed by plugins.

> **Note**
>
> **Important:** All file I/O must go through `sbomgen.*` functions (e.g., `sbomgen.read_file`, `sbomgen.open_file`, `sbomgen.get_file_list`). Using `io.open` or any direct filesystem access will raise a runtime error. The `sbomgen` API ensures plugins interact with the artifact abstraction layer, which provides consistent behavior whether scanning a directory, container image, archive, or volume.

Plugin Lifecycle Globals

A plugin is a Lua file named `init.lua` that defines certain top-level global functions. These globals are **not** on the `sbomgen` table — they are functions the plugin defines for sbomgen to call. The set of valid globals differs between discovery plugins and collection plugins. For every function below, if the plugin omits it, the default shown in the table is used.

Discovery plugins

Function	Arity	Required	Default (when omitted)	Description
`discover()`	0	**Yes**	—	Returns the files this plugin has found. Return a sequential table of path strings (single-event mode) or a table keyed by event-name strings whose values are

Function	Arity	Required	Default (when omitted)	Description
				tables of paths (multi-event mode).
`get_event_name()`	0	No	`"lua:{platform}/{category}/{ecosystem}"`	Returns the event name under which files are published. Must be unique across all discovery plugins.
`get_scanner_name()`	0	No	ecosystem directory name	Returns the scanner's display name. Must be unique across all discovery plugins.
`get_scanner_description()`	0	No	`"Lua discovery plugin: {ecosystem}"`	Returns a human-readable description.
`get_scanner_groups()`	0	No	Derived from the category directory (see the developer guide)	Returns a table of scanner group strings. Use `sbomgen.groups.*` constants.

Function	Arity	Required	Default (when omitted)	Description
`get_localhost_scan_paths()`	0	No	—	Returns a table of file/directory paths to include when scanning a localhost artifact. Only consulted for `localhost` scans.
`get_fileless_events()`	0	No	—	Returns a table of event names to publish even when `discover()` found no files for them. See the section called “Collecting without a discovered file”.

Collection plugins

Function	Arity	Required	Default (when omitted)	Description
`collect(file_path)`	1	**Yes**	—	Called once per file published to the subscribed event. Parse the file and emit findings

Function	Arity	Required	Default (when omitted)	Description
				via `sbomgen.push_package()`. Returns nothing.
`subscribe_to_event()`	0	No	`"lua:{platform}/{category}/{ecosystem}"`	Returns the event name this collector subscribes to. Should match the corresponding discovery plugin's `get_event_name()`.
`get_collector_name()`	0	No	ecosystem directory name	Returns the collector's display name. Must be unique across all collection plugins.
`get_collector_description()`	0	No	`""` (empty)	Returns a human-readable description.

Function	Arity	Required	Default (when omitted)	Description
`finalize()`	0	No	—	Called once after the last file of an event has been through `collect()` . Use it to emit findings grouped across several files. See the section called “Grouping findings across files”.

File I/O

All file operations must go through the `sbomgen.*` API. Direct filesystem access via Lua's `io` library is not available (see the section called “Sandbox Restrictions”). The `sbomgen` file I/O functions route through the artifact interface, ensuring your plugin works identically whether scanning a directory on disk, a container image, a compressed archive, or a mounted volume.

File access boundary

The `sbomgen.*` file functions confine reads to the artifact under inventory. A path that resolves outside the artifact root — for example via `../` traversal — is rejected, and the call returns an error rather than reading the host filesystem. This applies to `read_file`, `open_file`, `read_dir`, `file_stat`, and the binary/hash helpers that take a path.

The exception is the `localhost` artifact type, which inventories the host itself; there the host filesystem is the artifact, so reads are not confined to a narrower root.

This boundary governs file *reads* only. It does not constrain what a plugin writes into the SBOM — see the section called “SBOM contents are not sanitized”.

`sbomgen.get_file_list()`

Returns all file paths in the artifact as a table of strings.

- **Returns:** `{string, ...}` — table of absolute file path strings
- **Performance:** This function copies every file path in the artifact into the Lua VM as a Lua string. On large artifacts (e.g., a localhost scan with 300K+ files), this alone takes several seconds. Iterating the returned table in Lua with `string.match()` adds further overhead — a full scan can take 15+ seconds. **The more files in the artifact, the slower your plugin will be.**

Note

Prefer these targeted alternatives whenever possible:

Function	Use when...
`sbomgen.find_files_by_name()`	You know the exact filename(s) to match (e.g., `"requirements.txt"` , `"curl"`)
`sbomgen.find_files_by_name_icase()`	Same as above, but case-insensitive
`sbomgen.find_files_by_suffix()`	You need to match path suffixes (e.g., `"/pom.properties"` , `"curlver.h"`)
`sbomgen.find_files_by_path_regex()`	You need full-path regex matching
`sbomgen.glob_find_files()`	You need glob-style basename matching

These functions perform matching outside the Lua VM and return only the matched paths, completing in under 1 millisecond even on 300K-file artifacts. Use `get_file_list()` only when your matching logic cannot be expressed with any of the above.
The `find_files_by_*` and `glob_find_files` helpers skip symlinks, returning only concrete files, so a symlink alias and its target are not both inventoried. `get_file_list()` returns every entry, including symlinks.

```
-- AVOID in discovery plugins when possible:
local files = sbomgen.get_file_list()
for _, f in ipairs(files) do
    if string.match(f, "pattern$") then ... end
end

-- PREFER:
local matches = sbomgen.find_files_by_name({"target-file.txt"})
```

sbomgen.read_file(path)

Reads the entire contents of a file and returns it as a string.

- **Returns:** `string, err`
- On failure: `nil, error_string`

```
local content, err = sbomgen.read_file("/app/package.json")
if err then
    sbomgen.log_error("read failed: " .. err)
    return
end
```

sbomgen.open_file(path)

Opens a file for streaming reads. Returns a FileHandle object. Use this for large files where loading the entire content into memory is impractical.

- **Returns:** `FileHandle, err`

```
local fh, err = sbomgen.open_file(path)
if err then return end
local line = fh:read_line()
while line do
    -- process line
    line = fh:read_line()
end
fh:close()
```

sbomgen.glob_find_files(pattern)

Returns files matching a Go `filepath.Match` glob pattern. The pattern is matched against the base filename. Symlinks are skipped; only concrete files are returned.

- **Returns:** `{string, ...}, err`

```
local files, err = sbomgen.glob_find_files("*.txt")
```

Use `sbomgen.get_file_list()` with `string.match` for full path pattern matching.

sbomgen.find_files_by_name(names)

Returns files whose basename (last path component) exactly matches one of the given names. The iteration and comparison happen in Go, making this significantly faster than iterating `sbomgen.get_file_list()` in Lua.

- **Parameters:** `names` — table of strings (basenames to match)
- **Returns:** `{string, ...}` — matching file paths, excluding symlinks (no error tuple)

```
local curl_bins = sbomgen.find_files_by_name({"curl", "curl.exe"})
local headers = sbomgen.find_files_by_name({"curlver.h"})
```

sbomgen.find_files_by_name_icase(names)

Returns files whose basename matches one of the given names, ignoring case. For example, `"version"` matches `VERSION`, `Version`, and `version`. Like `find_files_by_name`, matching happens outside the Lua VM.

- **Parameters:** `names` — table of strings (basenames to match, case-insensitive)
- **Returns:** `{string, ...}` — matching file paths, excluding symlinks (no error tuple)

```
local version_files = sbomgen.find_files_by_name_icase({"version"})
local war_files = sbomgen.find_files_by_name_icase({"jenkins.war"})
```

`sbomgen.find_files_by_suffix(suffixes)`

Returns files whose full (forward-slash-normalized) path ends with one of the given suffixes. Like `find_files_by_name`, matching happens outside the Lua VM.

- **Parameters:** `suffixes` — table of strings (path suffixes to match)
- **Returns:** `{string, ...}` — matching file paths, excluding symlinks (no error tuple)

```
local pom_files = sbomgen.find_files_by_suffix({"/pom.properties"})
local release_headers = sbomgen.find_files_by_suffix({"ap_release.h", "openssl.h"})
```

`sbomgen.find_files_by_path_regex(patterns)`

Returns files whose forward-slash-normalized path matches any of the given Go (RE2) regex patterns. Matching happens outside the Lua VM, which makes this efficient on large file lists.

- **Parameters:** `patterns` — table of Go regex strings
- **Returns:** `{string, ...}` — matching file paths, excluding symlinks (no error tuple)
- **Raises:** a Lua error if any pattern fails to compile

```
local configs = sbomgen.find_files_by_path_regex({"/etc/.*\\.conf$", "/opt/.*/config\
\.json$"})
```

Performance: `find_files_by_*` vs `get_file_list`

For discovery plugins, prefer `find_files_by_name`, `find_files_by_suffix`, or `find_files_by_path_regex` over iterating `get_file_list()` in Lua. On a localhost scan with 300K files, iterating the file list in Lua with `string.match()` takes ~15 seconds, while `find_files_by_name` completes in under 1 millisecond. The difference is that `get_file_list()` copies every file path into the Lua VM as a string, then Lua interprets the loop and pattern match for each one. The `find_files_by_*` functions perform the matching outside the Lua VM and return only the matched paths, avoiding both the copy and the per-path interpretation overhead.

Use `get_file_list()` only when you need custom matching logic that cannot be expressed as a basename, suffix, or regex match.

sbomgen.read_dir(path)

Lists entries in a directory.

- **Returns:** `{{name, is_dir}, ...}, err`

```
local entries, err = sbomgen.read_dir("/app/node_modules")
if err then return end
for _, e in ipairs(entries) do
    if e.is_dir then
        sbomgen.log_debug("directory: " .. e.name)
    end
end
```

sbomgen.file_stat(path)

Returns metadata about a file.

- **Returns:** `{is_regular, is_dir, size}, err`

```
local info, err = sbomgen.file_stat(path)
if err then return end
if info.is_regular and info.size > 0 then
    -- process file
end
```

sbomgen.read_zip_entry(path, entry_path)

Reads a single entry from a ZIP, JAR, or WAR archive.

- **Returns:** `string, err`

```
local manifest, err = sbomgen.read_zip_entry(
    "/app/lib/example.jar",
    "META-INF/MANIFEST.MF"
)
```

`sbomgen.search_binary(path, regex)`

Parses a file as an ELF, PE, or Mach-O binary and searches the default constant/variable section for a Go regex match.

- **Returns:** `string|nil, err` — the matched string, or nil if no match

```
local version, err = sbomgen.search_binary(path, "Version:\\s+([\\d.]+)")
if version then
    sbomgen.log_info("found version: " .. version)
end
```

`sbomgen.search_binary_all(path, regex [, n])`

Parses a file as an ELF, PE, or Mach-O binary and returns all unique first capture group matches from the default constant/variable section. Pass `n` to limit results.

- **Returns:** `{string, ...}|nil, err` — table of matched strings, or nil if no matches

```
local versions, err = sbomgen.search_binary_all(path, "version[= ]+([\\d.]+)", 5)
if versions then
    for _, v in ipairs(versions) do
        sbomgen.log_info("found: " .. v)
    end
end
```

`sbomgen.search_binary_raw(path, regex)`

Searches the entire binary file for the first regex match, not limited to a specific section. Use when section-based search (`search_binary`) is insufficient — for example, when version strings are in non-standard sections.

- **Returns:** `string|nil, err` — the matched string, or nil if no match

```
local version, err = sbomgen.search_binary_raw(path, "ProductVersion[\\x00\\s]+([\
\d.]+)")
```

FileHandle Methods

FileHandle objects are returned by `sbomgen.open_file()`.

`fh:read_line()`

Reads the next line (without the newline character). Returns `nil` at EOF.

- **Returns:** `string|nil, err`

`fh:read(n)`

Reads up to n bytes. Returns `nil` at EOF.

- **Returns:** `string|nil, err`

`fh:close()`

Closes the file handle. Always close handles when done.

Binary Utilities

`sbomgen.hash(data, algorithm)`

Returns the hex-encoded digest of an in-memory byte string under the given algorithm. Pair with `sbomgen.read_file(path)` to hash a file you've already read for parsing — preferred over `sbomgen.hash_file(path)` when you need both the bytes and the digest, since `hash` doesn't re-read the file.

- **Returns:** `string, err`
- **Algorithms:** see the section called “Component Hashes” for the list of accepted algorithm constants.

```
local data, err = sbomgen.read_file("/path/to/manifest.json")
if data then
    local sha256 = sbomgen.hash(data, sbomgen.hash_algorithms.SHA256)
    sbomgen.log_info("SHA-256: " .. sha256)
end
```

`sbomgen.hash_file(path, algorithm)`

Returns the hex-encoded digest of a file's contents under the given algorithm. Routes the read through the artifact I/O layer, so it works uniformly across directory, container, archive, volume, and localhost artifacts. Use this when the digest is the only thing you need from the file.

- **Returns:** `string, err`

```
local sha256, err = sbomgen.hash_file("/app/bin/server",
 sbomgen.hash_algorithms.SHA256)
if sha256 then
    sbomgen.log_info("SHA-256: " .. sha256)
end
```

`sbomgen.sha256(path)`

> **Important**
>
> **Deprecated.** Use `sbomgen.hash_file(path, sbomgen.hash_algorithms.SHA256)` instead. This alias is retained for backwards compatibility and will continue to work, but it will be removed in a future release. New plugins should call `hash_file` so the algorithm choice is explicit.

Equivalent to `sbomgen.hash_file(path, "SHA-256")`.

- **Returns:** `string, err`

```
local hash, err = sbomgen.sha256("/app/bin/server")
if hash then
    sbomgen.log_info("SHA-256: " .. hash)
end
```

`sbomgen.contains_bytes(path, patterns)`

Checks whether a file contains each of the given byte patterns. Returns a table of booleans in the same order as the input patterns.

- **Returns:** {bool, ...}, err

```
local results, err = sbomgen.contains_bytes(path, {
    "\xff Go buildinf:",   -- Go build identifier
    "/rustc/",             -- Rust build identifier
})
if results then
    local is_go = results[1]
    local is_rust = results[2]
end
```

sbomgen.get_pe_version_info(path)

Parses Windows PE version resources from a binary file. Returns a table with version fields, or nil, err if the file is not a PE binary or has no version resource.

- **Returns:** {product_version, file_version, string_table}, err

The product_version and file_version fields come from the PE FixedFileInfo structure, formatted as "major.minor.build.revision". The string_table field is a nested table keyed by **locale code** (e.g., "040904B0" for US English Unicode). Each locale maps to a table of name/value pairs drawn from the PE StringFileInfo (ProductVersion, ProductName, FileDescription, etc.). A PE binary may expose one or more locales.

```
local info, err = sbomgen.get_pe_version_info(file_path)
if err then return end

-- Fixed version fields (always flat)
local product_ver = info.product_version  -- e.g. "25.1.0.0"
local file_ver    = info.file_version     -- e.g. "25.1.0.0"

-- String table — iterate locales, or address a known locale by key
for locale, fields in pairs(info.string_table or {}) do
    sbomgen.log_info(string.format("%s ProductName=%s", locale, fields.ProductName or
 ""))
end

-- US English Unicode is the most common locale for PE files
local us = (info.string_table or {})["040904B0"]
if us then
```

```
    local display_ver = us.ProductVersion  -- e.g. "25.01"
    local name        = us.ProductName     -- e.g. "7-Zip"
end
```

sbomgen.parse_product_version(path)

Convenience wrapper that returns just the product version string from a PE binary's FixedFileInfo. Equivalent to calling `get_pe_version_info(path)` and reading `product_version`.

- **Returns:** `string, err`

```
local version, err = sbomgen.parse_product_version(file_path)
if version then
    sbomgen.log_info("product version: " .. version)
end
```

sbomgen.parse_file_version(path)

Convenience wrapper that returns just the file version string from a PE binary's FixedFileInfo. Equivalent to calling `get_pe_version_info(path)` and reading `file_version`.

- **Returns:** `string, err`

```
local version, err = sbomgen.parse_file_version(file_path)
if version then
    sbomgen.log_info("file version: " .. version)
end
```

Package Output

sbomgen.push_package(pkg)

Pushes a package finding into the SBOM. Only available in collection plugins.

The pkg table supports the following fields:

Field	Type	Required	Description
`name`	string	Yes	Package name, used as the component name in the SBOM
`purl_name`	string	No	PURL name component; defaults to `name`. Set it when the display name is not a good package identifier, e.g. `name = "Visual Studio Code"` with `purl_name = "visualstudiocode"`
`version`	string	No	Resolved version string
`namespace`	string	No	PURL namespace (e.g., `"curl"`, `"wordpress/plugin"`)
`purl_type`	string	Yes	PURL type (e.g., `"pypi"`, `"npm"`, `"cargo"`, `"deb"`, `"generic"`), or `sbomgen.purl_types.NONE` for components that carry no package URL (see the section called “PURL Type Constants”)

Field	Type	Required	Description
component_type	string	Yes	CycloneDX component type; use sbomgen.component_types.* constants (e.g., sbomgen.component_types.LIBRARY)
qualifiers	table	No	PURL qualifiers as key-value pairs (appear in the package URL)
properties	table	No	CycloneDX component properties as key-value pairs (see the section called "CycloneDX Properties")
hashes	table	No	Component hashes keyed by algorithm name; see the section called "Component Hashes"
licenses	table	No	List of license texts, normalized to an SPDX expression; see the section called "Component Licenses"

Field	Type	Required	Description
children	table	No	Nested child packages, each with the same shape as pkg (required fields are validated recursively)
source_paths	table	No	List of provenance paths recorded as source_path properties. Defaults to the file collect() was invoked with. Set it to point at a directory, at several paths, or to give a child its own path
provenance	string	No	Where the package was read from; use sbomgen.provenance.* constants. Defaults to sbomgen.provenance.FILE. See the section called "Package Provenance"

```
sbomgen.push_package({
    name = "requests",
    version = "2.28.1",
    purl_type = "pypi",
    component_type = sbomgen.component_types.LIBRARY,
```

```
    qualifiers = { example_qualifier = "example_qualifier_value" },
    properties = {
        -- Use your own namespace; amazon:inspector:* is reserved for Amazon Inspector.
        ["acme:example:extra_field"] = "example_value",
    },
    hashes = {
        [sbomgen.hash_algorithms.SHA256] =
 "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824",
    },
})
```

Package Provenance

Most packages are found by reading a file, and the SBOM records that file as the component's provenance. Some are not: a Windows application may only be described by a registry key, with nothing on disk naming its version.

The `provenance` field states which case applies.

Value	Provenance recorded
`sbomgen.provenance.FILE` (default)	`source_file_scanner` plus a `source_path` per entry in `source_paths`, defaulting to the file `collect()` received
`sbomgen.provenance.REGISTRY`	`source_registry_scanner`, and no `source_path`, because no file describes the package

An unrecognized `provenance` drops the package with a warning rather than silently falling back to file provenance.

```
sbomgen.push_package({
    name = "Example App",
    purl_name = "exampleapp",
    version = "1.2.3",
    namespace = "vendor",
    purl_type = "generic",
    component_type = sbomgen.component_types.APPLICATION,
```

```
    provenance = sbomgen.provenance.REGISTRY,
    properties = {
        [sbomgen.properties.WINDOWS_REGISTRY_KEY] = registry_key,
    },
})
```

Collecting without a discovered file

A collector normally runs once per discovered file, so an event with no files never reaches it. A discovery plugin can list events that should fire anyway, for collectors that read the registry or another non-file source:

```
function get_fileless_events()
    return { "FoundExampleApp" }
end
```

`collect()` then runs exactly once for that event with an empty `file_path`. Events not listed stay silent when discovery finds nothing, so returning an empty table from `discover()` keeps its usual meaning.

Grouping findings across files

`collect()` sees one file at a time, so a plugin that pushes a package per call emits a separate component per file. When several files describe one installation of the same thing, that is usually wrong: four copies of a browser executable at the same version are one component with four source paths, not four components.

Accumulate across calls and emit from `finalize()`, which runs once after the last file of the event:

```
local pending = {}

function collect(file_path)
    local version = sbomgen.parse_product_version(file_path)
    if not version then
        return
    end
    pending[version] = pending[version] or {}
    table.insert(pending[version], dir_name(file_path))
end
```

```
function finalize()
    local batch = pending
    pending = {}
    for version, paths in pairs(batch) do
        sbomgen.push_package({
            name = "Example App",
            version = version,
            purl_type = "generic",
            component_type = sbomgen.component_types.APPLICATION,
            source_paths = paths,
        })
    end
end
```

Clear the accumulator before emitting, as above, not after. A plugin's Lua state persists for the whole scan, so anything left behind leaks into the next event, and clearing first means an error partway through emitting cannot strand entries.

Component Hashes

The optional `hashes` field on `sbomgen.push_package()` records integrity digests for a component. Entries are keyed by algorithm name and serialized into the CycloneDX `components[].hashes` array, matching the schema Amazon Inspector expects.

Supported algorithms

Use the constants under `sbomgen.hash_algorithms` so the algorithm name passed to `sbomgen.hash()` / `sbomgen.hash_file()` is the same string accepted by `push_package({ hashes = ... })`:

Constant	Value	Hex digest length
`sbomgen.hash_algorithms.SHA1`	`"SHA-1"`	40
`sbomgen.hash_algorithms.SHA256`	`"SHA-256"`	64

Validation rules

`push_package()` validates `hashes` before emitting a finding. A package whose hashes fail validation is dropped and a warning is logged. Validation:

- Algorithm names must match an entry in `sbomgen.hash_algorithms` (case sensitive, exactly `"SHA-1"` or `"SHA-256"`).
- Values must be non-empty, lower-case hexadecimal digits.
- Values must be the correct length for the algorithm (40 chars for SHA-1, 64 chars for SHA-256).
- Validation is recursive: a malformed hash inside `children[].hashes` rejects the whole package.

Example: hashing a manifest and attaching the digest

```
function collect(file_path)
    local data, err = sbomgen.read_file(file_path)
    if err then return end

    local sha256 = sbomgen.hash(data, sbomgen.hash_algorithms.SHA256)

    sbomgen.push_package({
        name = "skill-manifest",
        version = "1.0.0",
        purl_type = "generic",
        component_type = sbomgen.component_types.DATA,
        hashes = {
            [sbomgen.hash_algorithms.SHA256] = sha256,
        },
    })
end
```

When the digest is the only thing you need from the file, prefer `sbomgen.hash_file(path, algo)` over reading the file twice — it routes the read through the artifact I/O layer in a single pass.

Component Licenses

The optional `licenses` field on `sbomgen.push_package()` records the licenses a package declares. Pass the license texts as you read them; sbomgen normalizes them into a single SPDX license expression and serializes it into the CycloneDX `components[].licenses` array.

```
sbomgen.push_package({
    name = "docker",
    version = "29.7.1",
    purl_type = "brew",
    component_type = sbomgen.component_types.APPLICATION,
    licenses = {"Apache-2.0"},
})
```

Produces:

```
{
  "type": "application",
  "name": "docker",
  "version": "29.7.1",
  "purl": "pkg:brew/docker@29.7.1",
  "licenses": [ { "expression": "Apache-2.0" } ]
}
```

Normalization

Values do not have to be valid SPDX identifiers. sbomgen normalizes them, so a plugin can pass through whatever a package manifest declares:

Input	Resulting expression
`{"Apache-2.0"}`	`Apache-2.0`
`{"GPL3+"}`	`GPL-3.0-or-later`
`{"MIT", "Apache-2.0"}`	`Apache-2.0 AND MIT`
`{"Apache-2.0", "Apache-2.0"}`	`Apache-2.0` (deduplicated)
`{"Custom Vendor License"}`	`LicenseRef-Custom-Vendor-License`

Multiple entries are combined with `AND`. Text that is not a recognized SPDX identifier becomes a `LicenseRef-` expression rather than being dropped. Empty strings are ignored.

Licenses are only emitted when the user asks for them

Licenses appear in the SBOM only when the scan was run with `--collect-licenses`. Without that flag the field is omitted, exactly as it is for sbomgen's built-in collectors.

Reading license data can be expensive, for example when it means opening a separate license file per package. Check `sbomgen.should_collect_licenses()` and skip that work when it is not needed:

```
function collect(file_path)
    local licenses = nil
    if sbomgen.should_collect_licenses() then
        licenses = read_license_file(file_path)
    end

    sbomgen.push_package({
        name = "example",
        version = "1.0.0",
        purl_type = "generic",
        component_type = sbomgen.component_types.APPLICATION,
        licenses = licenses,
    })
end
```

Passing `licenses` unconditionally is also fine. The field is simply ignored when license collection is disabled.

Child packages

A package in `children` carries its own `licenses`. Licenses are not inherited from the parent, because a nested package may be licensed differently than the component it ships under.

CycloneDX Properties

CycloneDX properties are key-value metadata attached to a component in the SBOM. They are distinct from PURL qualifiers:

- **`qualifiers`** — PURL qualifiers. These become part of the package URL string (e.g., `pkg:deb/debian/curl@7.88.1?arch=amd64`). Some PURL qualifiers carry semantic meaning to

Amazon Inspector and influence vulnerability identification. See What is a package URL? for Inspector's per-type conventions.

- **`properties`** — CycloneDX component properties. These appear in the SBOM's `components[].properties` array and do not change how the component is identified.

Reserved namespaces

The `amazon:inspector:*` family of CycloneDX property namespaces is reserved for Amazon Inspector:

- `amazon:inspector:sbom_generator:*` — used by sbomgen and its built-in scanners.
- `amazon:inspector:sbom_scanner:*` — used by the Amazon Inspector Scan API.

Plugin-defined properties must not use these namespaces. Writing into a reserved namespace can shadow or conflict with values Inspector relies on, and the resulting SBOM may be interpreted incorrectly during vulnerability identification. See Using CycloneDX namespaces with Amazon Inspector for the full list of reserved keys.

Key naming rules

Property keys passed to `sbomgen.push_package()` are processed as follows:

Input key	Resulting key in SBOM	Recommended for custom plugins?
Contains `:` (e.g., `acme:my_plugin:field`)	Used verbatim	**Yes** — place every plugin-defined property in your own namespace
No `:` (e.g., `field`)	Auto-prefixed to `amazon:inspector:sbom_generator:field`	**No** — this writes into a reserved namespace

Always include at least one colon in property keys you define. Use a namespace unique to your organization or plugin (for example `acme:python-pip:*`):

```
properties = {
```

```
    -- Custom namespace — safe to use (recommended)
    ["acme:python-pip:manifest_path"] = file_path,
    ["acme:python-pip:pinned"]        = "true",

    -- Fully-qualified key outside amazon:inspector:* — also fine
    ["my:custom:namespace:key"] = "value",

    -- No colon: avoid — ends up as "amazon:inspector:sbom_generator:custom_field"
    -- custom_field = "value",
}
```

Properties set by sbomgen

Sbomgen may attach properties of its own to every component it emits. These values come from the reserved `amazon:inspector:sbom_generator:*` namespace and should not be produced by plugins. Observed runtime behavior:

- `source_path` is always added by sbomgen.
- `source_file_scanner` and `source_package_collector` are added when `--enable-debug-props` is enabled.

The full taxonomy of reserved keys is maintained in the Amazon Inspector user guide: Using CycloneDX namespaces with Amazon Inspector.

SBOM contents are not sanitized

Sbomgen does not inspect or filter the data a plugin emits. Component names, versions, PURLs, hashes, and property values are written to the SBOM as provided. Sbomgen does not detect or redact secrets, credentials, tokens, or other sensitive data — if a plugin places such a value into a finding, it appears in the output SBOM and travels wherever that SBOM is published.

You are responsible for what your plugins write. Only emit data derived from the artifact you intend to inventory, and treat the SBOM as a shareable artifact when deciding what to include.

Property Constants

Built-in property key constants are available via `sbomgen.properties`. Every constant below resolves to a key inside the reserved `amazon:inspector:sbom_generator:*` namespace. These constants exist so that sbomgen's built-in scanners emit consistent property keys. **They are not extension points for custom plugins** — using them in a custom plugin writes into a

reserved namespace, which can shadow values Inspector relies on. See the section called “Reserved namespaces” above.

Custom plugin authors should define properties under their own namespace (for example `acme:my_plugin:*`) rather than reusing these constants.

Constant	Resolved Value
`sbomgen.properties.NAMESPACE`	`amazon:inspector:sbom_generator:`
`sbomgen.properties.VENDOR`	`amazon:inspector:sbom_generator:vendor`
`sbomgen.properties.FILE_SIZE_BYTES`	`amazon:inspector:sbom_generator:file_size_bytes`
`sbomgen.properties.KERNEL_COMPONENT`	`amazon:inspector:sbom_generator:kernel_component`
`sbomgen.properties.RUNNING_KERNEL`	`amazon:inspector:sbom_generator:running_kernel`
`sbomgen.properties.UNRESOLVED_VERSION`	`amazon:inspector:sbom_generator:unresolved_version`
`sbomgen.properties.TRANSITIVE_DEPENDENCY`	`amazon:inspector:sbom_generator:experimental:transitive_dependency`
`sbomgen.properties.GO_REPLACE_DIRECTIVE`	`amazon:inspector:sbom_generator:replaced_by`
`sbomgen.properties.DUPLICATE_PACKAGE`	`amazon:inspector:sbom_generator:is_duplicate_package`
`sbomgen.properties.DUPLICATE_PURL`	`amazon:inspector:sbom_generator:duplicate_purl`
`sbomgen.properties.DOCKERFILE_CHECK`	`amazon:inspector:sbom_generator:dockerfile_finding`

Constant	Resolved Value
sbomgen.properties.CERTIFICATE_FINDING	amazon:inspector:sbom_generator:certificate_finding
sbomgen.properties.CERTIFICATE_SUBJECT_NAME	amazon:inspector:sbom_generator:certificate:subject_name
sbomgen.properties.CERTIFICATE_ISSUER_NAME	amazon:inspector:sbom_generator:certificate:issuer_name
sbomgen.properties.CERTIFICATE_SIGNATURE_ALGORITHM	amazon:inspector:sbom_generator:certificate:signature_algorithm
sbomgen.properties.CERTIFICATE_NOT_VALID_BEFORE	amazon:inspector:sbom_generator:certificate:not_valid_before
sbomgen.properties.CERTIFICATE_NOT_VALID_AFTER	amazon:inspector:sbom_generator:certificate:not_valid_after
sbomgen.properties.WINDOWS_REGISTRY_KEY	amazon:inspector:sbom_generator:registry_key
sbomgen.properties.SUBSCRIPTION_ENABLED	amazon:inspector:sbom_generator:subscription:enabled
sbomgen.properties.SUBSCRIPTION_NAME	amazon:inspector:sbom_generator:subscription:name
sbomgen.properties.SUBSCRIPTION_LOCKED_VERSION	amazon:inspector:sbom_generator:subscription:locked_version
sbomgen.properties.OPENSSL_FULL_VERSION	amazon:inspector:sbom_generator:openssl:full_version

Constant	Resolved Value
`sbomgen.properties.HARDENED_IMAGE_VENDOR`	`amazon:inspector:sbom_generator:hardened_image:vendor`

Scanner Groups

Discovery plugins must declare their scanner groups via `get_scanner_groups()`. Groups categorize scanners and allow users to selectively enable or disable categories. Constants are available via `sbomgen.groups`:

Constant	Value	Description
`sbomgen.groups.OS`	`"os"`	OS package managers (dpkg, rpm, etc.)
`sbomgen.groups.PROGRAMMING_LANGUAGE`	`"programming-language-packages"`	Language package managers (pip, npm, maven, etc.)
`sbomgen.groups.BINARY`	`"binary"`	Compiled binary analysis (Go, Rust)
`sbomgen.groups.PACKAGE_COLLECTOR`	`"pkg-scanner"`	General package collection
`sbomgen.groups.EXTRA_ECOSYSTEMS`	`"extra-ecosystems"`	Additional ecosystems (curl, nginx, etc.)
`sbomgen.groups.CERTIFICATE`	`"certificate"`	Certificate scanning
`sbomgen.groups.CUSTOM`	`"custom"`	Automatically added to all custom plugins loaded via `--plugin-dir`
`sbomgen.groups.MACHINE_LEARNING`	`"machine-learning"`	Machine learning model detection

Example:

```
function get_scanner_groups()
    return {sbomgen.groups.PROGRAMMING_LANGUAGE, sbomgen.groups.PACKAGE_COLLECTOR}
end
```

Component Type Constants

The `component_type` field in `push_package()` must be one of the CycloneDX 1.5 component types. Constants are available via `sbomgen.component_types`:

Constant	Value
`sbomgen.component_types.APPLICATION`	`"application"`
`sbomgen.component_types.FRAMEWORK`	`"framework"`
`sbomgen.component_types.LIBRARY`	`"library"`
`sbomgen.component_types.CONTAINER`	`"container"`
`sbomgen.component_types.PLATFORM`	`"platform"`
`sbomgen.component_types.OPERATING_SYSTEM`	`"operating-system"`
`sbomgen.component_types.DEVICE`	`"device"`
`sbomgen.component_types.DEVICE_DRIVER`	`"device-driver"`
`sbomgen.component_types.FIRMWARE`	`"firmware"`
`sbomgen.component_types.FILE`	`"file"`
`sbomgen.component_types.MACHINE_LEARNING_MODEL`	`"machine-learning-model"`

Constant	Value
`sbomgen.component_types.DATA`	`"data"`

Example:

```
sbomgen.push_package({
    name = "requests",
    version = "2.28.1",
    purl_type = "pypi",
    component_type = sbomgen.component_types.LIBRARY,
})
```

PURL Type Constants

`purl_type` is normally a free-form type string from the PURL specification. The one enumerated value is available via `sbomgen.purl_types`:

Constant	Value	Description
`sbomgen.purl_types.NONE`	`"none"`	Emit the component without a package URL

Use `NONE` when no PURL type meaningfully identifies the component, such as a machine learning model file discovered on disk. The resulting SBOM component has no `purl` field.

`purl_type` remains required. Omitting it, or misspelling the field name, drops the package with a warning rather than silently producing a component without a package URL.

```
sbomgen.push_package({
    name = "model.gguf",
    purl_type = sbomgen.purl_types.NONE,
    component_type = sbomgen.component_types.MACHINE_LEARNING_MODEL,
})
```

Platform Constants

Constants for comparing against `sbomgen.get_platform()`. Available via `sbomgen.platform`:

Constant	Value
`sbomgen.platform.LINUX`	`"linux"`
`sbomgen.platform.WINDOWS`	`"windows"`
`sbomgen.platform.DARWIN`	`"darwin"`

Example:

```
if sbomgen.get_platform() == sbomgen.platform.WINDOWS then
    -- Windows-specific logic
end
```

Hash Algorithm Constants

Constants for the algorithm parameter of `sbomgen.hash()`, `sbomgen.hash_file()`, and the `hashes` field of `sbomgen.push_package()`. The string values match the CycloneDX hash algorithm names so the same constant flows through the entire hashing path without translation.

Constant	Value
`sbomgen.hash_algorithms.SHA1`	`"SHA-1"`
`sbomgen.hash_algorithms.SHA256`	`"SHA-256"`

Example:

```
local digest = sbomgen.hash_file(path, sbomgen.hash_algorithms.SHA256)
sbomgen.push_package({
    name = "example",
    purl_type = "generic",
    component_type = sbomgen.component_types.LIBRARY,
    hashes = { [sbomgen.hash_algorithms.SHA256] = digest },
})
```

Artifact Info

`sbomgen.get_platform()`

Returns the runtime platform string (e.g., `"linux"`, `"windows"`, `"darwin"`).

`sbomgen.get_artifact_type()`

Returns the type of artifact being scanned (e.g., `"directory"`, `"archive"`).

`sbomgen.should_collect_licenses()`

Returns `true` if the user enabled license collection via `--collect-licenses`.

`sbomgen.get_env_vars()`

Returns environment variables from the artifact as a table of `{key, value}` entries.

```
local env_vars = sbomgen.get_env_vars()
for _, env in ipairs(env_vars) do
    if env.key == "NODE_ENV" then
        sbomgen.log_info("Node environment: " .. env.value)
    end
end
```

`sbomgen.get_system_drive()`

Returns the system drive letter (e.g., `"C:"`) from the artifact's environment. Reads the `SystemDrive` environment variable, defaulting to `"C:"` if not set. This is the Lua equivalent of `strutils.GetSystemDriverLetter()`.

```
local drive = sbomgen.get_system_drive()
local program_files = drive .. "/Program Files/"
```

`sbomgen.resolve_glob_paths(patterns)`

Expands filesystem glob patterns against the host filesystem. Localhost-only: returns `nil` plus an error on other artifact types.

```
function get_localhost_scan_paths()
    return sbomgen.resolve_glob_paths({
```

```
        "/home/*/.cache/huggingface/hub",
        "/Users/*/.cache/huggingface/hub",
        "C:/Users/*/.cache/huggingface/hub",
    })
end
```

Behavior:

- Pattern syntax follows Go's `filepath.Match`: `*`, `?`, `[abc]`, `[a-z]`.
- Input patterns and output paths are normalized: redundant separators (a//b), dot segments (a/./b), and trailing separators (a/b/) are collapsed.
- Output is deduplicated; the first occurrence of a path wins. Input pattern order is preserved across the result.
- Patterns that match nothing return no entries. Empty-string patterns are silently skipped. Malformed patterns (e.g. mismatched brackets) emit a warning and are skipped.

Cross-platform path separators:

- **Use forward slashes (/) for all paths.** Forward slashes work on Linux, macOS, and Windows; Go's filepath logic translates them to the native separator on Windows.
- **Backslash separators only work on Windows.** On Linux and macOS, \ is a literal filename character, not a path separator — a pattern like `"C:\\Users\\*"` matches nothing on POSIX systems.
- **Avoid literal Windows-style paths in Lua strings.** A Lua string like `"C:\Users"` is interpreted as `C:<form-feed>sers` because \U is not a valid Lua escape (and \f, \n, \t etc. are), so the pattern silently fails. Either use forward slashes, escaped backslashes (`"C:\\Users"`), or a long-bracket raw string (`[[C:\Users]]`).

`sbomgen.get_home_dirs()`

Returns the user home directory roots for the artifact, as a table of paths. Localhost enumerates the real host filesystem; container and volume artifacts enumerate their own (rootfs or mounted) filesystem; all other artifact types return an empty table. Paths are forward-slash normalized, de-duplicated, and sorted.

This is the artifact-aware way to locate per-user directories (such as model caches or tool configuration) without hardcoding /home/* or /Users/*. It includes root's home directory and

skips well-known non-user directories (built-in profiles such as `Public` and `Default` on Windows, `Shared` on macOS, and `lost+found` on Linux).

```
function get_localhost_scan_paths()
    local patterns = {}
    for _, home in ipairs(sbomgen.get_home_dirs()) do
        table.insert(patterns, home .. "/.cache/huggingface/hub")
    end
    return sbomgen.resolve_glob_paths(patterns)
end
```

The composition above is localhost-only, because `resolve_glob_paths` errors on non-localhost artifacts. For container and volume scans, match the returned home roots against the artifact's file list (for example, with `sbomgen.find_files_by_path_regex`) instead.

Behavior:

- Defined for `localhost`, `container`, and `volume` artifacts. Other artifact types return an empty table, both because a per-user home concept does not apply to a bare directory or binary scan and because reading absolute host paths there would escape the scan root.
- On localhost, returns absolute host paths (for example, `/home/alice`, `/root`). On a container or volume, returns paths as they appear inside that artifact's filesystem.
- Enumeration reads through the artifact interface, so container and volume homes resolve against their own filesystem rather than the host.
- Symlinked entries are not followed; only real directories are returned.
- On Windows, the `Users` directory is located on the scanning host's `SystemDrive`, so a Windows container or volume is enumerated under the host's drive letter. This is a known limitation shared with `sbomgen.get_system_drive`.

System Info

These functions return metadata about the artifact's operating system and hardware. Values may be empty strings if the information is not available (e.g., when scanning a directory without OS metadata).

Function	Returns
`sbomgen.get_os_name()`	OS name (e.g., `"Ubuntu"`, `"Alpine Linux"`)
`sbomgen.get_os_version()`	OS version (e.g., `"22.04"`, `"3.18"`)
`sbomgen.get_os_codename()`	OS codename (e.g., `"jammy"`, `"bookworm"`)
`sbomgen.get_os_id()`	OS identifier (e.g., `"ubuntu"`, `"alpine"`)
`sbomgen.get_kernel_name()`	Kernel name (e.g., `"Linux"`)
`sbomgen.get_kernel_version()`	Kernel version string
`sbomgen.get_cpu_arch()`	CPU architecture (e.g., `"x86_64"`, `"aarch64"`)
`sbomgen.get_hostname()`	Hostname of the system

Regular Expressions

Lua's built-in patterns lack features like alternation (`|`), quantifier ranges (`{n,}`), and lookahead. To close this gap, sbomgen exposes Go's `regexp` package directly. These functions use Go regex syntax (RE2), not Lua patterns.

`sbomgen.regex_find(str, pattern)`

Returns the first match of a Go regex pattern, or `nil` if no match.

- **Returns:** `string|nil, err`

```
local version = sbomgen.regex_find(content, "\\d+\\.\\d+\\.\\d+")
```

`sbomgen.regex_match(str, pattern)`

Returns capture groups from the first match. Index 1 is the full match, 2+ are capture groups.

- **Returns:** `{string, ...}|nil, err`

```
local groups = sbomgen.regex_match(content, "(MySQL|MariaDB) (\\d+)\\.(\\d+)\\.(\\d+)")
if groups then
    local db_type = groups[2]   -- "MySQL" or "MariaDB"
    local major   = groups[3]
end
```

`sbomgen.regex_find_all(str, pattern [, n])`

Returns all non-overlapping matches. Pass n to limit results (default: all).

- **Returns:** `{string, ...}|nil, err`

```
local versions = sbomgen.regex_find_all(content, "\\d+\\.\\d+\\.\\d+")
```

`sbomgen.regex_replace(str, pattern, replacement)`

Replaces all matches. The replacement string can use $1, $2, etc. for capture group references.

- **Returns:** `string, err`

```
local cleaned = sbomgen.regex_replace(raw_version, "(1[6-9]\\d{8,}|
buildkitsandbox.*)$", "")
```

When to use regex vs Lua patterns

Use Lua's built-in `string.match`/`string.find` for simple patterns — they're faster and don't require escaping backslashes. Use `sbomgen.regex_*` when you need:

- Alternation: `(foo|bar)`
- Quantifier ranges: `\d{8,}`
- Complex character classes not expressible in Lua patterns

Structured Parsing

Sbomgen exposes lightweight helpers for decoding structured text formats directly into Lua tables.

`sbomgen.json_decode(str)`

Parses a JSON string into a Lua table.

- **Returns:** `table|nil, err`

```
local doc, err = sbomgen.json_decode('{"name":"requests","version":"2.28.1"}')
if err then return end
sbomgen.log_info(doc.name)
```

`sbomgen.xml_decode(str)`

Parses an XML string into a Lua table.

- **Returns:** `table|nil, err`

XML values use the following shape:

- `_name` — element name
- `_attr` — attribute table, when present
- `_text` — trimmed text content, when present
- numeric indices `1..n` — child elements

```
local doc, err = sbomgen.xml_decode('<package id="Newtonsoft.Json" version="13.0.3" /
>')
if err then return end
sbomgen.log_info(doc._attr.id)
```

Windows Registry

These functions provide read-only access to the Windows registry. On non-Windows artifacts, `registry_open_key` returns an error. The registry accessor is initialized lazily on first use and supports both live Windows API access (localhost scans on Windows) and file-based REGF hive parsing (container/volume scans).

`sbomgen.registry_open_key(path)`

Opens a registry key. Returns a key handle that must be closed with `registry_close`.

- **Returns:** `key, err`

```
local key, err = sbomgen.registry_open_key("SOFTWARE\\Microsoft\\Windows\
\CurrentVersion\\Uninstall\\7-Zip")
if err then return end
-- use key...
sbomgen.registry_close(key)
```

`sbomgen.registry_get_string(key, value_name)`

Reads a string value from an open registry key.

- **Returns:** `string, err`

```
local version, err = sbomgen.registry_get_string(key, "DisplayVersion")
```

`sbomgen.registry_get_integer(key, value_name)`

Reads an integer value from an open registry key.

- **Returns:** `number, err`

`sbomgen.registry_get_strings(key, value_name)`

Reads a multi-string (REG_MULTI_SZ) value from an open registry key. Returns a table of strings.

- **Returns:** `{string, ...}, err`

```
local paths, err = sbomgen.registry_get_strings(key, "DependsOnService")
if paths then
    for _, p in ipairs(paths) do
        sbomgen.log_info("depends on: " .. p)
    end
end
```

`sbomgen.registry_get_subkeys(key)`

Returns all subkey names under an open registry key.

- **Returns:** `{string, ...}, err`

```
local subkeys, err = sbomgen.registry_get_subkeys(key)
for _, name in ipairs(subkeys) do
    local subkey, err = sbomgen.registry_open_key(parent_path .. "\\" .. name)
    -- ...
end
```

`sbomgen.registry_close(key)`

Closes a registry key handle. Key handles are also closed automatically by the garbage collector, but explicit closing is recommended.

Logging

Log messages are written to sbomgen's console output. Every message emitted by a plugin is automatically prefixed with the plugin's source label and ecosystem, for example:

```
[custom:python-pip] Parsing requirements.txt
```

`log_info`, `log_warn`, and `log_error` always print. `log_debug` only prints when sbomgen is invoked with `--verbose`.

Function	Level	Visible by default?
`sbomgen.log_debug(message)`	DEBUG	No — requires `--verbose`
`sbomgen.log_info(message)`	INFO	Yes
`sbomgen.log_warn(message)`	WARN	Yes

Function	Level	Visible by default?
`sbomgen.log_error(message)`	ERROR	Yes

Use `string.format` for formatted messages:

```
sbomgen.log_info(string.format("found %d packages in %s", count, file_path))
```

Debugging Functions

`sbomgen.breakpoint(message)`

Prints `message` to stderr and blocks execution until the user presses Enter. If `message` is omitted, prints a default message.

Use this as a crude debugger by placing breakpoints at key points in your plugin and running with `--verbose` to see surrounding log output.

```
sbomgen.log_info("state: " .. some_variable)
sbomgen.breakpoint("paused after state dump — press Enter to continue")
```

Testing API

Functions under the global `testing` table are only available inside plugin test files (`*_test.lua`), loaded by `inspector-sbomgen plugin test`. They are not available at runtime in discovery or collection plugins. The full `sbomgen.*` API is also available inside test files, but `sbomgen.*` functions that require an artifact (for example `sbomgen.read_file()`) only produce meaningful results when called from inside a scan. For a narrative guide, see the the section called “Plugin testing guide”.

Scan functions

Each scan function creates an artifact of the given kind, runs the current plugin's discovery→collection pipeline against it, and returns the resulting findings. The `path` argument is resolved relative to the test file's directory.

Function	Artifact kind
`testing.scan_directory(path)`	Directory
`testing.scan_archive(path)`	Directory (alias of `scan_directory`)
`testing.scan_localhost(path)`	Localhost
`testing.scan_binary(path)`	Binary
`testing.scan_volume(path)`	Volume
`testing.scan_container(path)`	Container

All six return a result table with the shape below.

Result shape

Each finding table projects only the fields listed below. In particular, `namespace` and `purl_type` are not projected separately — they are incorporated into the full `purl` string.

```
local result = testing.scan_directory("_testdata/example")
-- result.findings                        -- array of finding tables
-- result.findings[i].name                -- string
-- result.findings[i].version             -- string
-- result.findings[i].component_type      -- string
-- result.findings[i].purl                -- string (the full Package URL, or "" if
 none)
-- result.findings[i].properties          -- table<string, string>
-- result.findings[i].children            -- array of finding tables (same shape,
 recursive)
```

Assertions

Function	Signature	Description
`testing.assert_equals`	`(expected: any, actual: any, message?: string)`	Fails if `tostring(expected) ~= tostring(actual)` .

Function	Signature	Description
`testing.assert_not_equals`	`(expected: any, actual: any, message?: string)`	Fails if `tostring(expected) == tostring(actual)` .
`testing.assert_true`	`(value: any, message?: string)`	Fails if `value` is `false` or `nil`.
`testing.assert_false`	`(value: any, message?: string)`	Fails if `value` is not `false` and not `nil`.
`testing.assert_nil`	`(value: any, message?: string)`	Fails if `value` is not `nil`.
`testing.assert_not_nil`	`(value: any, message?: string)`	Fails if `value` is `nil`.
`testing.assert_contains`	`(haystack: string, needle: string, message?: string)`	Fails if `haystack` does not contain `needle` (substring match).
`testing.assert_matches`	`(str: string, pattern: string, message?: string)`	Fails if `str` does not match the given **Go (RE2)** regex.
`testing.assert_length`	`(tbl: table, expected: integer, message?: string)`	Fails if `#tbl` does not equal `expected`.

Control flow

Function	Signature	Description
`testing.fail`	`(message: string)`	Fails the current test immediately with the given message.

Function	Signature	Description
`testing.skip`	`(message: string)`	Skips the current test. The result is reported as skipped, not failed.

Test discovery

Any global Lua function whose name starts with `test_` in a file matching `*_test.lua` is treated as a test. The test file must sit next to an `init.lua` at the normal `{phase}/{platform}/{category}/{ecosystem}/` depth. Fixture data goes in `_testdata/` next to the test file — the runner does not descend into `_testdata/` when searching for test files.

Error Handling

API functions that can fail return two values: `value, err`. On success, `err` is `nil`. On failure, the first value is `nil` and `err` is an error string.

```
local content, err = sbomgen.read_file(path)
if err then
    sbomgen.log_error("failed to read " .. path .. ": " .. err)
    return
end
-- content is safe to use here
```

If a plugin raises an unhandled Lua error, sbomgen logs a warning and continues with the next file or plugin. Other plugins are not affected.

Integrating Amazon Inspector scans into your CI/CD pipeline

The Amazon Inspector CI/CD integration utilizes the Amazon Inspector SBOM Generator and Amazon Inspector Scan API to produce vulnerability reports for container images. The Amazon Inspector SBOM Generator creates a software bill of materials (SBOM) for archives, container images, directories, local systems, and compiled Go and Rust binaries. The Amazon Inspector Scan API scans the SBOM to create a report with details about detected vulnerabilities. You can integrate Amazon Inspector container image scans with your CI/CD pipeline to scan for software vulnerabilities and produce vulnerability reports, which allow you to investigate and remediate risks before deployment. To set up your CI/CD integration, you can use plugins or create a custom CI/CD integration using the Amazon Inspector SBOM Generator and Amazon Inspector Scan API.

Topics

- Plugin integration
- Custom integration
- Setting up an AWS account to use the Amazon Inspector CI/CD integration
- Amazon Inspector Dockerfile checks
- Creating a custom CI/CD pipeline integration with Amazon Inspector Scan
- Using Amazon Inspector with GitHub actions
- Using the Amazon Inspector Jenkins plugin
- Using CodeCatalyst actions with Amazon Inspector
- Using Amazon Inspector Scan actions with CodePipeline
- Using the Amazon Inspector TeamCity plugin
- Using Amazon Inspector with GitLab components

Plugin integration

Amazon Inspector provides plugins for supported CI/CD solutions. You can install these plugins from their respective marketplaces and then use them to add Amazon Inspector Scans as a build step in your pipeline. The plugin build step runs the Amazon Inspector SBOM generator on the image you supply, and then runs the Amazon Inspector Scan API on the generated SBOM.

The following is an overview of how an Amazon Inspector CI/CD integration works through plugins:

1. You configure an AWS account to allow access to the Amazon Inspector Scan API. For instructions, see Setting up an AWS account to use the Amazon Inspector CI/CD integration.
2. You install the Amazon Inspector plugin from the marketplace.
3. You install and configure the Amazon Inspector SBOM Generator binary. For instructions, see Amazon Inspector SBOM Generator.
4. You add Amazon Inspector Scans as a build step in your CI/CD pipeline and configure the scan.
5. When you run a build, the plugin takes your container image as input and then runs the Amazon Inspector SBOM Generator on the image to generate a CycloneDX compatible SBOM.
6. From there, the plugin sends the generated SBOM to an Amazon Inspector Scan API endpoint which assesses each SBOM component for vulnerabilities.
7. The Amazon Inspector Scan API response is transformed into a vulnerability report in CSV, SBOM JSON, and HTML formats. The report contains details about any vulnerabilities that Amazon Inspector found.

Supported CI/CD solutions

Amazon Inspector currently supports the following CI/CD solutions. For complete instructions on setting up the CI/CD integration using a plugin, select the plugin for your CI/CD solution:

- GitHub actions
- Jenkins plugin
- TeamCity plugin

Custom integration

If Amazon Inspector does not provide plugins for your CI/CD solution, you can create your own custom CI/CD integration using a combination of the Amazon Inspector SBOM Generator and the Amazon Inspector Scan API. You can also use a custom integration to fine-tune scans using the options available through Amazon Inspector SBOM Generator.

The following is an overview of how a custom Amazon Inspector CI/CD integration works:

1. You configure an AWS account to allow access to the Amazon Inspector Scan API. For instructions, see Setting up an AWS account to use the Amazon Inspector CI/CD integration.
2. You install and configure the Amazon Inspector SBOM Generator binary. For instructions, see Amazon Inspector SBOM Generator.
3. You use the Amazon Inspector SBOM Generator to generate a CycloneDX compatible SBOM for your container image.
4. You use the Amazon Inspector Scan API on the generated SBOM to produce a vulnerability report.

For instructions on setting up a custom integration, see Creating a custom CI/CD pipeline integration with Amazon Inspector Scan.

Setting up an AWS account to use the Amazon Inspector CI/CD integration

To use the Amazon Inspector CI/CD integration, you must sign up for an AWS account. The AWS account must have an IAM role that grants your CI/CD pipleline access to the Amazon Inspector Scan API. Complete the tasks in the following topics to sign up for an AWS account, create an administrator user, and configure an IAM role for CI/CD integration.

> **Note**
>
> If you already signed up for an AWS account, you can skip to Configure an IAM role for CI/CD integration.

Topics

- Sign up for an AWS account
- Configure an IAM role for CI/CD integration

Sign up for an AWS account

To get started with AWS, you need an AWS account. There are two ways to sign up for AWS. If you want complete control over account configuration and where your content is stored, or if you have

regulated workloads such as HIPAA or FedRAMP, use Sign up for AWS (advanced). If you are a new user and want to use a login you already own to sign in to AWS and create an AWS environment with preconfigured defaults in select AWS Regions, use Sign up for AWS.

Configure an IAM role for CI/CD integration

To integrate Amazon Inspector scanning into your CI/CD pipeline you need to create an IAM policy that allows access to the Amazon Inspector Scan API that scans the software bill of materials (SBOMs). Then, you can attach that policy to an IAM role that your account can assume to run the Amazon Inspector Scan API.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, **Policies** and then choose **Create Policy**.
3. In **Policy Editor** select **JSON** and paste the following statement:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "inspector-scan:ScanSbom",
            "Resource": "*"
        }
    ]
}
```

4. Choose **Next**.
5. Give the policy a name, for example `InspectorCICDscan-policy`, and add an optional description, then choose **Create Policy**. This policy will be attached to the role you'll create in the next steps.
6. In the navigation pane of the IAM console, select **Roles** and then select **Create New Role**.
7. For **Trusted entity type** choose **Custom trust policy** and paste the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

8. Choose **Next**.
9. In **Add permissions** search for and select the policy you created earlier, then choose **Next**.
10. Give the role a name, for example `InspectorCICDscan-role`, and add an optional description, then choose `Create Role`.

Amazon Inspector Dockerfile checks

This section describes how to use the Amazon Inspector SBOM Generator to scan Dockerfiles and Docker container images for misconfigurations that introduce security vulnerabilities.

Topics

- Using Sbomgen Dockerfile checks
- Supported Dockerfile checks

Using Sbomgen Dockerfile checks

Dockerfile checks are conducted automatically when a file named `Dockerfile` or `*.Dockerfile` is discovered and when a Docker image is scanned.

You can disable Dockerfile checks using the `--skip-scanners dockerfile` argument. You also can combine Dockerfile checks with any available scanner, such as OS or 3rd-party packages.

Example Docker check commands

The following example commands show how to generate SBOMs for Dockerfiles and Docker container images, as well as for OS and 3rd-party packages.

```
# generate SBOM only containing Docker checks for Dockerfiles in a local directory
./inspector-sbomgen directory --path ./project/ --scanners dockerfile

# generate SBOM for container image will by default include Dockerfile checks
./inspector-sbomgen container --image image:tag

# generate SBOM only containing Docker checks for specific Dockerfiles and Alpine,
 Debian, and Rhel OS packages in a local directory
/inspector-sbomgen directory --path ./project/ --scanners dockerfile,dpkg,alpine-
apk,rhel-rpm

# generate SBOM only containing Docker checks for specific Dockerfiles in a local
 directory
./inspector-sbomgen directory --path ./project/ --skip-scanners dockerfile
```

Example file component

The following is an example of a Dockerfile finding for a file component.

```
{
     "bom-ref": "comp-2",
      "name": "dockerfile:data/docker/Dockerfile",
      "properties": [
        {
          "name": "amazon:inspector:sbom_scanner:dockerfile_finding:IN-DOCKER-001",
          "value": "affected_lines:27-27"
        }
      ],
      "type": "file"
    },
```

Example vulnerability response component

The following is an example of a Dockerfile finding for a vulnerability response component.

```
{
      "advisories": [
        {
```

```
          "url": "https://docs.docker.com/develop/develop-images/instructions/"
        }
      ],
      "affects": [
        {
          "ref": "comp-2"
        }
      ],
      "analysis": {
        "state": "in_triage"
      },
      "bom-ref": "vuln-13",
      "created": "2024-03-27T14:36:39Z",
      "description": "apt-get layer caching: Using apt-get update alone in a RUN
statement causes caching issues and subsequent apt-get install instructions to fail.",
      "id": "IN-DOCKER-001",
      "ratings": [
        {
          "method": "other",
          "severity": "info",
          "source": {
            "name": "AMAZON_INSPECTOR",
            "url": "https://aws.amazon.com/inspector/"
          }
        }
      ],
      "source": {
        "name": "AMAZON_INSPECTOR",
        "url": "https://aws.amazon.com/inspector/"
      },
      "updated": "2024-03-27T14:36:39Z"
    },
```


Note

If you invoke Sbomgen without the `--scan-sbom` flag, you can only view raw Dockerfile findings.

Supported Dockerfile checks

Sbomgen Dockerfile checks are supported for the following:

- The Sudo binary package
- Debian APT utilities
- Hardcoded secrets
- Root containers
- Runtime weakening command flags
- Runtime weakening environment variables

Each of these Dockerfile checks has a corresponding severity rating, which is noted at the top of the following topics.

> **Note**
>
> The recommendations described in the following topics are based on industry best practices.

The Sudo binary package

> **Note**
>
> The severity rating for this check is **Info**.

We recommend not installing or using the Sudo binary package because it has unpredictable TTY and signal-forwarding behavior. For more information, see User in the Docker Docs website. If your use case requires functionality similar the Sudo binary package, we recommened using Gosu.

Debian APT utilities

> **Note**
>
> The severity rating for this check is **High**.

The following are best practices for using Debian APT utilities.

Combining `apt-get` commands in a single Run statement to avoid caching issues

We recommend combining `apt-get` commands in a single RUN statement inside of your Docker container. Using `apt-get update` by itself results in caching issues and subsequent `apt-get install` instructions to fail. For more information, see apt-get in the Docker Docs website.

> **Note**
>
> The caching behavior described also can occur inside of your Docker container if the Docker container software is out of date.

Using the APT command-line utility in a non-interactive manner

We recommend using the APT command-line utility interactively. The APT command-line utility is designed as an end-user tool, and its behavior changes between versions. For more information, see Script Usage and differences from other APT tools in the Debian website.

Hard-coded secrets

> **Note**
>
> The severity rating for this check is **Critical**.

Confidential information in your Dockerfile is considered a hard-coded secret. The following hard-coded secrets can be identified through Sbomgen Docker file checks:

- AWS access key IDs – `AKIAIOSFODNN7EXAMPLE`
- AWS secret keys – `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
- DockerHub personal access tokens – `dckr_pat_thisisa27charexample1234567`
- GitHub personal access tokens – `ghp_examplev61wY7Pj1YnotrealUoY123456789`
- GitLab personal access tokens – `glpat-12345example12345678`

Root containers

> **Note**
>
> The severity marker for this check is **Info**.

We recommend running Docker containers without root privileges. For containerized workloads that cannot run without root privileges, we recommend building your applications using a principle with the least amount of privileges. For more information, see User in the Docker Docs website.

Runtime weakening environment variables

> **Note**
>
> The severity rating for this check is **High**.

Several command line utilities or programming language runtimes support bypassing secure defaults, which allows execution through insecure methods.

NODE_TLS_REJECT_UNAUTHORIZED=0

When Node.js processes run with `NODE_TLS_REJECT_UNAUTHORIZED` set to `0`, TLS certificate validation is disabled. For more information, see NODE_TLS_REJECT_UNAUTHORIZED=0 in the Node.js website.

GIT_SSL_NO_VERIFY=*

When git command line processes run with `GIT_SSL_NO_VERIFY` set, Git skips verifying TLS certificates. For more information, see Environment variables in the Git website.

PIP_TRUSTED_HOST=*

When Python pip command line processes run with `PIP_TRUSTED_HOST` set, Pip skips verifying TLS certificates on the specified domain. For more information, see --trusted-host in the Pip website.

NPM_CONFIG_STRICT_SSL=false

When Node.js npm command line processes run with `NPM_CONFIG_STRICT_SSL` set to false, the Node Package Manager (npm) utility will connect to the NPM registry without validating TLS certificates. For more information, see strict-ssl in the npm Docs website website.

Runtime weakening command flags

> **Note**
>
> The severity rating for this check is **High**.

Similar to runtime weakening environment variables, several command line utilities or programming language runtimes support bypassing secure defaults, which allows execution through insecure methods.

`npm --strict-ssl=false`

When Node.js npm command line processes are run with the `--strict-ssl=false` flag, the Node Package Manager (npm) utility connects to the NPM registry without validating TLS certificates. For more information, see strict-ssl in the npm Docs website.

`apk --allow-untrusted`

When the Alpine Package Keeper utility is run with the `--allow-untrusted` flag, `apk` will install packages with no or untrusted signatures. For more information, see the following repository in the Apline website.

`apt-get --allow-unauthenticated`

When the Debian `apt-get` package utility is run with the `--allow-unauthenticated` flag, `apt-get` doesn't check package validity. For more information, see APT-Get(8) in the Debian website.

`pip --trusted-host`

When the Python pip utility is run with the `--trusted-host` flag, the specified hostname will bypass TLS certificate validation. For more information, see --trusted-host in the Pip website.

`rpm --nodigest, --nosignature, --noverify, --nofiledigest`

When the RPM-based package manager `rpm` is run with the `--nodigest`, `--nosignature`, `--noverify`, and `--nofiledigest` flags, the RPM package manager doesn't validate package headers, signatures, or files when installing a package. For more information, see the following RPM manual page in the RPM website.

`yum-config-manager --setopt=sslverify false`

When the RPM-based package manager `yum-config-manager` is run with the `--setopt=sslverify` flag set to false, the YUM package manager doesn't validate TLS certificates. For more information, see the following YUM manual page in the Man7 website.

`yum --nogpgcheck`

When the RPM-based package manager `yum` is run with the `--nogpgcheck` flag, the YUM package manager skips checking GPG signatures on packages. For more information, see yum(8) in the Man7 website.

`curl --insecure, curl -k`

When `curl` is run with the `--insecure` or `-k` flag, TLS certificate validation is disabled. By default, every secure connection that `curl` makes is verified to be secure before the transfer takes place. This option makes `curl` skip the verification step and proceed without checking. For more information, see the following Curl manual page in the Curl website.

`wget --no-check-certificate`

When `wget` is run with the `--no-check-certificate` flag, TLS certificate validation is disabled. For more information, see the following Wget manual page in the GNU website.

Removal checks for OS package databases within containers

> **Note**
>
> The severity rating for this check is **Info**.

The removal of an operating system package database reduces the ability to scan the complete inventory of a container image's software. These databases should be left intact during container build steps.

Removal checks for an OS package database is supported for the following package managers:

Alpine Package Keeper (APK)

Container images utilizing the APK package manager for installed software must make sure APK system files are not removed during a build. For more information, see the APK manpages system files documentation on the Arch Linux website.

Debian Package Manager (DPKG)

Containers utilizing the DPKG package manager, such as Debian, Ubuntu, or Distroless based images, must make sure the DPKG database are not removed during a container build. For more information, see the DPKG manpages system files documentation on the Ubuntu website.

RPM Package Manager (RPM)

Containers utilizing the RPM Package Manager (yum/dnf), such as Amazon Linux or Red Hat Enterprise Linux, must make sure the RPM database is not removed during a container build. For more information, see the RPM manpages system files documentation on the RPM website.

Creating a custom CI/CD pipeline integration with Amazon Inspector Scan

We recommend that you use the Amazon Inspector CI/CD plugins if the Amazon Inspector CI/CD plugins are available for your CI/CD solution. If the Amazon Inspector CI/CD plugins aren't available for your CI/CD solution, you can use a combination of the Amazon Inspector SBOM Generator and the Amazon Inspector Scan API to create a custom CI/CD integration. The following steps describe how to create a custom CI/CD pipeline integration with Amazon Inspector Scan.

> **Tip**
>
> You can use the Amazon Inspector SBOM Generator (Sbomgen) to skip Step 3 and Step 4 if you want to generate and scan your SBOM in a single command.

Step 1. Configuring AWS account

Configure an AWS account that provides access to the Amazon Inspector Scan API. For more information, see Setting up an AWS account to use the Amazon Inspector CI/CD integration.

Step 2. Installing Sbomgen binary

Install and configure the Sbomgen binary. For more information, see Installing Sbomgen.

Step 3. Using Sbomgen

Use the Sbomgen to create an SBOM file for a container image that you want to scan.

You can use the following example. Replace *image:id* with the name of the image that you to scan. Replace *sbom_path.json* with the location where you want to save the SBOM output.

Example

```
./inspector-sbomgen container --image image:id -o sbom_path.json
```

Step 4. Calling the Amazon Inspector Scan API

Call the `inspector-scan` API to scan the generated SBOM and provide a vulnerability report.

You can use the following example. Replace *sbom_path.json* with the location of a valid CycloneDX compatible SBOM file. Replace *ENDPOINT* with the API endpoint for the AWS Region where you're currently authenticated. Replace *REGION* with the corresponding Region.

Example

```
aws inspector-scan scan-sbom --sbom file://sbom_path.json --endpoint
ENDPOINT-URL --region REGION
```

For a complete list of AWS Regions and endpoints, see Regions and endpoints.

(Optional) Step 5. Generate and scan SBOM in a single command

> **Note**
>
> Only complete this step if you skipped Step 3 and Step 4.

Generate and scan your SBOM in a single command using the `--scan-bom` flag.

You can use the following example. Replace *image:id* with the name of the image that you want to scan. Replace *profile* with the corresponding profile. Replace *REGION* with the corresponding Region. Replace */tmp/scan.json* with the location of the scan.json file in the tmp directory.

Example

```
./inspector-sbomgen container --image image:id --scan-sbom --aws-profile
profile --aws-region REGION -o /tmp/scan.json
```

For a complete list of AWS Regions and endpoints, see Regions and endpoints.

API output formats

The Amazon Inspector Scan API can output a vulnerability report in CycloneDX 1.5 format or Amazon Inspector finding JSON. The default can be changed using the `--output-format` flag.

Example of CycloneDX 1.5 format output - Linux


```
{
  "status": "SBOM parsed successfully, 1 vulnerabilities found",
  "sbom": {
    "bomFormat": "CycloneDX",
    "specVersion": "1.5",
    "serialNumber": "urn:uuid:0077b45b-ff1e-4dbb-8950-ded11d8242b1",
    "metadata": {
      "properties": [
        {
          "name": "amazon:inspector:sbom_scanner:critical_vulnerabilities",
          "value": "1"
        },
        {
          "name": "amazon:inspector:sbom_scanner:high_vulnerabilities",
          "value": "0"
        },
        {
          "name": "amazon:inspector:sbom_scanner:medium_vulnerabilities",
          "value": "0"
        },
        {
          "name": "amazon:inspector:sbom_scanner:low_vulnerabilities",
          "value": "0"
        }
      ],
      "tools": [
        {
          "name": "CycloneDX SBOM API",
          "vendor": "Amazon Inspector",
          "version": "empty:083c9b00:083c9b00:083c9b00"
        }
      ],
      "timestamp": "2023-06-28T14:15:53.760Z"
    },
    "components": [
      {
```

```
        "bom-ref": "comp-1",
        "type": "library",
        "name": "log4j-core",
        "purl": "pkg:maven/org.apache.logging.log4j/log4j-core@2.12.1",
        "properties": [
          {
            "name": "amazon:inspector:sbom_scanner:path",
            "value": "/home/dev/foo.jar"
          }
        ]
      }
    ],
    "vulnerabilities": [
      {
        "bom-ref": "vuln-1",
        "id": "CVE-2021-44228",
        "source": {
          "name": "NVD",
          "url": "https://nvd.nist.gov/vuln/detail/CVE-2021-44228"
        },
        "references": [
          {
            "id": "GHSA-jfh8-c2jp-5v3q",
            "source": {
              "name": "GITHUB",
              "url": "https://github.com/advisories/GHSA-jfh8-c2jp-5v3q"
            }
          }
        ],
        "ratings": [
          {
            "source": {
              "name": "NVD",
              "url": "https://www.first.org/cvss/v3-1/"
            },
            "score": 10.0,
            "severity": "critical",
            "method": "CVSSv31",
            "vector": "AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H"
          },
          {
            "source": {
              "name": "NVD",
              "url": "https://www.first.org/cvss/v2/"
```

```
            },
            "score": 9.3,
            "severity": "critical",
            "method": "CVSSv2",
            "vector": "AC:M/Au:N/C:C/I:C/A:C"
          },
          {
            "source": {
              "name": "EPSS",
              "url": "https://www.first.org/epss/"
            },
            "score": 0.97565,
            "severity": "none",
            "method": "other",
            "vector": "model:v2023.03.01,date:2023-06-27T00:00:00+0000"
          },
          {
            "source": {
              "name": "GITHUB",
              "url": "https://github.com/advisories/GHSA-jfh8-c2jp-5v3q"
            },
            "score": 10.0,
            "severity": "critical",
            "method": "CVSSv31",
            "vector": "AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H"
          }
        ],
        "cwes": [
          400,
          20,
          502
        ],
        "description": "Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security
releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages,
and parameters do not protect against attacker controlled LDAP and other JNDI related
endpoints. An attacker who can control log messages or log message parameters can
execute arbitrary code loaded from LDAP servers when message lookup substitution is
enabled. From log4j 2.15.0, this behavior has been disabled by default. From version
2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely
removed. Note that this vulnerability is specific to log4j-core and does not affect
log4net, log4cxx, or other Apache Logging Services projects.",
        "advisories": [
          {
```

```
            "url": "https://www.intel.com/content/www/us/en/security-center/advisory/
intel-sa-00646.html"
          },
          {
            "url": "https://support.apple.com/kb/HT213189"
          },
          {
            "url": "https://msrc-blog.microsoft.com/2021/12/11/microsofts-response-to-
cve-2021-44228-apache-log4j2/"
          },
          {
            "url": "https://logging.apache.org/log4j/2.x/security.html"
          },
          {
            "url": "https://www.debian.org/security/2021/dsa-5020"
          },
          {
            "url": "https://cert-portal.siemens.com/productcert/pdf/ssa-479842.pdf"
          },
          {
            "url": "https://www.oracle.com/security-alerts/alert-cve-2021-44228.html"
          },
          {
            "url": "https://www.oracle.com/security-alerts/cpujan2022.html"
          },
          {
            "url": "https://cert-portal.siemens.com/productcert/pdf/ssa-714170.pdf"
          },
          {
            "url": "https://lists.fedoraproject.org/archives/list/package-
announce@lists.fedoraproject.org/message/M5CSVUNV4HWZZXGOKNSK6L7RPM7BOKIB/"
          },
          {
            "url": "https://cert-portal.siemens.com/productcert/pdf/ssa-397453.pdf"
          },
          {
            "url": "https://cert-portal.siemens.com/productcert/pdf/ssa-661247.pdf"
          },
          {
            "url": "https://lists.fedoraproject.org/archives/list/package-
announce@lists.fedoraproject.org/message/VU57UJDCFIASIO35GC55JMKSRXJMCDFM/"
          },
          {
            "url": "https://www.oracle.com/security-alerts/cpuapr2022.html"
```

```
          },
          {
            "url": "https://twitter.com/kurtseifried/status/1469345530182455296"
          },
          {
            "url": "https://tools.cisco.com/security/center/content/
CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd"
          },
          {
            "url": "https://lists.debian.org/debian-lts-announce/2021/12/msg00007.html"
          },
          {
            "url": "https://www.kb.cert.org/vuls/id/930724"
          }
        ],
        "created": "2021-12-10T10:15:00Z",
        "updated": "2023-04-03T20:15:00Z",
        "affects": [
          {
            "ref": "comp-1"
          }
        ],
        "properties": [
          {
            "name": "amazon:inspector:sbom_scanner:exploit_available",
            "value": "true"
          },
          {
            "name": "amazon:inspector:sbom_scanner:exploit_last_seen_in_public",
            "value": "2023-03-06T00:00:00Z"
          },
          {
            "name": "amazon:inspector:sbom_scanner:cisa_kev_date_added",
            "value": "2021-12-10T00:00:00Z"
          },
          {
            "name": "amazon:inspector:sbom_scanner:cisa_kev_date_due",
            "value": "2021-12-24T00:00:00Z"
          },
          {
            "name": "amazon:inspector:sbom_scanner:fixed_version:comp-1",
            "value": "2.15.0"
          }
        ]
```

```
      }
    ]
  }
}
```

Example of CycloneDX 1.5 format output - Windows

```
{
  "sbom": {
    "specVersion": "1.5",
    "metadata": {
      "tools": {
        "services": [
          {
            "name": "Amazon Inspector Scan SBOM API",
            "version": "d79c681c+d73b8663+5e50a5ab"
          }
        ]
      },
      "properties": [
        {
          "name": "amazon:inspector:sbom_scanner:critical_vulnerabilities",
          "value": "0"
        },
        {
          "name": "amazon:inspector:sbom_scanner:high_vulnerabilities",
          "value": "0"
        },
        {
          "name": "amazon:inspector:sbom_scanner:medium_vulnerabilities",
          "value": "1"
        },
        {
          "name": "amazon:inspector:sbom_scanner:low_vulnerabilities",
          "value": "0"
        },
        {
          "name": "amazon:inspector:sbom_scanner:other_vulnerabilities",
          "value": "0"
        }
      ],
      "timestamp": "2026-03-17T00:00:52.344Z"
    },
```

```
    "components": [
      {
        "bom-ref": "comp-1",
        "name": "defender",
        "purl": "pkg:generic/microsoft/defender@4.18.25110.5",
        "type": "application",
        "version": "4.18.25110.5",
        "properties": [
          {
            "name": "amazon:inspector:sbom_scanner:source_file_scanner",
            "value": "windows-apps"
          },
          {
            "name": "amazon:inspector:sbom_scanner:source_package_collector",
            "value": "windows-app-defender"
          },
          {
            "name": "amazon:inspector:sbom_scanner:path",
            "value": "vol-0d994b0984fdaa2af:\\ProgramData\\Microsoft\\Windows Defender\
\platform\\4.18.25110.5-0"
          }
        ]
      }
    ],
    "serialNumber": "urn:uuid:6bed582d-191e-4cb7-9875-950dd0b99700",
    "bomFormat": "CycloneDX",
    "vulnerabilities": [
      {
        "advisories": [
          {
            "url": "https://support.microsoft.com/help/5011487"
          },
          {
            "url": "https://catalog.update.microsoft.com/v7/site/Search.aspx?
q=KB5011487"
          }
        ],
        "bom-ref": "vuln-1",
        "references": [
          {
            "id": "CVE-2022-23278",
            "source": {
              "name": "MICROSOFT",
```

```
              "url": "https://msrc.microsoft.com/update-guide/vulnerability/
CVE-2022-23278"
            }
          }
        ],
        "ratings": [
          {
            "severity": "none",
            "score": 0.02691,
            "method": "other",
            "vector": "model:v2025.03.14,date:2026-03-15T12:55:00Z",
            "source": {
              "name": "EPSS",
              "url": "https://api.first.org/data/v1/epss?cve=CVE-2022-23278"
            }
          },
          {
            "severity": "medium",
            "score": 5.9,
            "method": "CVSSv31",
            "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N",
            "source": {
              "name": "MICROSOFT",
              "url": "https://catalog.update.microsoft.com/v7/site/Search.aspx?
q=KB5011487"
            }
          }
        ],
        "created": "2022-03-08T08:00:00Z",
        "description": "Security Update for Defender (2022-03). Install KB5011487 to
 remediate. A reboot is required for this update to take effect.",
        "affects": [
          {
            "ref": "comp-1"
          }
        ],
        "id": "KB5011487",
        "source": {
          "name": "MICROSOFT",
          "url": "https://catalog.update.microsoft.com/v7/site/Search.aspx?q=KB5011487"
        },
        "published": "2022-03-08T08:00:00Z",
        "analysis": {
          "state": "in_triage"
```

```
        },
        "properties": [
          {
            "name": "amazon:inspector:sbom_scanner:priority",
            "value": "standard"
          },
          {
            "name": "amazon:inspector:sbom_scanner:priority_intelligence",
            "value": "unverified"
          },
          {
            "name": "amazon:inspector:sbom_scanner:fixed_version:comp-1",
            "value": "10.0.19042.1586"
          }
        ]
      }
    ]
  }
}
```

Example of Inspector format output - Linux

```
{
  "status": "SBOM parsed successfully, 1 vulnerability found",
  "inspector": {
    "messages": [
      {
        "name": "foo",
        "purl": "pkg:maven/foo@1.0.0", // Will not exist in output if missing in sbom
        "info": "Component skipped: no rules found."
      }
    ],
    "vulnerability_count": {
      "critical": 1,
      "high": 0,
      "medium": 0,
      "low": 0
    },
    "vulnerabilities": [
      {
        "id": "CVE-2021-44228",
        "severity": "critical",
        "source": "https://nvd.nist.gov/vuln/detail/CVE-2021-44228",
```

```
        "related": [
          "GHSA-jfh8-c2jp-5v3q"
        ],
        "description": "Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security
 releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages,
 and parameters do not protect against attacker controlled LDAP and other JNDI related
 endpoints. An attacker who can control log messages or log message parameters can
 execute arbitrary code loaded from LDAP servers when message lookup substitution is
 enabled. From log4j 2.15.0, this behavior has been disabled by default. From version
 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely
 removed. Note that this vulnerability is specific to log4j-core and does not affect
 log4net, log4cxx, or other Apache Logging Services projects.",
        "references": [
          "https://www.intel.com/content/www/us/en/security-center/advisory/intel-
sa-00646.html",
          "https://support.apple.com/kb/HT213189",
          "https://msrc-blog.microsoft.com/2021/12/11/microsofts-response-to-
cve-2021-44228-apache-log4j2/",
          "https://logging.apache.org/log4j/2.x/security.html",
          "https://www.debian.org/security/2021/dsa-5020",
          "https://cert-portal.siemens.com/productcert/pdf/ssa-479842.pdf",
          "https://www.oracle.com/security-alerts/alert-cve-2021-44228.html",
          "https://www.oracle.com/security-alerts/cpujan2022.html",
          "https://cert-portal.siemens.com/productcert/pdf/ssa-714170.pdf",
          "https://lists.fedoraproject.org/archives/list/package-
announce@lists.fedoraproject.org/message/M5CSVUNV4HWZZXGOKNSK6L7RPM7BOKIB/",
          "https://cert-portal.siemens.com/productcert/pdf/ssa-397453.pdf",
          "https://cert-portal.siemens.com/productcert/pdf/ssa-661247.pdf",
          "https://lists.fedoraproject.org/archives/list/package-
announce@lists.fedoraproject.org/message/VU57UJDCFIASIO35GC55JMKSRXJMCDFM/",
          "https://www.oracle.com/security-alerts/cpuapr2022.html",
          "https://twitter.com/kurtseifried/status/1469345530182455296",
          "https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-
sa-apache-log4j-qRuKNEbd",
          "https://lists.debian.org/debian-lts-announce/2021/12/msg00007.html",
          "https://www.kb.cert.org/vuls/id/930724"
        ],
        "created": "2021-12-10T10:15:00Z",
        "updated": "2023-04-03T20:15:00Z",
        "properties": {
          "cisa_kev_date_added": "2021-12-10T00:00:00Z",
          "cisa_kev_date_due": "2021-12-24T00:00:00Z",
          "cwes": [
            400,
```

```
            20,
            502
          ],
          "cvss": [
            {
              "source": "NVD",
              "severity": "critical",
              "cvss3_base_score": 10.0,
              "cvss3_base_vector": "AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H",
              "cvss2_base_score": 9.3,
              "cvss2_base_vector": "AC:M/Au:N/C:C/I:C/A:C"
            },
            {
              "source": "GITHUB",
              "severity": "critical",
              "cvss3_base_score": 10.0,
              "cvss3_base_vector": "AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H"
            }
          ],
          "epss": 0.97565,
          "exploit_available": true,
          "exploit_last_seen_in_public": "2023-03-06T00:00:00Z"
        },
        "affects": [
          {
            "installed_version": "pkg:maven/org.apache.logging.log4j/log4j-
core@2.12.1",
            "fixed_version": "2.15.0",
            "path": "/home/dev/foo.jar"
          }
        ]
      }
    ]
  }
}
```

Example of Inspector format output - Windows

```
{
  "sbom": {
    "vulnerabilities": [
      {
```

```
        "severity": "medium",
        "priority_intelligence": "unverified",
        "related": [
          "CVE-2022-23278"
        ],
        "references": [
          "https://support.microsoft.com/help/5011487",
          "https://catalog.update.microsoft.com/v7/site/Search.aspx?q=KB5011487"
        ],
        "created": "2022-03-08T08:00:00Z",
        "description": "Security Update for Defender (2022-03). Install KB5011487 to
 remediate. A reboot is required for this update to take effect.",
        "affects": [
          {
            "path": "vol-0d994b0984fdaa2af:\\ProgramData\\Microsoft\\Windows Defender\
\platform\\4.18.25110.5-0",
            "fixed_version": "10.0.19042.1586",
            "installed_version": "pkg:generic/microsoft/defender@4.18.25110.5"
          }
        ],
        "id": "KB5011487",
        "source": "https://catalog.update.microsoft.com/v7/site/Search.aspx?
q=KB5011487",
        "published": "2022-03-08T08:00:00Z",
        "priority": "standard",
        "properties": {
          "epss": 0.0269099995,
          "cvss": [
            {
              "severity": "medium",
              "cvss_3_base_score": 5.9000000954,
              "cvss_3_base_vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N",
              "source": "MICROSOFT",
              "url": "https://catalog.update.microsoft.com/v7/site/Search.aspx?
q=KB5011487"
            }
          ]
        }
      }
    ],
    "vulnerability_count": {
      "high": 0,
      "other": 0,
      "critical": 0,
```

```
      "low": 0,
      "medium": 1
    }
  }
}
```

Using Amazon Inspector with GitHub actions

You can use Amazon Inspector with GitHub actions to add Amazon Inspector vulnerability scans to your GitHub workflows. This leverages the Amazon Inspector SBOM Generator and Amazon Inspector Scan API to produce detailed reports at the end of your build, so you can investigate and remediate risk before deployment. Amazon Inspector vulnerability scans can be configured to pass or fail workflows based on the number and severity of vulnerabilities detected. You can view the latest version of the Amazon Inspector action on the GitHub website. For information about how to integrate Amazon Inspector Scan into your CI/CD pipeline, see Integrating Amazon Inspector scans into your CI/CD pipeline. For a list of operating systems and programming languages that Amazon Inspector supports, see Supported operating systems and programming languages.

Using the Amazon Inspector Jenkins plugin

The Amazon Inspector Jenkins plugin adds Amazon Inspector vulnerability scans to your Jenkins builds. The plugin runs the Amazon Inspector SBOM Generator to create a software bill of materials (SBOM) for a container image. It sends the SBOM to the Amazon Inspector Scan API and produces detailed reports at the end of your build, so you can investigate and remediate risk before deployment. You can fail the build based on the number and severity of vulnerabilities, Exploit Prediction Scoring System (EPSS) scores, specific common vulnerabilities and exposures (CVEs), or malicious packages. For the latest version of the plugin, see Amazon Inspector Scanner on the Jenkins plugins website. The following steps describe how to set up the Amazon Inspector Jenkins plugin.

> **Important**
>
> The plugin requires Jenkins 2.541.3 or later. The Amazon Inspector SBOM Generator is available only for Linux, so run your builds on Linux with an x86_64 (amd64) or arm64 CPU. With the **Automatic** installation method, the Jenkins controller must also run Linux.

Step 1. Set up an AWS account

Configure an AWS account with an IAM role that allows access to the Amazon Inspector Scan API. For instructions, see Setting up an AWS account to use the Amazon Inspector CI/CD integration.

Step 2. Install the Amazon Inspector Jenkins plugin

The following procedure describes how to install the Amazon Inspector Jenkins plugin from the Jenkins dashboard.

1. From the Jenkins dashboard, choose **Manage Jenkins**, and then choose **Plugins**.
2. Choose **Available plugins**.
3. Search for **Amazon Inspector Scanner**, select it, and then choose **Install**.

(Optional) Step 3. Add Docker credentials to Jenkins

> **Note**
>
> Only add Docker credentials if the image is in a private repository. Otherwise, skip this step.

The following procedure describes how to add Docker credentials to Jenkins from the Jenkins dashboard.

1. From the Jenkins dashboard, choose **Manage Jenkins**, **Credentials**, and then **System**.
2. Choose **Global**, and then choose **Add Credentials**.
3. Select **Username with password**, and then choose **Next**.
4. For **Scope**, select **Global (Jenkins, nodes, items, all child items, etc)**.
5. Enter the user name and password for your registry, and then choose **Create**. If your registry supports access tokens, we recommend that you use a read-only token as the password instead of your account password.

(Optional) Step 4. Add AWS credentials to Jenkins

The plugin calls the Amazon Inspector Scan API from the Jenkins controller. When you configure the build step in Step 6, you choose one of the following ways for the plugin to authenticate:

- **(Recommended) IAM role** – The plugin assumes the role that you created in Step 1. To assume the role, the plugin uses the credentials that the Jenkins controller already has, such as an Amazon EC2 instance profile. Those credentials need permission to call `sts:AssumeRole` on the role. The plugin can also assume the role with an OpenID Connect (OIDC) ID token that Jenkins issues. OIDC doesn't require you to store long-term AWS credentials in Jenkins.
- **AWS credentials** – An access key for an IAM user, stored in Jenkins.
- **AWS profile name** – A named profile in the AWS configuration files of the user that runs the Jenkins controller.

If you don't configure any of these, the plugin uses the default credential provider chain on the Jenkins controller, such as environment variables or an Amazon EC2 instance profile. Any job that uses the build step can use those credentials, so give them only the permissions of the role that you created in Step 1. You only need to add credentials to Jenkins to use an access key or OIDC. Otherwise, skip this step.

> **Warning**
>
> This scenario requires IAM users with programmatic access and long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed. Access keys can be updated if necessary. For more information, see Update access keys in the *IAM User Guide*.

To add an access key

1. From the Jenkins dashboard, choose **Manage Jenkins**, **Credentials**, and then **System**.
2. Choose **Global**, and then choose **Add Credentials**.
3. Select **AWS Credentials**, and then choose **Next**.
4. Enter your **Access Key ID** and **Secret Access Key**, and then choose **Create**.

To add an OIDC credential

1. From the Jenkins dashboard, choose **Manage Jenkins**, **Credentials**, and then **System**.
2. Choose **Global**, and then choose **Add Credentials**.
3. Select **OpenID Connect id token**, and then choose **Next**.

4. For **Audience**, enter `sts.amazonaws.com`, and then choose **Create**. To see the issuer URI of the credential, open it and choose **Update credential**.
5. In IAM, create an OIDC identity provider with the issuer URI as the provider URL and `sts.amazonaws.com` as the audience. For more information, see Create an OpenID Connect (OIDC) identity provider in IAM.
6. Add a statement to the trust policy of the role that you created in Step 1 that allows the provider to call `sts:AssumeRoleWithWebIdentity`. Include a condition on the provider's `aud` key with the value `sts.amazonaws.com`. To allow only specific jobs to assume the role, also add a condition on the `sub` key, which contains the URL of the job by default. The following example statement allows only the job *`my-job`* on a Jenkins controller at *`jenkins.example.com`* to assume the role.

```
{
    "Effect": "Allow",
    "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/jenkins.example.com/
oidc"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
        "StringEquals": {
            "jenkins.example.com/oidc:aud": "sts.amazonaws.com",
            "jenkins.example.com/oidc:sub": "https://jenkins.example.com/job/my-
job/"
        }
    }
}
```

Note

AWS must be able to reach the issuer URI over HTTPS, and the URI shouldn't include a port number. If your Jenkins controller isn't reachable from the internet, or its URL includes a port, use a different issuer. For more information, see Picking an issuer in the OpenID Connect Provider plugin documentation on the GitHub website.

Step 5. Allow Jenkins to display the HTML report

The plugin creates an HTML report that uses inline CSS and JavaScript. By default, Jenkins serves build artifacts with a Content Security Policy (CSP) that blocks both. As a result, the report has no styling and shows zero findings, even when the scan found vulnerabilities. Use one of the following options so that the report displays correctly.

- **(Recommended) Serve build artifacts from another domain** – Choose **Manage Jenkins** and then **System**. Under **Serve resource files from another domain**, enter a **Resource Root URL** that points to your Jenkins controller with a different host name than your Jenkins URL. Jenkins doesn't apply the CSP to files that it serves from this URL. For more information, see Resource Root URL in the Jenkins documentation.
- **Remove the CSP** – Set the `hudson.model.DirectoryBrowserSupport.CSP` system property to an empty value. To apply the change immediately, choose **Manage Jenkins** and then **Script Console**, run `System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "")`, and then reload the report. Only Jenkins administrators can use the Script Console. This change lasts until Jenkins restarts. To keep it, start Jenkins with the Java option `-Dhudson.model.DirectoryBrowserSupport.CSP=`.

> **Warning**
>
> Removing the CSP allows any HTML file in a workspace or build artifact to run scripts in the browsers of your Jenkins users. Only remove it if you trust everyone who can change files in your builds, and all of your build agents. For more information, see Configuring Content Security Policy in the Jenkins documentation.

Step 6. Add Amazon Inspector Scan to your build

You can add Amazon Inspector Scan as a build step in a freestyle project, or as a step in a Jenkins pipeline.

Add Amazon Inspector Scan as a build step

1. On the configuration page of your project, under **Build Steps**, choose **Add build step**, and then choose **Amazon Inspector Scan**.
2. For **Inspector-sbomgen Installation Method**, choose one of the following:

- **Automatic** – The plugin downloads the latest version of the Amazon Inspector SBOM Generator at the start of each build. The Jenkins controller must run Linux and be able to reach `amazon-inspector-sbomgen.s3.amazonaws.com` over HTTPS.
- **Manual** – Enter the full path to an `inspector-sbomgen` binary that you downloaded, such as `/opt/inspector/inspector-sbomgen`. Choose this option to use a specific version of the Amazon Inspector SBOM Generator, or if the Jenkins controller can't download it. The plugin checks the path on the Jenkins controller, but runs the binary on the machine that runs the build. If these are different machines, put the binary at the same path on both. For download links, see Installing Sbomgen.

3. For **Image ID**, enter the image to scan. The image can be local, remote, or archived, and its name must follow the Docker naming convention:
   - For a local or remote image: `NAME[:TAG|@DIGEST]`
   - For an image exported to a tar file: `/path/to/image.tar`
4. (Optional) For **Report Artifact Name**, enter a prefix for the file name of the SBOM that the build archives. The default is `default-report`. The name can contain up to 255 letters, numbers, periods (.), underscores (_), and hyphens (-).
5. (Optional) For **Skip Files**, enter the files and directories to exclude from the scan, one per line or separated by commas without spaces. Consider this option for large directories that don't need to be scanned.
6. (Optional) For **Docker Credentials**, choose the credentials that you added in Step 3.
7. For **AWS Region**, choose the Region to send the scan request to.
8. Under **AWS Authentication Options (Choose One)**, configure one of the following options. If you leave all of them empty, the plugin uses the default credential provider chain on the Jenkins controller.
   - For **IAM Role**, enter the Amazon Resource Name (ARN) of the role that you created in Step 1, such as `arn:aws:iam::111122223333:role/InspectorCICDscan-role`. To assume the role with OIDC, also choose the credential that you added in Step 4 for **OIDC Credentials**.
   - For **AWS Credentials**, choose the access key that you added in Step 4.
   - For **AWS Profile Name**, enter the name of a profile on the Jenkins controller.

If you configure more than one option, the plugin uses the first one in this order: **AWS Credentials**, **IAM Role** with **OIDC Credentials**, **IAM Role**, and then **AWS Profile Name**.

9. (Optional) Under **Scan Output Filters**, choose the checks that can fail the build. For a description of each option, see Scan output filters.
10. Choose **Save**.

Add Amazon Inspector Scan to a Jenkins pipeline

The following declarative pipeline downloads the Amazon Inspector SBOM Generator automatically and authenticates with an IAM role. It fails the build if the scan finds more vulnerabilities than the severity thresholds allow. Replace *IMAGE_ID* with your image, such as `alpine:latest`, and *REGION* with your AWS Region, such as `us-east-1`. Replace *IAM_ROLE_ARN* with the ARN of the role that you created in Step 1.

```
pipeline {
    agent any
    stages {
        stage('Amazon Inspector scan') {
            steps {
                script {
                    step([
                        $class:
 'com.amazon.inspector.jenkins.amazoninspectorbuildstep.AmazonInspectorBuilder',
                        sbomgenSelection: 'automatic',
                        archivePath: 'IMAGE_ID',
                        awsRegion: 'REGION',
                        iamRole: 'IAM_ROLE_ARN',
                        credentialId: '',
                        isSeverityThresholdEnabled: true,
                        countCritical: 0,
                        countHigh: 0,
                        countMedium: 5,
                        countLow: 10
                    ])
                }
            }
        }
    }
```

```
}
```

To use an `inspector-sbomgen` binary that you installed, replace `sbomgenSelection: 'automatic'` with the following lines. Replace *SBOMGEN_PATH* with the full path to the binary, such as `/opt/inspector/inspector-sbomgen`.

```
sbomgenSelection: 'manual',
sbomgenPath: 'SBOMGEN_PATH',
```

The step accepts the following parameters. Each parameter matches an option of the build step.

Amazon Inspector Scan pipeline parameters

Parameter	Description
`archivePath`	Required. The image to scan (**Image ID**).
`awsRegion`	Required. The AWS Region to send the scan request to.
`sbomgenSelection`	`automatic` (default) or `manual`. Any other value uses `automatic` .
`sbomgenPath`	The full path to the `inspector-sbomgen` binary. Required when `sbomgenSelection` is `manual`.
`iamRole`	The ARN of the IAM role to assume.
`oidcCredentialId`	The ID of the OIDC credential that the plugin uses to assume `iamRole`.
`awsCredentialId`	The ID of the access key credential that you added in Step 4.
`awsProfileName`	The name of a profile on the Jenkins controlle r.

Parameter	Description
`credentialId`	The ID of the Docker credentials for a private repository. Use an empty value (`''`) for a public image.
`reportArtifactName`	A prefix for the file name of the SBOM that the build archives. The default is `default-report` .
`sbomgenSkipFiles`	The files and directories to exclude from the scan, one per line or separated by commas without spaces.
`isSeverityThresholdEnabled` , `countCritical` , `countHigh` , `countMedium` , `countLow`	Severity thresholds. A count that you don't set is `0`.
`isEpssThresholdEnabled` , `epssThreshold`	EPSS threshold. If you enable it without setting `epssThreshold` , the plugin skips the check.
`isSuppressedCveEnabled` , `suppressedCveList`	CVE suppression list.
`isAutoFailCveEnabled` , `autoFailCveList`	CVE auto-fail list.
`isMaliciousPackageBlockingEnabled`	Malicious package blocking.
`isLicenseCollectionEnabled`	License collection.

For a description of the threshold and filter parameters, see Scan output filters.

> **Note**
>
> The plugin still accepts the earlier parameter names `isThresholdEnabled` and `isEpssEnabled`, but logs a warning that suggests the current names, `isSeverityThresholdEnabled` and `isEpssThresholdEnabled`.

Scan output filters

The following options control when the scan fails the build. Each option is off by default. When you turn on more than one, the build fails if any of them fails. The console output of the build shows which options are on and why the build failed.

Enable severity thresholds (`isSeverityThresholdEnabled`)

Fails the build if the number of findings of a severity is greater than its threshold. You can set a threshold for **Critical** (`countCritical`), **High** (`countHigh`), **Medium** (`countMedium`), and **Low** (`countLow`). The counts include Dockerfile findings, and each finding counts once, at its highest severity rating. A threshold of `0` fails the build if the scan finds anything of that severity. Findings with other severities don't count toward the thresholds.

Enable EPSS threshold (`isEpssThresholdEnabled`, `epssThreshold`)

Fails the build if any vulnerability has an EPSS score greater than or equal to the threshold. Vulnerabilities without an EPSS score don't count. Enter a value from 0 to 1, such as `0.7`.

Enable CVE suppression list (`isSuppressedCveEnabled`, `suppressedCveList`)

Excludes the CVEs that you list from the severity thresholds and the EPSS threshold, for example false positives that you reviewed. Suppressed CVEs still appear in the reports. Separate CVE IDs with commas or line breaks, such as `CVE-2023-1234,CVE-2023-5678`.

Enable CVE auto-fail list (`isAutoFailCveEnabled`, `autoFailCveList`)

Fails the build if the scan finds any CVE that you list, regardless of your other settings, including the CVE suppression list. Use this list for vulnerabilities that must never be deployed. Separate CVE IDs with commas or line breaks, such as `CVE-2024-9999`.

Enable malicious package blocking (`isMaliciousPackageBlockingEnabled`)

Fails the build if Amazon Inspector identifies any malicious packages in the image, regardless of your other settings. If the scan result doesn't include malicious package information, the plugin

skips this check with a message in the console output and doesn't fail the build. This option is available in plugin version 649.v0fe6e5596568 and later. For information about how Amazon Inspector identifies malicious packages, see Amazon Inspector Security Research.

Enable license collection (`isLicenseCollectionEnabled`)

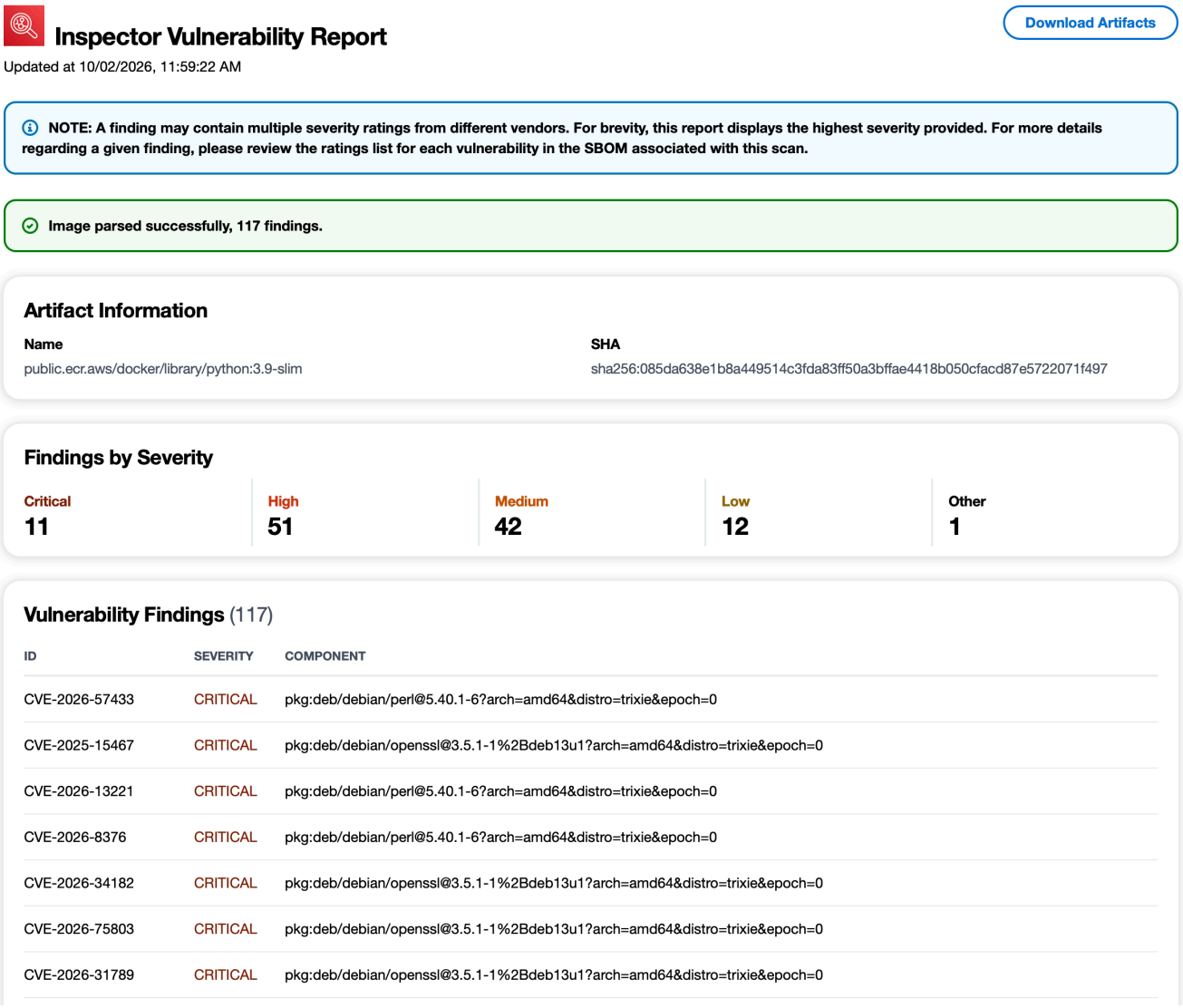
Adds package license information to the SBOM that the build archives. This option doesn't fail the build. With the **Manual** installation method, it requires Amazon Inspector SBOM Generator version 1.8.0 or later. For more information, see Amazon Inspector SBOM Generator license collection.

Step 7. View your Amazon Inspector vulnerability report

1. Run a build of your project.
2. Open the build. When the scan completes, the console output ends with a security assessment summary and whether the build passed.
3. Under **Build Artifacts**, choose `index.html` to open the HTML report. The build also archives the following files:

   - The SBOM that the Amazon Inspector SBOM Generator created, in JSON format.
   - A CSV file with the vulnerability findings, if the scan found any.
   - A CSV file with the Dockerfile findings, if the scan found any.

   The report shows the scanned image, the number of findings for each severity, and a table that lists each vulnerability with its ID, severity, and affected package. The following image shows an example report.



Inspector Vulnerability Report

Download Artifacts

Updated at 10/02/2026, 11:59:22 AM

NOTE: A finding may contain multiple severity ratings from different vendors. For brevity, this report displays the highest severity provided. For more details regarding a given finding, please review the ratings list for each vulnerability in the SBOM associated with this scan.

Image parsed successfully, 117 findings.

Artifact Information

Name
public.ecr.aws/docker/library/python:3.9-slim

SHA
sha256:085da638e1b8a449514c3fda83ff50a3bffae4418b050cfacd87e5722071f497

Findings by Severity

Critical	High	Medium	Low	Other
11	51	42	12	1

Vulnerability Findings (117)

ID	SEVERITY	COMPONENT
CVE-2026-57433	CRITICAL	pkg:deb/debian/perl@5.40.1-6?arch=amd64&distro=trixie&epoch=0
CVE-2025-15467	CRITICAL	pkg:deb/debian/openssl@3.5.1-1%2Bdeb13u1?arch=amd64&distro=trixie&epoch=0
CVE-2026-13221	CRITICAL	pkg:deb/debian/perl@5.40.1-6?arch=amd64&distro=trixie&epoch=0
CVE-2026-8376	CRITICAL	pkg:deb/debian/perl@5.40.1-6?arch=amd64&distro=trixie&epoch=0
CVE-2026-34182	CRITICAL	pkg:deb/debian/openssl@3.5.1-1%2Bdeb13u1?arch=amd64&distro=trixie&epoch=0
CVE-2026-75803	CRITICAL	pkg:deb/debian/openssl@3.5.1-1%2Bdeb13u1?arch=amd64&distro=trixie&epoch=0
CVE-2026-31789	CRITICAL	pkg:deb/debian/openssl@3.5.1-1%2Bdeb13u1?arch=amd64&distro=trixie&epoch=0

Troubleshooting

The following are common errors you can encounter when using the Amazon Inspector Jenkins plugin. If the plugin fails, the build fails, and the console output shows `Plugin execution failed.` followed by the error.

Unable to load credentials error

Error:

`software.amazon.awssdk.core.exception.SdkClientException: Unable to load credentials from any of the providers in the chain`, followed by the list of

credential providers that the plugin tried. If you didn't set an **IAM Role**, the last error can instead be `Profile file contained no credentials for profile 'default'`.

Earlier versions of the plugin can show this error as `InstanceProfileCredentialsProvider(): Failed to load credentials or sts exception.`

Resolution:

The plugin couldn't get AWS credentials on the Jenkins controller. Configure one of the authentication options in the build step, as described in Step 4 and Step 6. Alternatively, make credentials available to the controller, for example with an Amazon EC2 instance profile. If the scan request fails, the plugin tries once more without the access key, OIDC token, or profile name. If you set an **IAM Role**, it assumes the role with the controller's own credentials. Otherwise, it uses the `default` profile on the controller. Because of this retry, the first error in the console output usually shows the cause.

Failed to load image from tarball, local, or remote sources

Error:

`[ImageDownloadFailed]: failed to load image from tarball, local, or remote sources.`

Depending on the cause, the Amazon Inspector SBOM Generator can instead report a more specific error, such as `[ImageDownloadManifestUnknown]` or `[ImageRegistryAccessUnauthorized]`. The console output shows `Exception:com.amazon.inspector.jenkins.amazoninspectorbuildstep.exception.Malfor Sbom scanning output formatted incorrectly.`, followed by `Sbom Content:` and the Amazon Inspector SBOM Generator error.

Resolution:

Verify the following:

- The user that runs the build has read permissions to the image you want to scan.
- For a local image, the image is present in the Docker engine on the machine that runs the build.
- The image name is correct.
- For an image in a private repository, the build step uses the Docker credentials from Step 3.

Inspector-sbomgen path error

Error:

```
Exception:java.lang.IllegalArgumentException: Provided SBOMgen path is
invalid or not executable: /opt/inspector/inspector-sbomgen
```

Earlier versions of the plugin show the following error instead:
`Exception:com.amazon.inspector.jenkins.amazoninspectorbuildstep.exception.Sbomge`
`There was an issue running inspector-sbomgen, is /opt/inspector/inspector-sbomgen the correct path?`

Resolution:

Complete the following procedure to resolve the issue.

1. Download the Amazon Inspector SBOM Generator for the CPU architecture of the machine that runs the build. For more information, see Installing Sbomgen.
2. Make the binary executable by running the following command: `chmod +x inspector-sbomgen`.
3. In the build step, enter the full path to the binary, such as `/opt/inspector/inspector-sbomgen`. The plugin checks the path on the Jenkins controller, so if the build runs on an agent, the binary must be at the same path on both.
4. Save the configuration, and run the build again.

If the error is `Invalid sbomgen path: path`, the path contains a character that isn't allowed. The path can contain only letters, numbers, spaces, periods (.), underscores (_), hyphens (-), colons (:), and forward slashes (/).

Unsupported OS or unsupported architecture error

Error:

```
Exception:java.lang.UnsupportedOperationException: Unsupported OS: mac os x
```

The error can also be `Exception:java.lang.UnsupportedOperationException: Unsupported architecture: architecture`.

Resolution:

The Amazon Inspector SBOM Generator is available only for Linux with an x86_64 (amd64) or arm64 CPU. With the **Automatic** installation method, the plugin checks the operating system of the Jenkins controller and the CPU of the machine that runs the build. Run the controller on Linux, and run your build on Linux with a supported CPU.

HTML report has no styling or shows zero findings

Error:

The HTML report has no styling and shows zero findings. The top of the report shows `Jenkins may be blocking CSS/JS, visit https://www.jenkins.io/doc/book/security/configuring-content-security-policy/ for a possible fix.`

Resolution:

Jenkins is blocking the CSS and JavaScript in the report. To allow the report to display, see Step 5. Allow Jenkins to display the HTML report. The CSV files show the findings even when the HTML report doesn't display.

Using CodeCatalyst actions with Amazon Inspector

You can use Amazon Inspector with Amazon CodeCatalyst to add Amazon Inspector vulnerability scans to your CodeCatalyst workflows. This leverages the Amazon Inspector SBOM Generator and Amazon Inspector Scan API to produce detailed reports at the end of your build, so you can investigate and remediate risk before deployment. Amazon Inspector vulnerability scans can be configured to pass or fail workflows based on the number and severity of vulnerabilities detected. For information about how to integrate Amazon Inspector Scan into your CI/CD pipeline, see Integrating Amazon Inspector scans into your CI/CD pipeline. For a list of operating systems and programming languages that Amazon Inspector supports, see Supported operating systems and programming languages.

Using Amazon Inspector Scan actions with CodePipeline

You can use Amazon Inspector with AWS CodePipeline by adding vulnerability scans to your workflows. This integration leverages the Amazon Inspector SBOM Generator and Amazon Inspector Scan API to produce detailed reports at the end of your build. The integration helps you investigate and remediate risk before deployment. The `InspectorScan` action is a managed compute action in CodePipeline that automates detecting and fixing security vulnerabilities in

your open source code. You can use this action with application source code in your third-party repository, such as GitHub or Bitbucket Cloud, or with images for container applications. For more information, see InspectorScan invoke action reference in the *AWS CodePipeline User Guide*.

Using the Amazon Inspector TeamCity plugin

> **Note**
>
> The Amazon Inspector TeamCity plugin is discontinued and archived on GitHub.

The Amazon Inspector TeamCity plugin leverages the Amazon Inspector SBOM Generator binary and Amazon Inspector Scan API to produce detailed reports at the end of your build, so you can investigate and remediate risk before deployment. With the Amazon Inspector TeamCity plugin, you can add Amazon Inspector vulnerability scans to your TeamCity pipeline. Amazon Inspector vulnerability scans can be configured to pass or fail pipeline executions based on the number and severity of vulnerabilities detected. You can view the archived version of the Amazon Inspector TeamCity plugin on GitHub at https://github.com/aws/amazon-inspector-container-image-scanner-teamcity-plugin.

Using Amazon Inspector with GitLab components

> **Note**
>
> The Amazon Inspector GitLab component is not officially supported by Amazon. It is maintained and provided as open-source by GitLab.

You can use Amazon Inspector with GitLab CI/CD components to add Amazon Inspector vulnerability scans to your GitLab projects. This leverages the Amazon Inspector SBOM Generator and Amazon Inspector Scan API to produce detailed reports at the end of your build, so you can investigate and remediate risk before deployment. Amazon Inspector vulnerability scans can be configured to pass or fail workflows based on the number and severity of vulnerabilities detected. You can view the latest version of the Amazon Inspector component on the GitLab website.
For information about how to integrate Amazon Inspector Scan into your CI/CD pipeline, see Integrating Amazon Inspector scans into your CI/CD pipeline. For a list of operating systems and

programming languages that Amazon Inspector supports, see Supported operating systems and programming languages.

Assessing Amazon Inspector coverage of your AWS environment

You can assess Amazon Inspector coverage of your AWS environment from the **Account management** screen in the Amazon Inspector console, which shows details and statistics about the status of Amazon Inspector scans for your accounts and resources.

> **Note**
>
> If you're the delegated administrator for an organization, you can view details and statistics for all the accounts in the organization.

The following procedure describes how to assess coverage of your Amazon Inspector environment.

To assess Amazon Inspector coverage of your AWS environment

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Account management**.
3. To review coverage, choose one of the following tabs:

   - Choose **Accounts** to review account-level coverage.
   - Choose **Instances** to review coverage for Amazon Elastic Compute Cloud (Amazon EC2) instances.
   - Choose **Machine images** to review coverage for the machine images (AMIs) that your accounts own.
   - Choose **Container repositories** to review coverage of Amazon Elastic Container Registry (Amazon ECR) repositories.
   - Choose **Container images** to review coverage for Amazon ECR container images.
   - Choose **Lambda functions** to review coverage for Lambda functions.

The following topics describe the information each of these tabs provide.

Topics

- Assessing account-level coverage
- Assessing coverage of Amazon EC2 instances
- Assessing coverage of Amazon ECR repositories
- Assessing coverage of Amazon ECR container images
- Assessing coverage of AWS Lambda functions
- Assessing coverage of machine images

Assessing account-level coverage

If your account is not part of an organization or is not the delegated Amazon Inspector administrator account for an organization, the **Accounts** tab provides information about your account and the status of resource scanning for your account. On this tab, you can activate or deactivate scanning for all or only specific types of resources for your account. For more information, see Automated scan types in Amazon Inspector.

If your account is the delegated Amazon Inspector administrator account for an organization, the **Accounts** tab provides automatic activation settings for accounts in your organization, and it lists all the accounts in your organization. For each account, the list indicates whether Amazon Inspector is activated for the account and, if so, the resource scanning types that are activated for the account. As the delegated administrator, you can use this tab to change the automatic activation settings for your organization. You can also activate or deactivate specific types of resource scanning for individual member accounts. For more information, see Activating Amazon Inspector scans for member accounts.

Assessing coverage of Amazon EC2 instances

The **Instances** tab shows Amazon EC2 instances in your AWS environment. The lists are organized into groups on the following tabs:

- **All** – Shows all the instances in your environment. The **Status** column indicates the current scanning status for an instance.
- **Scanning** – Shows all the instances that Amazon Inspector is actively monitoring and scanning in your environment.
- **Not scanning** – Shows all the instances that Amazon Inspector is not monitoring and scanning in your environment. The **Reason** column indicates why Amazon Inspector is not monitoring and scanning an instance.

An EC2 instance can appear on the **Not scanning** tab for any of several reasons. Amazon Inspector uses AWS Systems Manager (SSM) and the SSM Agent to automatically monitor and scan your EC2 instances for vulnerabilities. If an instance does not have the SSM Agent running, does not have an AWS Identity and Access Management (IAM) role that supports Systems Manager, or is not running a supported operating system or architecture, Amazon Inspector cannot monitor and scan the instance. For more information, see Amazon EC2 instance scanning.

On each tab, the **Account** column specifies the AWS account that owns an instance.

EC2 instance tags – This column shows you the tags associated with the instance and can be used to determine if your instance has been excluded from scans by tags.

Operating system – This column shows you the operating system type, which can be `WINDOWS`, `MAC`, `LINUX`, or `UNKNOWN`.

Monitored using – This column shows whether Amazon Inspector is using the agent-based or agentless scan method on this instance.

Last scanned – This column shows you when Amazon Inspector last checked that resource for vulnerabilities. The frequency that Amazon Inspector performs scans depends on the scan method it's using to scan the instance.

To review additional details about an EC2 instance, choose the link in the **EC2 instance** column. Amazon Inspector then displays details about the instance and current findings for the instance. To review the details of a finding, choose the link in the **Title** column. For information about these details, see Viewing details for your Amazon Inspector findings.

Scanning status values for Amazon EC2 instances

For an Amazon Elastic Compute Cloud (Amazon EC2) instance, the possible **Status** values are:

- **Actively monitoring** – Amazon Inspector is continuously monitoring and scanning the instance.
- **Agentless instance storage limit exceeded** – Amazon Inspector uses this status when the combined size of all volumes attached to an instance is greater than 1200 GB, or an instance has more than 8 volumes attached to it.
- **Agentless instance collection time limit exceeded** – Amazon Inspector times out while trying to run an agentless scan on an instance.

- **EC2 instance stopped** – Amazon Inspector paused scanning for the instance because the instance is in a stopped state. Any existing findings will persist until the instance is terminated. If the instance is restarted, Amazon Inspector will automatically resume scanning for the instance.
- **Internal error** – An internal error occurred when Amazon Inspector attempted to scan the instance. Amazon Inspector will automatically address the error and resume scanning as soon as possible.
- **No inventory** – Amazon Inspector couldn't find the software application inventory to scan for the instance. The Amazon Inspector associations for the instance might have been deleted or they might have failed to run.

  To remediate this issue, use AWS Systems Manager to ensure that the `InspectorInventoryCollection-do-not-delete` association exists and its association status is successful. In addition, use AWS Systems Manager Fleet Manager to verify the software application inventory for the instance.
- **Pending disable** – Amazon Inspector has stopped scanning the instance. The instance is being disabled, pending completion of clean-up tasks.
- **Pending initial scan** – Amazon Inspector has queued the instance for an initial scan.
- **Resource terminated** – The instance was terminated. Amazon Inspector is currently cleaning up existing findings and coverage data for the instance.
- **Stale inventory** – Amazon Inspector wasn't able to collect an updated software application inventory that was captured within the past 7 days for the instance.

  To remediate this issue, use AWS Systems Manager to ensure that the required Amazon Inspector associations exist and are running for the instance. In addition, use AWS Systems Manager Fleet Manager to verify the software application inventory for the instance.
- **Unmanaged EC2 instance** – Amazon Inspector isn't monitoring or scanning the instance. The instance isn't managed by AWS Systems Manager.

  To remediate this issue, you can use the AWSSupport-TroubleshootManagedInstance runbook provided by AWS Systems Manager Automation. After you configure AWS Systems Manager to manage the instance, Amazon Inspector will automatically begin to continuously monitor and scan the instance.
- **Unsupported OS** – Amazon Inspector isn't monitoring or scanning the instance. The instance uses an operating system or architecture that Amazon Inspector doesn't support. For a list of operating systems that Amazon Inspector supports, see Amazon EC2 instances status values.

- **Actively monitoring with partial errors** – This status means that EC2 scanning is active, but there are errors associated with Amazon Inspector deep inspection for Linux-based Amazon EC2 instances. The possible deep inspections errors are:
  - **Deep inspection package collection limit exceeded** – The instance has exceeded the 5000 package limit for Amazon Inspector deep inspection. To resume deep inspection for this instance, you can try to adjust the custom paths associated with the account.
  - **Deep inspection daily ssm inventory limit exceeded** – The SSM agent couldn't send inventory to Amazon Inspector because the SSM quota for *Inventory data collected per instance per day* has already been reached for this instance. For more information, see Amazon EC2 Systems Manager endpoints and quotas.
  - **Deep inspection collection time limit exceeded** – Amazon Inspector failed to extract the package inventory because the package collection time exceeding the maximum threshold of 15 minutes.
  - **Deep inspection has no inventory** – The Amazon Inspector SSM plugin hasn't yet been able to collect an inventory of packages for this instance. This is usually the result of a pending scan, however, if this status persists after 6 hours, use Amazon EC2 Systems Manager to ensure that the required Amazon Inspector associations exist and are running for the instance. This error can also occur if the instance is missing the required `ssm:PutInventory` and `ssm:GetParameter` permissions. If the instance has an IAM instance profile, verify these permissions are included in the profile. If no instance profile is configured, verify these permissions are included in the Default Host Management Configuration role.

For details about configuring the scanning settings for an EC2 instance, see Amazon EC2 instance scanning.

Assessing coverage of Amazon ECR repositories

The **Repositories** tab shows Amazon ECR repositories in your AWS environment. The lists are organized into groups on the following tabs:

- **All** – Shows all the repositories in your environment. The **Status** column indicates the current scanning status for a repository.
- **Activated** – Shows all the repositories that Amazon Inspector is configured to monitor and scan in your environment. The **Status** column indicates the current scanning status for a repository.

- **Not activated** – Shows all the repositories that Amazon Inspector is not monitoring and scanning in your environment. The **Reason** column indicates why Amazon Inspector is not monitoring and scanning a repository.

On each tab, the **Account** column specifies the AWS account that owns a repository.

To review additional details about a repository, choose the repository's name. Amazon Inspector then displays a list of container images in the repository and details for each image. The details include the image tag, image digest, and scanning status. They also include key finding statistics, such as the number of **Critical** findings for the image. To drill down and review supporting data for finding statistics, choose the image tag for the image.

> **Note**
>
> Amazon ECR images without continuous scanning are not included in coverage widgets.

Scanning status values for Amazon ECR repositories

For an Amazon Elastic Container Registry (Amazon ECR) repository, the possible **Status** values are:

- **Activated (Continuous)** – For a repository, Amazon Inspector is continuously monitoring images in this repository. The enhanced scanning setting for the repository is set to continuous scanning. Amazon Inspector initially scans new images when they are pushed and rescans images if a new CVE relevant to that image is published. Amazon Inspector will continue to be monitor images in this repository for the Amazon ECR re-scan duration you configure.
- **Activated (On push)** – Amazon Inspector automatically scans individual container images in the repository when a new image is pushed. Enhanced scanning is activated for the repository and set to scan on push.
- **Access denied** – Amazon Inspector isn't allowed to access the repository or any container images in the repository.

  To remediate this issue, ensure that AWS Identity and Access Management (IAM) policies for the repository allow Amazon Inspector to access the repository.
- **Deactivated (Manual)** – Amazon Inspector isn't monitoring or scanning any container images in the repository. The Amazon ECR scanning setting for the repository is set to basic, manual scanning.

To start scanning images in the repository with Amazon Inspector, change the scanning setting for the repository to enhanced scanning, and then choose whether to scan images continuously or only when a new image is pushed.

- **Activated (On push)** – Amazon Inspector automatically scans individual container images in the repository when a new image is pushed. The enhanced scanning setting for the repository is set to scan on push.
- **Internal error** – An internal error occurred when Amazon Inspector attempted to scan the repository. Amazon Inspector will automatically address the error and resume scanning as soon as possible.

For details about configuring the scanning settings for repositories Amazon ECR container image scanning.

Assessing coverage of Amazon ECR container images

The **Images** tab shows Amazon ECR container images in your AWS environment. The lists are organized into groups on the following tabs:

- **All** – Shows all the container images in your environment. The **Status** column indicates the current scanning status for an image.
- **Scanning** – Shows all the container images that Amazon Inspector is configured to monitor and scan in your environment. The **Status** column indicates the current scanning status for an image.
- **Not scanning** – Shows all the container images that Amazon Inspector is not monitoring and scanning in your environment. The **Reason** column indicates why Amazon Inspector is not monitoring and scanning an image.

  A container image can appear on the **Not activated** tab for any of several reasons. The image might be stored in a repository that Amazon Inspector scans are not activated for, or Amazon ECR filtering rules prevent that repository from being scanned. Or the image has not been pushed or pulled within the number of days your configured for the **ECR re-scan duration**. For more information, see Configuring the Amazon ECR re-scan duration.

On each tab, the **Repository name** column specifies the name of the repository that stores a container image. The **Account** column specifies the AWS account that owns the repository. The **Last scanned** column shows you when Amazon Inspector last checked that resource for vulnerabilities.

This can include checks when there is an update to finding metadata, when there is an update to the application inventory of the resource, or when a rescan is done in response to a new CVE. For more information, see Scan behaviors for Amazon ECR scanning.

To review additional details about a container image, choose the link in the **ECR container image** column. Amazon Inspector then displays details about the image and current findings for the image. To review the details of a finding, choose the link in the **Title** column. For information about these details, see Viewing details for your Amazon Inspector findings.

Scanning status values for Amazon ECR container images

For an Amazon Elastic Container Registry container image, the possible **Status** values are:

- **Actively monitoring (Continuous)** – Amazon Inspector is continuously monitoring and the image and new scans are performed on it whenever a new relevant CVE is published. The Amazon ECR rescan duration for the image is refreshed whenever the image is pushed or pulled. Enhanced scanning is enabled for the repository that stores the image, and the enhanced scanning setting for the repository is set to continuous scanning.
- **Activated (On push)** – Amazon Inspector automatically scans the image each time a new image is pushed. Enhanced scanning is activated for the repository that stores the image, and the enhanced scanning setting for the repository is set to scan on push.
- **Internal error** – An internal error occurred when Amazon Inspector attempted to scan the container image. Amazon Inspector will automatically address the error and resume scanning as soon as possible.
- **Pending initial scan** – Amazon Inspector has queued the image for an initial scan.
- **Scan eligibility expired (Continuous)** – Amazon Inspector suspended scanning for the image. The image hasn't been updated within the duration that you specified for automated re-scans of images in the repository. You can push or pull the image to resume scanning.
- **Scan eligibility expired (On push)** – Amazon Inspector suspended scanning for the image. The image hasn't been updated within the duration that you specified for automated re-scans of images in the repository. You can push the image to resume scanning.
- **Scan frequency manual (Manual)** – Amazon Inspector doesn't scan the Amazon ECR container image. The Amazon ECR scanning setting for the repository that stores image is set to basic, manual scanning. To start scanning the image automatically with Amazon Inspector, change the repository setting to enhanced scanning, and then choose whether to scan images continuously or only when a new image is pushed.

- **Unsupported OS** – Amazon Inspector isn't monitoring or scanning the image. The image is based on an operating system that Amazon Inspector doesn't support, or it uses a media type that Amazon Inspector doesn't support.

  For a list of operating systems that Amazon Inspector supports, see Supported operating systems: Amazon ECR scanning with Amazon Inspector. For a list of media types that Amazon Inspector supports, see Supported media types.

For details about configuring the scanning settings for repositories and images, see Amazon ECR container image scanning.

Assessing coverage of AWS Lambda functions

The **Lambda** tab shows Lambda functions in your AWS environment. This page two tables, one that shows function coverage details for Lambda standard scanning and another for Lambda code scanning. You can group functions based on the following tabs:

- **All** – Shows all the Lambda functions in your environment. The **Status** column indicates the current scanning status for a Lambda function.
- **Scanning** – Shows the Lambda functions that Amazon Inspector is configured to scan. The **Status** column indicates the current scanning status for each Lambda function.
- **Not scanning** – Shows the Lambda functions that Amazon Inspector is not configured to scan. The **Reason** column indicates why Amazon Inspector is not monitoring and scanning a function.

  A Lambda function can appear on the **Not scanning** tab for several reasons. The Lambda function might belong to an account that hasn't been added to Amazon Inspector or filtering rules prevent this function from being scanned. For more information, see Lambda function scanning.

On each tab, the **Function name** column specifies the name of the Lambda function. The **Account** column specifies the AWS account that owns the function. **Runtime** specifies the function's runtime. The **Status** column indicates the current scanning status for each Lambda function. **Resource tags** shows the tags that have been applied to the function. The **Last scanned** column shows you when Amazon Inspector last checked that resource for vulnerabilities. This can include checks when there is an update to finding metadata, when there is an update to the application inventory of the resource, or when a rescan is done in response to a new CVE. For more information, see Scan behaviors for Lambda function scanning.

Scanning status values for AWS Lambda functions

For a Lambda function, the possible **Status** values are:

- **Actively monitoring** – Amazon Inspector is continuously monitoring and scanning Lambda functions. Continuous scanning includes an initial scan of new functions when they are pushed to the repository and automated re-scans of functions when they are updated or when new Common Vulnerabilities and Exposures (CVEs) are released.
- **Excluded by tag**– Amazon Inspector isn't scanning this function because it has been excluded from scans by tags.
- **Scan eligibility expired**– Amazon Inspector is not monitoring this function because it has been 90 days or more since it was last invoked or updated.
- **Internal error**–An internal error occurred when Amazon Inspector attempted to scan the function. Amazon Inspector will automatically address the error and resume scanning as soon as possible.
- **Pending initial scan**– Amazon Inspector has queued the function for an initial scan.
- **Unsupported**– The Lambda function has an unsupported runtime.

Assessing coverage of machine images

The **Machine images** tab shows the machine images (AMIs) that your accounts own. You can group machine images based on the following tabs:

- **All** – Shows all the machine images that your accounts own. The **Status** column indicates the current scanning status for a machine image.
- **Scanning** – Shows the machine images that Amazon Inspector is configured to scan. The **Status** column indicates the current scanning status for each machine image.
- **Not scanning** – Shows the machine images that Amazon Inspector is not configured to scan. The **Reason** column indicates why Amazon Inspector is not monitoring and scanning a machine image.

  A machine image can appear on the **Not scanning** tab for several reasons. Your scan configuration might exclude the image, its operating system or file system might not be supported, or it might fall outside of your scan eligibility windows. For more information, see Machine image scanning.

On each tab, the **Machine image** column specifies the ID of the machine image. The **Account** column specifies the AWS account that owns the machine image. **Operating system** specifies the platform of the machine image. The **Status** column indicates the current scanning status for each machine image. **Resource tags** shows the tags that have been applied to the machine image. The **Last scanned** column shows you when Amazon Inspector last checked that machine image for vulnerabilities. For more information, see Scan behaviors for machine image scanning.

Scanning status values for machine images

For a machine image, the possible **Status** values are:

- **Actively monitoring** – Amazon Inspector is continuously monitoring and scanning the machine image. Continuous scanning includes an initial scan when you create, copy, register, or enable the machine image, and automated rescans when new Common Vulnerabilities and Exposures (CVEs) are released.
- **Pending initial scan** – Amazon Inspector has queued the machine image for an initial scan.
- **Excluded by tag** – Amazon Inspector isn't scanning this machine image because your scan mode or your exclusion options exclude it. In exclusion (`SCAN_ALL`) mode, the image carries your configured exclusion tag. Amazon Inspector also reports this reason when the image is a copy of another image or Amazon Data Lifecycle Manager or AWS Backup created it, and you chose to exclude copied or backup images. For more information, see Managing machine image scan mode and Excluding copied and backup machine images.
- **Scan eligibility expired** – Amazon Inspector is not monitoring this machine image because it falls outside of both of your scan eligibility windows. For more information, see Setting scan eligibility windows.
- **Access denied** – Amazon Inspector couldn't access the machine image or the snapshots that back it.
- **Unsupported operating system** – The operating system of the machine image isn't supported for scanning, or one or more of the snapshots that back the image uses a configuration that Amazon Inspector can't read, such as Logical Volume Manager (LVM).
- **Collection time limit exceeded** – Amazon Inspector timed out while collecting the software inventory from the snapshots that back the machine image.
- **Image size exceeded** – The combined size of the snapshots that back the machine image exceeds the limit that Amazon Inspector supports for scanning.

- **Internal error** – An internal error occurred when Amazon Inspector attempted to scan the machine image. Amazon Inspector will automatically address the error and resume scanning as soon as possible.

Managing multiple accounts in Amazon Inspector with AWS Organizations

You can use Amazon Inspector to manage multiple accounts in an organization. Amazon Inspector supports two approaches for multi-account management:

- **Delegated administrator for AWS Organizations policies** - Provides centralized governance to delegated administrator with automatic enablement of Amazon Inspector across organization accounts across regions. Organization policies enforce which scan types are enabled and take precedence over non policy managed delegated administrator and member account enablements.
- **Delegated administrator for non AWS Organizations policy** - An account designated to manage Amazon Inspector for the organization without using organization policies. The delegated administrator can enable Amazon Inspector for member accounts and configure scan settings.

These approaches can be used together. When organization policies are in place, they control resource type enablement (which scan types are enabled), while delegated administrators retain control over scan configuration settings such as scan modes and deep inspection paths. The following topics describe these management approaches, how to designate a delegated administrator, and how to manage member accounts.

Topics

- Understanding the delegated administrator account and member account in Amazon Inspector
- Designating a delegated administrator account for Amazon Inspector

Understanding the delegated administrator account and member account in Amazon Inspector

When using Amazon Inspector in a multi-account environment, the delegated administrator account has access to specific metadata. The metadata includes standard scanning for Amazon EC2, Amazon ECR, and Lambda, and Lambda code scanning. It also includes security finding results for member accounts. This section provides information about which actions the delegated adminstrator account can make and member accounts can make.

Organization policy governance model

When AWS Organizations policies are used to enable Amazon Inspector, a governance model is enforced that determines which actions are permitted:

Policy-managed resources

Resources explicitly enabled or disabled by organization policies cannot be modified by delegated administrators or member accounts. API requests to enable or disable policy-managed scan types will fail with a clear error indicating the resource is managed by organization policy.

Non-policy-managed resources

Resources not specified in organization policies can be managed normally by delegated administrators and member accounts using the Amazon Inspector console or API.

Scan configuration management

Delegated administrators can always configure scan settings such as EC2 scan modes, deep inspection paths, and ECR re-scan durations, regardless of whether the resource types are policy-managed. Organization policies control only whether scanning is enabled, not how it operates.

For more information about creating and managing Amazon Inspector organization policies, see the AWS Organizations documentation for Amazon Inspector policies.

Delegated administrator actions

Generally, when the delegated administrator applies settings to their account, those settings are applied to all of the other accounts in the organization. The delegated administrator can also view and retrieve information for their own account and any associated member. An Amazon Inspector delegated administrator account can perform the following actions:

- Only the AWS Organizations management account can designate and remove a delegated administrator.
- When designating a delegated administrator, you must be in the same organization as the member accounts you want to manage.
- View and manage the status of Amazon Inspector for associated accounts, including activating and deactivating Amazon Inspector.

- Activate or deactivate scanning types for all member accounts in the organization.
- View aggregated finding data across the organization and finding details for all member accounts within the organization.
- Create and manage suppression rules that apply to findings for all accounts in the organization.
- Activate Amazon ECR enhanced scanning for all members of the organization.
- View resource coverage for the entire organization.
- Define the duration for automated re-scans of ECR container images for all member accounts in the organization. The delegated administrator's scan duration setting overrides any setting that the member account previously set. All accounts in the organization share the Amazon ECR automated re-scan duration of the delegated administrators. You can't set different re-scan durations for individual accounts.
- Specify five custom paths for Amazon Inspector deep inspection for Amazon EC2 that will be used across all accounts in the organization. This is in addition to the five custom paths that a delegated administrator can set for their individual account. For more information about configuring deep inspection custom paths, see Custom paths for Amazon Inspector deep inspection.
- Activate and deactivate Amazon Inspector deep inspection for member accounts.
- Export SBOMs for any member accounts in the organization.
- Set the Amazon EC2 scan mode for all member accounts in the organization. For more information, see Managing scan mode.
- Create and manage CIS scan configurations for all accounts in the organization, except for any scan configurations created by member accounts.

> **Note**
>
> If a member account leaves the organization, the delegated administrator will no longer be able to see scan configurations scheduled by that account.

- View CIS scan results for all accounts in the organization.
- When organization policies are in use, configure scan settings for policy-managed resources but cannot enable or disable the policy-managed scan types themselves.

Member account actions

A member account can view and retrieve information about their account in Amazon Inspector, while settings for their account are managed by the delegated administrator. Member accounts within an organization can perform the following actions in Amazon Inspector:

- Activate Amazon Inspector for their own account.
- View resource coverage for their own account.
- View findings details for their own account.
- View the ECR container image automated re-scan duration setting for their own account.
- Specify five custom paths for Amazon Inspector deep inspection for EC2 that will be used for their individual account. These paths are scanned in addition to any custom paths that the delegated administrator has specified for the organization. For more information about configuring deep inspection paths, see Custom paths for Amazon Inspector deep inspection.
- View the custom paths set by your delegated administrator for Amazon Inspector deep inspection.
- Export SBOMs for any resources associated with their account.
- View the scan mode for their account.
- Create and manage CIS scan configurations for their account.
- View the results of any CIS scans for resources in their account, including those scheduled by the delegated administrator.
- Enable scan types that are not managed by organization policies. Policy-managed scan types cannot be enabled or disabled by member accounts.

> **Note**
>
> After activation, Amazon Inspector can be deactivated only by a delegated administrator account.

Designating a delegated administrator account for Amazon Inspector

The delegated administrator is an account that manages a service for an organiztion. This topic describes how to designate a delegated administrator for Amazon Inspector.

Considerations

Before designating a delegated administrator, note the following:

The delegated administrator can manage a maximum of 10,000 members.

If you exceed 10,000 member accounts, you receive a notification through the Amazon CloudWatch Personal Health Dashboard and email to the delegated administrator account.

> **Note**
>
> When Amazon Inspector is enabled through AWS Organizations policies for organizations with more than 10,000 accounts (up to 50,000), the policy applies to all accounts. However, only 10,000 accounts will be associated with the Amazon Inspector organization. i.e. the delegated administrator can view findings and account status for only these 10,000 accounts in the Amazon Inspector console.

The delegated administrator is Regional.

Amazon Inspector is a Regional service. You must repeat the steps in the procedure in every AWS Region where you plan to use Amazon Inspector.

An organization can have only one delegated administrator.

If designate an account as the delegated administrator in one AWS Region, that account must be the delegated administrator in all other AWS Regions.

Changing a delegated administrator does not deactivate Amazon Inspector for member accounts.

If you remove a delegated administrator, member accounts become standalone accounts and scan settings aren't affected.

Your AWS Organization must have all features activated.

This is the default setting for AWS Organizations. If it's not activated, see Activating all features in your organization.

Organization policies take precedence over delegated administrator settings.

If your organization uses AWS Organizations policies to enable Amazon Inspector, the policy settings determine which scan types are enabled. We recommend designating the delegated administrator before creating organization policies to ensure consistent governance. For more information, see Organization policy governance model.

Permissions required to designate a delegated administrator

You must have permission to activate Amazon Inspector and to designate an Amazon Inspector delegated administrator. Add the following statement to the end of your IAM policy to grant these permissions. For more information, see Managing IAM policies.

```
{
    "Sid": "PermissionsForInspectorAdmin",
    "Effect": "Allow",
    "Action": [
        "inspector2:EnableDelegatedAdminAccount",
        "organizations:EnableAWSServiceAccess",
        "organizations:RegisterDelegatedAdministrator",
        "organizations:ListDelegatedAdministrators",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:DescribeOrganizationalUnit",
        "organizations:DescribeAccount",
        "organizations:DescribeOrganization"
    ],
    "Resource": "*"
}
```

Designating a delegated administrator for your AWS organization

The following procedure describes how to designate a delegated administrator for your organization. Before you complete the procedure, make sure you are in the same organization as the member accounts you want the delegated administrator to manage.

> **Note**
>
> You must use the AWS Organizations management account to complete this procedure. Only the AWS Organizations management account can designate a delegated administrator. Permissions might be required to designate a delegated administrator. For more information, see Permissions required to designate a delegated administrator.

When you activate Amazon Inspector for the first time, Amazon Inspector creates the service linked role `AWSServiceRoleForAmazonInspector` for the account. For information about how Amazon Inspector uses service-linked roles, see Using service-linked roles for Amazon Inspector.

Console

To designate a delegated administrator for Amazon Inspector

1. Sign in to the AWS Organizations management account, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the AWS Region selector to specify the AWS Region where you want to designate the delegated administrator.
3. From the navigation pane, choose **General settings**.
4. Under **Delegated administrator**, enter the 12-digit ID of the AWS account you want to designate as the delegated administrator.
5. Choose **Delegate**, and then choose **Delegate** again.

When you designate a delegated administrator, all scan types are activated for the account by default. If you want to activate Amazon Inspector for the AWS Organizations management account, complete the following procedure.

To activate Amazon Inspector for the AWS Organizations management account

1. Sign in to the delegated administrator account, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. From the navigation pane, choose **Account management**.
3. Under **Accounts**, select the AWS Organizations management account, and then choose **Activate**.

4. Select which scan types you want to activate for the AWS Organizations management account, and then choose **Submit**.

API

Designate a delegated administrator using the API

- Run the EnableDelegatedAdminAccount API operation using the credentials of the AWS account of the Organizations management account. You can also use the AWS Command Line Interface to do this by running the following CLI command:`aws inspector2 enable-delegated-admin-account --delegated-admin-account-id 11111111111`.

> **Note**
>
> Make sure to specify the account ID of the account that you want to make an Amazon Inspector delegated administrator.

Activating Amazon Inspector scans for member accounts

You can activate Amazon Inspector for member accounts in your organization through multiple methods. The method you choose depends on your governance requirements and organizational structure.

AWS Organizations policies (Recommended for centralized governance)

Use AWS Organizations policies to automatically enable Amazon Inspector across your organization with centralized control. This approach ensures consistent scanning coverage and automatically applies to new accounts. For detailed instructions, see the AWS Organizations documentation for creating Amazon Inspector policies.

Delegated administrator activation

As the delegated administrator, you can manually activate Amazon Inspector for specific member accounts or all member accounts through the Amazon Inspector console or API. This approach provides flexibility when organization policies are not in use.

Member account self-activation

Member accounts can activate Amazon Inspector for their own account when not restricted by organization policies. Once activated, the account becomes associated with the delegated administrator.

Activate scanning for member accounts

The following procedures describe how to activate scanning for member accounts using the delegated administrator and member account methods. For information about Amazon Inspector scanning types, see Automated scan types in Amazon Inspector.

To automatically activate scanning for all member accounts

1. Sign in using the delegated administrator account credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the region selector to choose the AWS Region where you want to activate scanning for all member accounts.
3. From the navigation pane, choose **Account management**. The **Accounts** tab displays all member accounts associated with the AWS Organizations management account.
4. Under **Organization**, select the box next to **Account number**. Then choose **Activate** to select which scanning options you want to apply to member accounts. You can select the following scanning types:

   - Amazon EC2 scanning
   - Amazon ECR scanning
   - Lambda standard scanning
   - Lambda code scanning

   - After you select your preferred scanning types, choose **Save**.

> **Note**
>
> If you have multiple pages of accounts, you must repeat this step on each page. You can choose the gear icon to change the number of accounts displayed on each page.

5. Turn on the **Automatically activate Inspector for new member accounts** setting, and select which scanning options you want to apply to new member accounts added to your organization. You can select the following scanning types:

   - Amazon EC2 scanning
   - Amazon ECR scanning
   - Lambda standard scanning
   - Lambda code scanning

   - After you select your preferred scanning types, choose **Activate**.

   > **Note**
   >
   > The **Automatically activate Inspector for new member accounts** setting activates Amazon Inspector for all future members of your organization.
   > If the number of member accounts is more than 5,000, this setting is automatically turned off. If the total number of member accounts decreases to less than 5,000, the setting is automatically reactivated.

6. (Recommended) Repeat each of these steps in each AWS Region where you want to activate scanning for member accounts.

To activate scanning for specific member accounts

1. Sign in using the delegated administrator account credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the region selector to choose the AWS Region where you want to activate scanning for all member accounts.
3. From the navigation pane, choose **Account management**. The **Accounts** tab displays all member accounts associated with the AWS Organizations management account.
4. Under **Organization**, select the box next to each member account number you want to activate scanning for. Then choose **Activate** to select which scanning options you want to apply to member accounts. You can select the following scanning types:

   - Amazon EC2 scanning

- Amazon ECR scanning
- Lambda standard scanning
- Lambda code scanning

- After you select your preferred scanning types, choose **Save**.

> **Note**
>
> If you have multiple pages of accounts, you must repeat this step on each page. You can choose the gear icon to change the number of accounts displayed on each page.

5. (Recommended) Repeat each of these steps in each AWS Region where you want to activate scanning for specific members.

To activate scanning as a member account

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the region selector to choose the AWS Region where you want to activate scanning for all member accounts.
3. From the navigation pane, choose **Account management**. The **Accounts** tab displays all member accounts associated with the AWS Organizations management account.
4. Under **Organization**, select the box next to your account number. Then choose **Activate** to select which scanning options you want to apply. You can select the following scanning types:

   - Amazon EC2 scanning
   - Amazon ECR scanning
   - Lambda standard scanning
   - Lambda code scanning

   - After you select your preferred scanning types, choose **Save**.
5. (Recommended) Repeat these steps in each Region where you want to activate scanning for your member account.

> **Note**
>
> If your AWS Organizations management account has a delegated administrator account for Amazon Inspector, you can activate your account as a member account to view scan details.

> **Important**
>
> If organization policies are managing Amazon Inspector enablement for your accounts, the delegated administrator and member accounts cannot modify policy-managed scan types using Amazon Inspector enablement/disablement APIs. API requests will fail with an error indicating the resource is managed by organization policy. You can still enable additional scan types not managed by the policy.

Disassociating member accounts in Amazon Inspector

As the delegated administrator, you might need to disassociate a member account from your account. When you disassociate a member account, Amazon Inspector is still activated in the account, and the account becomes a standalone account. You also don't have permission to manage Amazon Inspector for the account anymore. However, you can associate previously disassociated member accounts with your account at any time. This section describes how to disassociate member accounts as the delegated administrator.

> **Note**
>
> To disassociate policy-managed accounts, there should be no Amazon Inspector organization policy attached to that account for the scan type.

Console

To disassociate member accounts using the console

1. Sign in using the delegated administrator account credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home

2. Use the region selector to choose the AWS Region where you want to disassociate member accounts.
3. From the navigation pane, choose **Account management**.
4. Under **Organization**, select the box next to each account number you want to disassociate.
5. Choose **Actions** menu, and then choose **Disassociate account**.

API

To disassociate member accounts using the API

Run the DisassociateMember API operation. In the request, provide the account IDs you're disassociating.

Removing the delegated administrator in Amazon Inspector

You might need to remove the Amazon Inspector delegated administrator account. You can do this from the AWS Organizations management account. When you remove the Amazon Inspector delegated administrator account, Amazon Inspector is still activated in the account and in all of its member accounts. The delegated administrator account and all of its member accounts become standalone accounts and retain their original scan settings.

> **Note**
>
> If AWS Organizations policies are managing Amazon Inspector enablement, removing the delegated administrator does not affect policy enforcement. Accounts will remain enabled according to the organization policy settings, though member account findings will no longer be visible in a central delegated administrator console until a new delegated administrator is designated.

This section describes how to remove the delegated administrator account.

Remove the Amazon Inspector delegated administrator

The following procedures describe how to remove the Amazon Inspector delegated administrator and how to associate member accounts from the delegated administrator account.

For information about how to assign an Amazon Inspector delegated admninistrator, see Designating a delegated administrator account for Amazon Inspector.

> **Note**
>
> After you assign an Amazon Inspector delegated administrator, the Amazon Inspector delegated administrator must associate member accounts manually.

To remove the delegated administrator

1. Sign in to the AWS Management Console using the AWS Organizations management account.
2. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
3. Use the region selector to choose the AWS Region where you want to remove the delegated administrator.
4. From the navigation pane, choose **General settings**.
5. Under **Delegated administrator**, choose **Remove**, and then confirm your action.

To associate members with a new delegated administrator

1. Sign in using the delegated administrator account credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. Use the region selector to choose the AWS Region where you want to associate members.
3. From the navigation pane, choose **Account management**.
4. Under **Organization**, select the box next to **Account number**.
5. Choose **Actions**, and then choose **Add member**.

Tagging Amazon Inspector resources

A tag is a label you add to an AWS resource. Tags help you categorize AWS resources based on specific criteria. Tags consist of a key-value pair. The tag key is a general label. The tag value is a description of the tag key. With Amazon Inspector, you can tag suppression rules and CIS scan configurations. You can add as many as 50 tags to each of your Amazon Inspector resources.

Tagging fundamentals

A tag consists of a key-value pair. The tag key is a general label. The tag value is a description of the tag key. This topic descibes the fundamentals of tagging Amazon Inspector resources. When tagging Amazon Inspector resources, consider the following:

- You can tag suppression rules and CIS scan configurations.
- You can add as many as 50 tags to each of your Amazon Inspector resources.
- Tag keys must be unique.
- A tag key can only have one tag value.
- Tag keys and tag values can have a maximum of 128 UTF-8 characters. The characters can be letters, numbers, spaces, or the following symbols: `_ . : / = + - @`.
- You cannot use the `aws` prefix in any of your tags or modify tags with this prefix. Tags with the `aws` prefix are reserved for use by AWS.
- Tags assigned to an Amazon Inspector resource are only available in your AWS account and in the AWS Region where you created them.
- When you delete a resource, all tags associated with it are deleted, too.

For more information about tags, see Best practices and strategies in the *Tagging AWS Resources and Tag Editor User Guide*.

> **Note**
>
> Tags are not intended to store confidential or sensitive information. Never use tags to store this type of data. Tags can be accessible from other AWS services.

Adding tags

You can add tags to Amazon Inspector resources. These resources include suppression rules and CIS scan configurations. Tags help you categorize AWS resources based on specific criteria. This topic describes how to add tags to Amazon Inspector resources.

Adding tags to Amazon Inspector resources

You can tag suppression rules and CIS scan configurations. The following procedures describe how to add tags in the console and with the Amazon Inspector API.

Adding tags in the console

You can add tags to Amazon Inspector resources in the console.

Adding tags to suppression rules

You can add tags to suppression rules during creation. For more information, see Creating a suppression rule.

You can also edit a suppression rule to include tags. For more information, see Editing a suppression rule.

Adding tags to a CIS scan configuration

You can add tags to a CIS scan configuration during creation. For more information, see Creating a CIS scan configuration.

You can also edit a CIS scan configuration to include tags. For more information, see Editing a CIS scan configuration.

Adding tags with the Amazon Inspector API

You can add tags to Amazon Inspector resources with the Amazon Inspector API.

Adding tags to Amazon Inspector resources

Use the `TagResource` API to add tags to Amazon Inspector resources. You must include the ARN of the resource and the key-value pair for the tag in the command. The following example command uses an empty resource ARN for a suppression filter. The key is `CostAllocation` and value is `dev`. For information about resource types for Amazon Inspector, see Actions, resources, and condition keys for Amazon Inspector2 in the *Service Authorization Reference*.

```
aws inspector2 tag-resource \
--resource-arn "arn:${Partition}:inspector2:${Region}:${Account}:owner/${OwnerId}/
filter/${FilterId}" \
--tags CostAllocation=dev \
--region us-west-2
```

Adding tags to suppression rules during creation

Use the `CreateFilter` API to add tags to a suppression rule during creation.

```
aws inspector2 create-filter \
--name "ExampleSuppressionRuleECR" \
--action SUPPRESS \
--filter-criteria 'resourceType=[{comparison="EQUALS", value="AWS_ECR_IMAGE"}]' \
--tags Owner=ApplicationSecurity \
--region us-west-2
```

Adding tags to a CIS scan configuration

Use the `CreateCisScanConfiguration` API to add a tag to a CIS scan configuration.

```
aws inspector2 create-cis-scan-configuration \
--scan-name "CreateConfigWithTagsSample" \
--security-level LEVEL_2 \
--targets accountIds=SELF,targetResourceTags={InspectorCisScan=True} \
--schedule 'daily={startTime={timeOfDay=11:10,timezone=UTC}}' \
--tags Owner=SecurityEngineering \
--region us-west-2
```

Removing tags

You can remove tags from Amazon Inspector resources. These resources include suppression rules and CIS scan configurations. Tags help you categorize AWS resources based on specific criteria. This topic describes how to remove tags from Amazon Inspector resources.

Removing tags from Amazon Inspector resources

You can remove tags from suppression rules and CIS scan configurations. The following procedures describe how to remove tags in the console and with the Amazon Inspector API.

Removing tags in the console

You can remove tags from Amazon Inspector resources in the console.

Removing tags from suppression rules

You can remove a tag from a suppression rule by editing the suppression rule to no longer include the tag. For more information, see Editing a suppression rule.

Removing tags from a CIS scan configuration

You can remove a tag from a CIS scan configuration by editing the CIS scan configuration to no longer include the tag. For more information, see Editing a CIS scan configuration.

Removing tags with the Amazon Inspector API

You can remove a tag from an Amazon Inspector resource with the Amazon Inspector API.

Removing tags from Amazon Inspector resources

Use the `UntagResource` API to remove tags from Amazon Inspector resources.

The following snippet shows an example of how to remove tag from an Amazon Inspector resource using `UntagResource`. You must include the ARN of the resource and key for tag in the command. The following example uses an empty resource ARN for a suppression filter. The key is `CostAllocation`. For information about resource types for Amazon Inspector, see Actions, resources, and condition keys for Amazon Inspector2 in the *Service Authorization Reference*.

```
aws inspector2 untag-resource \
--resource-arn "arn:${Partition}:inspector2:${Region}:${Account}:owner/${OwnerId}/cis-configuration/${CISScanConfigurationId}" \
--tag-keys CostAllocation \
--region us-west-2
```

Monitoring Usage and Cost in Amazon Inspector

You can use the Amazon Inspector console and API to project monthly Amazon Inspector costs for your environment. If you're the Amazon Inspector administrator for a multiple-account environment, you can view the total cost for your environment and cost metrics for all member accounts. This section describes how to access usage statistics and calculate usage costs.

Using the usage console

You can assess usage and projected cost for Amazon Inspector from the console.

To access usage statistics

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. By using the AWS Region selector in the upper-right corner of the page, select the Region you want to monitor costs in.
3. In the navigation pane, choose **Usage**.

In the **By account** tab you will see the projected total cost based on the 30 day period listed under **Account usage**. In the table under the **Projected cost** column select a value to see a breakdown of usage by scan type for that account. In this detail pane you can also see which scan types have a free trial active for that account.

If you are the delegated administrator for an organization you will see a row in the table for each account within your organization. If an account in your organization is disassociated the console shows it's projected cost as a **-**.

In the **By scan type** tab you can see a break down of actual usage so far in the current 30 day period by scan type. This is the information used to calculate the projected costs in the **By account** tab.

If you are the delegated administrator for an organization you can see the usage for each account in your organization.

In this tab, you can expand any of the following panes for usage statistics:

Amazon EC2 scanning

The Amazon Inspector usage console tracks the following metrics for agent-based scanning and agentless scanning:

- **Instances (Avg)** — Amazon Inspector uses the coverage hours to calculate the average number of resources for EC2 instance scanning. The average is the total coverage hours divided by 720 hours (the number of hours in a 30 day period).
- **Coverage hours** — for Amazon EC2 scanning this is the sum total number of hours within the last 30 days that Amazon Amazon Inspector provided active coverage for each EC2 instance in an account. For EC2 instances, coverage hours are the hours from when Amazon Inspector discovered the instance until it's terminated or stopped, or excluded from scans by tags. (when you restart a stopped instance or remove an exclusion tag, Amazon Inspector resumes coverage and coverage hours for that instance will continue to accrue).

CIS instance Scans — The total number of CIS scans performed for instances in the account.

Amazon ECR scanning

Initial scans — The sum total of first time scans of images in the account within the last 30 days.

Rescans — The sum total of rescans for images in the account within the last 30 days. A rescan is any scan done on an ECR image that Amazon Inspector has previously scanned. If you have configured your ECR repository for continuous scanning, rescans occur automatically when Amazon Inspector adds a new Common Vulnerabilities and Exposures (CVE) to it's database.

Lambda scanning

The Amazon Inspector usage console tracks the following metrics for Lambda standard scanning and Lambda code scanning:

- **Number of Lambda functions (Avg)** — Amazon Inspector uses the coverage hours to calculate the average number of functions for Lambda function scanning. Average is the total coverage hours divided by 720 hours (the number of hours in a 30 day period).
- **Coverage hours** — For Lambda function scanning this is the sum total number of hours within the last 30 days that Amazon Amazon Inspector provided active coverage for each Lambda function in an account. For AWS Lambda functions the coverage hours are calculated from when Amazon Inspector discovers a function until when it's deleted or excluded from scans. If an excluded function is included again, coverage hours for that function will continue to accrue.

Azure Virtual Machine scanning

The Amazon Inspector usage console tracks the following metrics for Azure VM scanning:

- **Number of Azure VMs (Avg)** — The average number of Azure Virtual Machines monitored by Amazon Inspector VM Scanner during the 30-day period.
- **Coverage hours** — The sum total number of hours that Amazon Inspector provided active vulnerability scanning coverage for each Azure VM in an account.

Azure Function App scanning

The Amazon Inspector usage console tracks the following metrics for Azure Function App scanning:

- **Number of Function Apps (Avg)** — The average number of Azure Function Apps monitored during the 30-day period.
- **Coverage hours** — The sum total number of hours that Amazon Inspector provided active code dependency scanning coverage for each Azure Function App in an account.

Azure Container Registry (ACR) image scanning

The Amazon Inspector usage console tracks the following metrics for Azure Container Registry image scanning:

- **Initial scans** — The total number of first-time scans of Azure container images within the last 30 days.
- **Rescans** — The total number of rescans for Azure container images within the last 30 days. A rescan occurs automatically when Amazon Inspector adds a new CVE to its database that affects a previously scanned image.

> **Note**
>
> Each Azure resource type includes a 30-day free trial. The free trial starts independently for each resource type when Amazon Inspector begins scanning that resource type in your account.

Understanding how Amazon Inspector calculates usage costs

The costs provided by Amazon Inspector are estimates, not actual costs, so they may differ from those in your AWS Billing console.

Note the following about how Amazon Inspector calculates cost on the **Usage** page:

- The usage cost reflects the current region only. Prices per scan type vary by AWS Region, to review exact prices per region, see the Pricing for Amazon Inspector
- All usage projections are rounded to the nearest US dollar.
- Discounts aren't included in the projected costs.
- The projected cost represent the total cost for the 30 day usage period per scan type. If there has been less than 30 days of usage for an account, Amazon Inspector projects the cost after 30 days as if any currently covered resources will remain covered for the rest of the 30 day period.
- The cost per scan type is calculated based on the following:
  - EC2 scanning: cost reflects the average number of EC2 instances covered by Amazon Inspector in the last 30 days.
  - Machine image scanning: cost reflects the sum of the number of initial machine image scans + machine image rescans in the last 30 days. This includes on-demand machine image scans that completed successfully. Amazon Inspector doesn't meter an on-demand scan that fails. For more information, see Running on-demand machine image scans with Amazon Inspector.
  - ECR container scanning: cost reflects the sum of the number of initial image scans + image rescans in the last 30 days.
  - Lambda standard scanning: cost reflects the average number of Lambda functions covered by Amazon Inspector in the last 30 days.
  - Lambda code scanning: cost reflects the average number of Lambda functions covered by Amazon Inspector in the last 30 days.

About the Amazon Inspector free trial

In Amazon Inspector, each scan type has a free trail. When you activate a scan type, you automatically enroll in a 15-day free trial for that scan type. Once the free trial starts, it automatically expires in 15 days, even if you deactivate the scan type.

> **Note**
>
> The free trial does not apply to CIS scanning.

Note

The free trial does not apply to Running on-demand machine image scans with Amazon Inspector. Amazon Inspector meters on-demand machine image scans from your first scan, whether or not you activated Amazon Inspector for the account.

Security in Amazon Inspector

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Inspector, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Inspector. The following topics show you how to configure Amazon Inspector to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Inspector resources.

Topics

- Data protection in Amazon Inspector
- Identity and Access Management for Amazon Inspector
- Monitoring Amazon Inspector
- Compliance validation for Amazon Inspector
- Resilience in Amazon Inspector
- Infrastructure security in Amazon Inspector
- Incident response in Amazon Inspector
- Amazon Inspector Security Research
- Access Amazon Inspector using an interface endpoint (AWS PrivateLink)

Data protection in Amazon Inspector

The AWS shared responsibility model applies to data protection in Amazon Inspector. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon Inspector or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- Encryption at rest

- Encryption in transit

Encryption at rest

By default, Amazon Inspector stores data at rest using AWS encryption solutions. Amazon Inspector encrypts data, such as the following:

- Resource inventory collected with AWS Systems Manager.
- Resource inventory parsed from Amazon Elastic Container Registry images
- Generated security findings using AWS owned encryption keys from AWS Key Management Service

You cannot manage, use, or view AWS owned keys. However, you don't need to take action or change programs to protect keys that encrypt your data. For more information, see AWS owned keys.

If you disable Amazon Inspector, it permanently deletes all resources it stores or maintains for you, such as collected inventory and security findings.

Encryption at rest for code in your findings

For Amazon Inspector Lambda code scanning, Amazon Inspector partners with Amazon Q to scan your code for vulnerabilities. When a vulnerability is detected, Amazon Q extracts a snippet of your code containing the vulnerability and stores that code until Amazon Inspector requests access. By default, Amazon Q uses an AWS owned key to encrypt the extracted code. However, you can configure Amazon Inspector to use your own customer-managed AWS KMS key for encryption.

The following workflow explains how Amazon Inspector uses the key you configure to encrypt your code:

1. You supply an AWS KMS key to Amazon Inspector using the Amazon Inspector UpdateEncryptionKey API.
2. Amazon Inspector forwards the information about your AWS KMS key to Amazon Q, and Amazon Q stores the information for future use.
3. Amazon Q uses the KMS key you configured in Amazon Inspector through the key policy.
4. Amazon Q creates an encrypted data key from your AWS KMS key and stores it. This data key is used to encrypt your code data stored by Amazon Q.

5. When Amazon Inspector requests data from code scans, Amazon Q uses the KMS key to decrypt the data key. When you disable Lambda Code Scanning, Amazon Q deletes the associated data key.

Permissions for code encryption with a customer managed key

For encryption, you must create a KMS key with a policy that includes a statement allowing Amazon Inspector and Amazon Q to perform the following actions.

- `kms:Decrypt`
- `kms:DescribeKey`
- `kms:Encrypt`
- `kms:GenerateDataKey`
- `kms:GenerateDataKeyWithoutPlainText`

Policy statement

You can use the following policy statement when creating the KMS key.

> **Note**
>
> Replace *`account-id`* with your 12-digit AWS account ID. Replace *`Region`* with the AWS Region where you enabled Amazon Inspector and Lambda code scanning. Replace *`role-ARN`* with the Amazon Resource Name for your IAM role.

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "q.amazonaws.com"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:GenerateDataKeyWithoutPlaintext",
    "kms:GenerateDataKey"
  ],
```

```
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:EncryptionContext:aws:qdeveloper:lambda-codescan-scope": "account-id"
    },
    "StringEquals": {
      "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:qdeveloper:Region:account-id:scans/*"
    }
  }
},
{
  "Effect": "Allow",
  "Principal": {
    "Service": "q.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:qdeveloper:Region:account-id:scans/*"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:GenerateDataKeyWithoutPlaintext",
    "kms:GenerateDataKey"
  ],
  "Principal": {
    "AWS": "role-ARN"
  },
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "inspector2.Region.amazonaws.com"
```

```
    },
    "StringLike": {
      "kms:EncryptionContext:aws:qdeveloper:lambda-codescan-scope": "account-id"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "kms:DescribeKey"
  ],
  "Principal": {
    "AWS": "role-ARN"
  },
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "inspector2.Region.amazonaws.com"
    }
  }
}
```

The policy statement is formatted in JSON. After you include the statement, review the policy to make sure the syntax is valid. If the statement is the last statement in the policy, place a comma after the closing brace for the previous statement. If the statement is the first statement or between two existing statements in the policy, place a comma after the closing brace for the statement.

Note

Amazon Inspector no longer supports grants to encrypt code snippets extracted from packages. If you are using a grant-based policy, you can still access your findings. However, if you ever update or reset your KMS key or disable Lambda Code Scanning, you will need to use the KMS key policy described in this section.

If you set, update, or reset the encryption key for your account, you must use an Amazon Inspector administrator policy, such as the AWS managed policy `AmazonInspector2FullAccess`.

Configuring encryption with a customer managed key

To configure encryption for your account using a customer managed key you must be an Amazon Inspector administrator with the permissions outlined in Permissions for code encryption with a customer managed key. Additionally you will need a AWS KMS key in the same AWS Region as your findings, or a multi-region key. You can use an existing symmetric key in your account or create a symmetric customer managed key by using the AWS Management Console, or the AWS KMS APIs. For more information see Creating symmetric encryption AWS KMS keys in the AWS KMS user guide.

> **Note**
>
> Effective June 13th, 2025, the service principal in AWS KMS requests logged in CloudTrail during code snippet encryption/decryption is changing from "codeguru-reviewer" to "q".

Using the Amazon Inspector API to configure encryption

To set a key for encryption the UpdateEncryptionKey operation of the Amazon Inspector API while signed in as an Amazon Inspector administrator. In the API request, use the `kmsKeyId` field to specify the ARN of the AWS KMS key you want to use. For `scanType` enter `CODE` and for `resourceType` enter `AWS_LAMBDA_FUNCTION`.

You can use UpdateEncryptionKey API to check view which AWS KMS key Amazon Inspector is using for encryption.

> **Note**
>
> If you attempt to use `GetEncryptionKey` when you haven't set a customer managed key the operation returns a `ResourceNotFoundException` error which means that an AWS owned key is being used for encryption.

If you delete the key or change it's policy to deny access to Amazon Inspector or Amazon Q you will be unable to access your code vulnerability findings and Lambda code scanning will fail for your account.

You can use `ResetEncryptionKey` to resume using an AWS owned key to encrypt code extracted as part of your Amazon Inspector findings.

Encryption in transit

AWS encrypts all data in transit between AWS internal systems and other AWS services. AWS Systems Manager gathers telemetry data from customer-owned EC2 instances it sends to AWS over a Transport Layer Security (TLS)-protected channel for assessment. Amazon ECR and AWS Lambda function scan findings that are sent to Security Hub CSPM are encrypted using a TLS-protected channel. For more information, see Data Protection in Systems Manager to understand how SSM encrypts data in transit.

Identity and Access Management for Amazon Inspector

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Inspector resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon Inspector works with IAM
- Identity-based policy examples for Amazon Inspector
- AWS managed policies for Amazon Inspector
- Using service-linked roles for Amazon Inspector
- Troubleshooting Amazon Inspector identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon Inspector identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon Inspector works with IAM)

- **IAM administrator** - write policies to manage access (see Identity-based policy examples for Amazon Inspector)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more

information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between

managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Inspector works with IAM

Before you use IAM to manage access to Amazon Inspector, learn what IAM features are available to use with Amazon Inspector.

IAM features you can use with Amazon Inspector

IAM feature	Amazon Inspector support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how Amazon Inspector and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon Inspector

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can

perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon Inspector

To view examples of Amazon Inspector identity-based policies, see Identity-based policy examples for Amazon Inspector.

Resource-based policies within Amazon Inspector

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon Inspector

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Inspector actions, see Actions defined by Amazon Inspector in the *Service Authorization Reference*.

Policy actions in Amazon Inspector use the following prefix before the action:

```
inspector2
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "inspector2:action1",
      "inspector2:action2"
         ]
```

To view examples of Amazon Inspector identity-based policies, see Identity-based policy examples for Amazon Inspector.

Policy resources for Amazon Inspector

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon Inspector resource types and their ARNs, see Resources defined by Amazon Inspector in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Inspector.

To view examples of Amazon Inspector identity-based policies, see Identity-based policy examples for Amazon Inspector.

Policy condition keys for Amazon Inspector

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon Inspector condition keys, see Condition keys for Amazon Inspector in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Inspector.

To view examples of Amazon Inspector identity-based policies, see Identity-based policy examples for Amazon Inspector.

ACLs in Amazon Inspector

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon Inspector

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amazon Inspector

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Amazon Inspector

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon Inspector

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon Inspector functionality. Edit service roles only when Amazon Inspector provides guidance to do so.

Service-linked roles for Amazon Inspector

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS

account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon Inspector

By default, users and roles don't have permission to create or modify Amazon Inspector resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon Inspector, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon Inspector in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Amazon Inspector console
- Allow users to view their own permissions
- Allow read-only access to all Amazon Inspector resources
- Allow full access to all Amazon Inspector resources

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon Inspector resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies

that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon Inspector console

To access the Amazon Inspector console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Inspector resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Amazon Inspector console, also attach the Amazon Inspector *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow read-only access to all Amazon Inspector resources

This example shows a policy that allows read-only access to all Amazon Inspector resources.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "inspector2:Describe*",
                "inspector2:Get*",
                "inspector2:BatchGet*",
                "inspector2:List*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "organizations:ListDelegatedAdministrators",
                "organizations:ListAWSServiceAccessForOrganization",
                "organizations:DescribeOrganizationalUnit",
                "organizations:DescribeAccount",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow full access to all Amazon Inspector resources

This example shows a policy that allows full access to all Amazon Inspector resources.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "inspector2:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "inspector2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "organizations:EnableAWSServiceAccess",
                "organizations:RegisterDelegatedAdministrator",
                "organizations:ListDelegatedAdministrators",
                "organizations:ListAWSServiceAccessForOrganization",
                "organizations:DescribeOrganizationalUnit",
                "organizations:DescribeAccount",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for Amazon Inspector

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AmazonInspector2FullAccess_v2

You can attach the `AmazonInspector2FullAccess_v2` policy to your IAM identities.

This policy grants full access to Amazon Inspector and access to other related services.

Permissions details

This policy includes the following permissions.

- `inspector2` – Allows complete access to Amazon Inspector APIs.
- `codeguru-security` – Allows administrators to retrieve security findings and configuration settings for an account.
- `iam` – Allows Amazon Inspector to create the service-linked roles `AWSServiceRoleForAmazonInspector2`, `AWSServiceRoleForAmazonInspector2Agentless`, and `AWSServiceRoleForAmazonInspector2ThirdParty`. `AWSServiceRoleForAmazonInspector2` is required for Amazon Inspector to perform operations like retrieving information about Amazon EC2 instances, Amazon ECR repositories, and Amazon ECR container images. It's also required to decrypt Amazon EBS snapshots encrypted with AWS KMS keys. Additionally, allows creation of the `AWSServiceRoleForConfigThirdParty` service-linked role, which is required by to record

multi-cloud resources. For more information, see Using service-linked roles for Amazon Inspector.

- `config` – Allows Amazon Inspector to create, read, and list connectors in for multi-cloud resource discovery.
- `organizations` – `AllowServicePrincipalBasedAccessToOrganizationApis` allows only service principals to create service-linked roles for AWS accounts, register an AWS account as a delegated administrator for an organization, and list delegated administrators in an organization. `AllowOrganizationalBasedAccessToOrganizationApis` allows the policy holder to retrieve information, specifically resource-level ARNs, about an organizational unit. `AllowAccountsBasedAccessToOrganizationApis` allows the policy holder to retrieve information, specifically resource-level ARNs, about an AWS account. `AllowAccessToOrganizationApis` allows the policy holder to view AWS services integrated with an organization and organization information. The policy allows listing Inspector organizational policies with filtering by Inspector policy types, viewing delegation resource policies established by management accounts, and viewing the effective Inspector policies applied to accounts.

> **Note**
>
> Amazon Inspector no longer uses CodeGuru to perform Lambda scans. AWS will discontinue support for CodeGuru on November 20, 2025. For more information, see End of support for CodeGuru Security. Amazon Inspector now uses Amazon Q to perform Lambda scans and does not require the permissions described in this section.

To review the permissions for this policy, see AmazonInspector2FullAccess_v2 in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AWSInspector2OrganizationsAccess

You can attach the `AWSInspector2OrganizationsAccess` policy to your IAM identities.

This policy grants administrative permissions to enable and manage Amazon Inspector for an organization in AWS Organizations. The permissions for this policy allow the organization management account to designate the delegated administrator account for Amazon Inspector. They also allow the delegated administrator account to enable organization accounts as member accounts.

This policy only provides permissions for AWS Organizations. The organization management account and delegated administrator account also require permissions for associated actions. These permissions can be granted using the `AmazonInspector2FullAccess_v2` managed policy.

Permissions details

This policy includes the following permissions.

- `organizations:ListAccounts` – Allows principals to retrieve the list of accounts that are part of an organization.
- `organizations:DescribeOrganization` – Allows principals to retrieve information about the organization.
- `organizations:ListRoots` – Allows principals to list the root of an organization.
- `organizations:ListDelegatedAdministrators` – Allows principals to list the delegated administrator of an organization.
- `organizations:ListAWSServiceAccessForOrganization` – Allows principals to list the AWS services that an organization uses.
- `organizations:ListOrganizationalUnitsForParent` – Allows principals to list the child organizational units (OU) of a parent OU.
- `organizations:ListAccountsForParent` – Allows principals to list the child accounts of a parent OU.
- `organizations:ListParents` – Lists the root or organizational units (OUs) that serve as the immediate parent of the specified child OU or account.
- `organizations:DescribeAccount` – Allows principals to retrieve information about an account in the organization.
- `organizations:DescribeOrganizationalUnit` – Allows principals to retrieve information about an OU in the organization.
- `organizations:ListPolicies` – Retrieves the list of all policies in an organization of a specified type.
- `organizations:ListPoliciesForTarget` – Lists the policies that are directly attached to the specified target root, organizational unit (OU), or account.
- `organizations:ListTargetsForPolicy` – Lists all the roots, organizational units (OUs), and accounts that the specified policy is attached to.
- `organizations:DescribeResourcePolicy` – Retrieves information about a resource policy.

- `organizations:EnableAWSServiceAccess` – Allows principals to enable the integration with Organizations.
- `organizations:RegisterDelegatedAdministrator` – Allows principals to designate the delegated administrator account.
- `organizations:DeregisterDelegatedAdministrator` – Allows principals to remove the delegated administrator account.
- `organizations:DescribePolicy` – Retrieves information about a policy.
- `organizations:DescribeEffectivePolicy` – Returns the contents of the effective policy for specified policy type and account.
- `organizations:CreatePolicy` – Creates a policy of a specified type that you can attach to a root, an organizational unit (OU), or an individual AWS account.
- `organizations:UpdatePolicy` – Updates an existing policy with a new name, description, or content.
- `organizations:DeletePolicy` – Deletes the specified policy from your organization.
- `organizations:AttachPolicy` – Attaches a policy to a root, an organizational unit (OU), or an individual account.
- `organizations:DetachPolicy` – Detaches a policy from a target root, organizational unit (OU), or account.
- `organizations:EnablePolicyType` – Enables a policy type in a root.
- `organizations:DisablePolicyType` – Disables an organizational policy type in a root.
- `organizations:TagResource` – Adds one or more tags to a specified resource.
- `organizations:UntagResource` – Removes any tags with the specified keys from a specified resource.
- `organizations:ListTagsForResource` – Lists tags that are attached to a specified resource.

To review the permissions for this policy, see AWSInspector2OrganizationsAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonInspector2FullAccess

You can attach the `AmazonInspector2FullAccess` policy to your IAM identities.

This policy grants administrative permissions that allow full access to Amazon Inspector.

> **Important**
>
> For enhanced security and restrictive permissions to Inspector 2 service principals, we recommend that you use AmazonInspector2FullAccess_v2.

Permissions details

This policy includes the following permissions.

- `inspector2` – Allows full access to Amazon Inspector functionality.
- `iam` – Allows Amazon Inspector to create the service-linked roles `AWSServiceRoleForAmazonInspector2`, `AWSServiceRoleForAmazonInspector2Agentless`, and `AWSServiceRoleForAmazonInspector2ThirdParty`. `AWSServiceRoleForAmazonInspector2` is required for Amazon Inspector to perform operations such as retrieve information about your Amazon EC2 instances, Amazon ECR repositories, and container images. It's also required for Amazon Inspector to analyze your VPC network and describe accounts that are associated with your organization. `AWSServiceRoleForAmazonInspector2Agentless` is required for Amazon Inspector to perform operations, such as retrieve information about your Amazon EC2 instances and Amazon EBS snapshots. It's also required to decrypt Amazon EBS snapshots that are encrypted with AWS KMS keys. `AWSServiceRoleForAmazonInspector2ThirdParty` is required for Amazon Inspector to perform multi-cloud scanning operations, such as managing cloud connectors and configuration recorders. Additionally, allows creation of the `AWSServiceRoleForConfigThirdParty` service-linked role, which is required by to record multi-cloud resources. For more information, see Using service-linked roles for Amazon Inspector.
- `config` – Allows Amazon Inspector to create, read, and list connectors in for multi-cloud resource discovery.
- `organizations` – Allows administrators to use Amazon Inspector for an organization in AWS Organizations. When you activate trusted access for Amazon Inspector in AWS Organizations, members of the delegated administrator account can manage settings and view findings across their organization.

- `codeguru-security` – Allows administrators to use Amazon Inspector to retrieve information code snippets and change encryption settings for code that CodeGuru Security stores. For more information, see Encryption at rest for code in your findings.

To review the permissions for this policy, see AmazonInspector2FullAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonInspector2ReadOnlyAccess

You can attach the `AmazonInspector2ReadOnlyAccess` policy to your IAM identities.

This policy grants permissions that allow read-only access to Amazon Inspector.

Permissions details

This policy includes the following permissions.

- `inspector2` – Allows read-only access to Amazon Inspector functionality.
- `organizations` – Allows details about Amazon Inspector coverage for an organization in AWS Organizations to be viewed. Additionally allows viewing of Inspector organizational policies through `ListPolicies` with filtering by Inspector policy types, viewing delegation resource policies through `DescribeResourcePolicy`, and viewing effective Inspector policies applied to accounts through `DescribeEffectivePolicy`. This allows users to understand centralized inspector enablement established through organizational policies without the ability to modify them.
- `codeguru-security` – Allows code snippets to be retrieved from CodeGuru Security. Also allows encryption settings for your code stored in CodeGuru Security to be viewed.
- `config` – Allows the details and health status of all Amazon Inspector multi-cloud connectors to be viewed. This includes listing and describing any configuration recorders that are associated with the connectors.
- `ssm` – Allows the details and health status of all Amazon Inspector multi-cloud connectors to be viewed.

To review the permissions for this policy, see AmazonInspector2ReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonInspector2ManagedCisPolicy

You can attach the `AmazonInspector2ManagedCisPolicy` policy to your IAM entities. This policy should be attached to a role that grants permissions to your Amazon EC2 instances to run CIS scans of the instance. You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Permissions details

This policy includes the following permissions.

- `inspector2` – Allows access to actions used to run CIS scans.

To review the permissions for this policy, see AmazonInspector2ManagedCisPolicy in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonInspector2ServiceRolePolicy

You can't attach the `AmazonInspector2ServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon Inspector to perform actions on your behalf. For more information, see Using service-linked roles for Amazon Inspector.

AWS managed policy: AmazonInspector2AgentlessServiceRolePolicy

You can't attach the `AmazonInspector2AgentlessServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon Inspector to perform actions on your behalf. For more information, see Using service-linked roles for Amazon Inspector.

AWS managed policy: AmazonInspector2AmiServiceRolePolicy

You can't attach the `AmazonInspector2AmiServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon Inspector to perform actions on

your behalf for machine image scanning. For more information, see Service-linked role permissions for Amazon Inspector machine image scans.

Permissions details

This policy includes the following permissions.

- `ec2` – Allows Amazon Inspector to retrieve information about the machine images that your account owns and the snapshots that back them.
- `ebs` – Allows Amazon Inspector to read the contents of the snapshots that back your machine images. Unlike agentless Amazon EC2 scanning, machine image scanning doesn't create new snapshots.
- `kms` – Allows Amazon Inspector to use select decryption actions to decrypt snapshots that are encrypted with AWS KMS customer managed keys.

To review the permissions for this policy, see AmazonInspector2AmiServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonInspector2ThirdPartyServiceRolePolicy

You can't attach the `AmazonInspector2ThirdPartyServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon Inspector to perform actions on your behalf for multi-cloud resource scanning. For more information, see Using service-linked roles for Amazon Inspector.

AWS managed policy: AmazonInspector2ManagedTelemetryPolicy

You can attach the `AmazonInspector2ManagedTelemetryPolicy` policy to your IAM entities. This policy grants permissions for Amazon Inspector telemetry operations, allowing the service to collect and transmit package inventory data for vulnerability scanning.

Permissions details

This policy includes the following permissions.

- `inspector2-telemetry` – Allows access to actions for package invetory data transmission.

To view more details about the policy, including the latest version of the JSON policy document, see AmazonInspector2ManagedTelemetryPolicy in the *AWS Managed Policy Reference Guide*.

Amazon Inspector updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Inspector since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Inspector Document history page.

Change	Description	Date
AmazonInspector2AmiServiceRolePolicy – New policy	Amazon Inspector has added a new service-linked role policy to allow scanning of the machine images (AMIs) that your account owns. The policy allows Amazon Inspector to retrieve information about your machine images and the Amazon EBS snapshots that back them, read the contents of those snapshots, and decrypt snapshots that are encrypted with AWS KMS customer managed keys.	September 8, 2026
AmazonInspector2FullAccess_v2 – Updates to an existing policy	Amazon Inspector added `iam` permissions that allow Amazon Inspector to create the service-linked role for multi-cloud connectors (`AWSServiceRoleForAmazonInspector2ThirdParty`) and the third-party service-linked role (`AWSServiceRoleForConfigThirdParty`).	July 7, 2026

Change	Description	Date
	Amazon Inspector also added the multi-cloud service principal (`multicloud.inspector2.amazonaws.com`) to the conditions for `organizations` actions, and added connector permissions for multi-cloud resource discovery.	
AmazonInspector2ReadOnlyAccess – Updates to an existing policy	Amazon Inspector added read-only and AWS Systems Manager permissions for viewing multi-cloud connector health status.	July 7, 2026
AWSInspector2OrganizationsAccess – New policy	Amazon Inspector has added a new managed policy that grants permissions needed to enable and manage Amazon Inspector via AWS Organizations policy.	March 3, 2026
AmazonInspector2ManagedTelemetryPolicy – New policy	Amazon Inspector has added a new managed policy that grants permissions for Amazon Inspector telemetry operations, allowing the service to collect and transmit package inventory data for vulnerability scanning.	February 5, 2026

Change	Description	Date
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added a new permission that allows Amazon Inspector to describe firewall metadata for network reachability analysis. Additionally, Amazon Inspector has added additional resource scoping to allow Amazon Inspector to create, update, and start SSM associations with SSM document `AWS-ConfigureAWSPackage` .	February 3, 2026
AmazonInspector2FullAccess_v2 and AmazonInspector2ReadOnlyAccess – Updates to existing policies	Amazon Inspector has added new permissions allowing the policy holders to view Inspector organizational policies and delegation configurations. This supports centralized management and visibility of Inspector enablement through AWS Organizations policies.	November 14, 2025
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions allowing the Amazon Inspector AWS Organizations policy to enforce the enablement and disablement of Amazon Inspector.	November 10, 2025

Change	Description	Date
AmazonInspector2FullAccess_v2 – New policy	Amazon Inspector has added a new managed policy that provides full access to Amazon Inspector and access to other related services.	July 03, 2025
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added a new permission that allows Amazon Inspector to describe IP addresses and internet gateways.	April 29, 2025
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow read-only access to Amazon ECS and Amazon EKS actions.	March 25, 2025
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to return function tags in AWS Lambda.	July 31, 2024
AmazonInspector2FullAccess – Updates to an existing policy	Amazon Inspector has added permissions that allow Amazon Inspector to create the service-linked role `AWSServiceRoleForAmazonInspector2Agentless` This allows users to perform agent-based scanning and agentless scanning when they enable Amazon Inspector.	April 24, 2024

Change	Description	Date
AmazonInspector2ManagedCisPolicy – New policy	Amazon Inspector has added a new managed policy that you can use as part of an instance profile to allow CIS scans on an instance.	January 23, 2024
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to start CIS scans on target instances.	January 23, 2024
AmazonInspector2AgentlessServiceRolePolicy – New policy	Amazon Inspector has added a new service-linked role policy to allow agentless scanning of EC2 instance.	November 27, 2023
AmazonInspector2ReadOnlyAccess – Updates to an existing policy	Amazon Inspector has added new permissions that allow read-only users to retrieve vulnerability intelligence details for package vulnerability findings.	September 22, 2023
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to scan network configurations of Amazon EC2 instances that are part of Elastic Load Balancing target groups.	August 31, 2023
AmazonInspector2ReadOnlyAccess – Updates to an existing policy	Amazon Inspector has added new permissions that allow read-only users to export Software Bill of Materials (SBOM) for their resources.	June 29, 2023

Change	Description	Date
AmazonInspector2ReadOnlyAccess – Updates to an existing policy	Amazon Inspector has added new permissions that allow read-only users to retrieve details of encryption settings for Lambda code scanning findings for their account.	June 13, 2023
AmazonInspector2FullAccess – Updates to an existing policy	Amazon Inspector has added new permissions that allow users configure a customer managed KMS key to encrypt code in findings from Lambda code scanning.	June 13, 2023
AmazonInspector2ReadOnlyAccess – Updates to an existing policy	Amazon Inspector has added new permissions that allow read-only users to retrieve details of Lambda code scanning status and findings for their account.	May 02, 2023
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to create AWS CloudTrail service-linked channels in your account when you activate Lambda scanning. This allows Amazon Inspector to monitor CloudTrail events in your account.	April 30, 2023

Change	Description	Date
AmazonInspector2FullAccess – Updates to an existing policy	Amazon Inspector has added new permissions that allow users to retrieve details of code vulnerability findings from Lambda code scanning.	April 21, 2023
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to send information to Amazon EC2 Systems Manager about the custom paths a customer has defined for Amazon EC2 deep inspection.	April 17, 2023
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to create AWS CloudTrail service-linked channels in your account when you activate Lambda scanning. This allows Amazon Inspector to monitor CloudTrail events in your account.	April 30, 2023

Change	Description	Date
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added new permissions that allow Amazon Inspector to request scans of the developer code in AWS Lambda functions , and receive scan data from Amazon CodeGuru Security. Additionally, Amazon Inspector has added permissions to review IAM policies. Amazon Inspector uses this information to scan Lambda functions for code vulnerabilities.	February 28, 2023
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added a new statement that allows Amazon Inspector to retrieve information from CloudWatch about when an AWS Lambda function was last invoked. Amazon Inspector uses this information to focus scans on the Lambda functions in your environment that have been active in the last 90 days.	February 20, 2023

Change	Description	Date
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added a new statement that allows Amazon Inspector to retrieve information about AWS Lambda functions, including each layer version that is associated with each function. Amazon Inspector uses this information to scan Lambda functions for security vulnerabilities.	November 28, 2022
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has added a new action to allow Amazon Inspector to describe SSM association executions. Additionally, Amazon Inspector has added additional resource scoping to allow Amazon Inspector to create, update, delete, and start SSM associations with `AmazonInspector2` owned SSM documents.	August 31, 2022
AmazonInspector2ServiceRolePolicy Updates to an existing policy	Amazon Inspector has updated the resource scoping of the policy to allow Amazon Inspector to collect software inventory in other AWS partitions.	August 12, 2022

Change	Description	Date
AmazonInspector2ServiceRolePolicy – Updates to an existing policy	Amazon Inspector has restructured the resource scoping of the actions allowing Amazon Inspector to create, delete, and update SSM associations.	August 10, 2022
AmazonInspector2ReadOnlyAccess – New policy	Amazon Inspector added a new policy to allow read-only access to Amazon Inspector functionality.	January 21, 2022
AmazonInspector2FullAccess – New policy	Amazon Inspector added a new policy to allow full access to Amazon Inspector functionality.	November 29, 2021
AmazonInspector2ServiceRolePolicy – New policy	Amazon Inspector added a new policy to allow Amazon Inspector to perform actions in other services on your behalf.	November 29, 2021
Amazon Inspector started tracking changes	Amazon Inspector started tracking changes for its AWS managed policies.	November 29, 2021

Using service-linked roles for Amazon Inspector

Amazon Inspector uses an AWS Identity and Access Management (IAM) service-linked role named `AWSServiceRoleForAmazonInspector2`. This service-linked role is an IAM role that is linked directly to Amazon Inspector. It is predefined by Amazon Inspector and it includes all the permissions that Amazon Inspector requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon Inspector easier because you don't have to manually add the necessary permissions. Amazon Inspector defines the permissions of its service-

linked role and, unless defined otherwise, only Amazon Inspector can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You must configure permissions to allow an IAM entity (such as a group or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*. You can delete a service-linked role only after deleting its related resources. This protects your Amazon Inspector resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to review the service-linked role documentation for that service.

Service-linked role permissions for Amazon Inspector

Amazon Inspector uses the managed policy named `AWSServiceRoleForAmazonInspector2`. This service-linked role trusts the `inspector2.amazonaws.com` service to assume the role.

The permissions policy for the role, which is named `AmazonInspector2ServiceRolePolicy`, allows Amazon Inspector to perform tasks such as:

- Use Amazon Elastic Compute Cloud (Amazon EC2) actions to retrieve information about your instances and network paths.
- Use AWS Systems Manager actions to retrieve inventory from your Amazon EC2 instances, and to retrieve information about third-party packages from custom paths.
- Use the AWS Systems Manager `SendCommand` action to invoke CIS scans for target instances.
- Use Amazon Elastic Container Registry actions to retrieve information about your container images.
- Use AWS Lambda actions to retrieve information about your Lambda functions.
- Use AWS Organizations actions to describe associated accounts.
- Use CloudWatch actions to retrieve information about the last time your Lambda functions were invoked.
- Use select IAM actions to retrieve information about your IAM policies that could create security vulnerabilities in your Lambda code.
- Use Amazon Q actions to perform scans of the code in your Lambda functions. Amazon Inspector uses the following Amazon Q actions:

- codeguru-security:CreateScan – Grants permission to create Amazon Q; scan.
- codeguru-security:GetScan – Grants permission to retrieve Amazon Q scan metadata.
- codeguru-security:ListFindings – Grants permission to retrieve findings generated by Amazon Q.
- codeguru-security:DeleteScansByCategory – Grants permission for Amazon Q to delete scans initiated by Amazon Inspector.
- codeguru-security:BatchGetFindings – Grants permission to retrieve a batch of specific findings generated by Amazon Q.

- Use select Elastic Load Balancing actions to preform network scans of EC2 instances that are part of Elastic Load Balancing target groups.
- Use Amazon ECS and Amazon EKS actions to allow read-only access to view clusters and tasks and describe tasks.
- Use AWS Organizations actions to list delegated administrators for Amazon Inspector across organizations.
- Use Amazon Inspector actions to enable and disable Amazon Inspector across organizations.
- Use Amazon Inspector actions to designate delegated administrator accounts and associate member accounts across organizations.

> **Note**
>
> Amazon Inspector no longer uses CodeGuru to perform Lambda scans. AWS will discontinue support for CodeGuru on November 20, 2025. For more information, see End of support for CodeGuru Security. Amazon Inspector now uses Amazon Q to perform Lambda scans and does not require the permissions described in this section.

To review the permissions for this policy, see AmazonInspector2ServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

Creating a service-linked role for Amazon Inspector

You don't need to manually create a service-linked role. When you activate Amazon Inspector in the AWS Management Console, the AWS CLI, or the AWS API, Amazon Inspector creates the service-linked role for you.

Editing a service-linked role for Amazon Inspector

Amazon Inspector does not allow you to edit the `AWSServiceRoleForAmazonInspector2` service-linked role. After a service-linked role is created, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role by using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon Inspector

If you no longer need to use Amazon Inspector, we recommend that you delete the `AWSServiceRoleForAmazonInspector2` service-linked role. Before you can delete the role, you must deactivate Amazon Inspector in each AWS Region where it's activated. When you deactivate Amazon Inspector, it doesn't delete the role for you. Therefore, if you activate Amazon Inspector again, it can use the existing role. That way you can avoid having an unused entity that's not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

If you delete this service-linked role and then need to create it again, you can use the same process to re-create the role in your account. When you activate Amazon Inspector, Amazon Inspector re-creates the service-linked role for you.

> **Note**
>
> If the Amazon Inspector service is using the role when you try to delete the resources, the deletion might fail. If that happens, wait a few minutes and then try the operation again.

You can use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonInspector2` service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Service-linked role permissions for Amazon Inspector agentless scans

Amazon Inspector agentless scanning uses the service-linked role named `AWSServiceRoleForAmazonInspector2Agentless`. This SLR allows Amazon Inspector to create an Amazon EBS volume snapshot in your account, and then access the data from that snapshot. This service-linked role trusts the `agentless.inspector2.amazonaws.com` service to assume the role.

Important

The statements in this service-linked role prevent Amazon Inspector from performing agentless scans on any EC2 instance that you have excluded from scans using the `InspectorEc2Exclusion` tag. Additionally the statements prevent Amazon Inspector from accessing encrypted data from a volume when the KMS key used to encrypt it has the `InspectorEc2Exclusion` tag. For more information, see Excluding instances from Amazon Inspector scans.

The permissions policy for the role, which is named `AmazonInspector2AgentlessServiceRolePolicy`, allows Amazon Inspector to perform tasks such as:

- Use Amazon Elastic Compute Cloud (Amazon EC2) actions to retrieve information about your EC2 instances, volumes, and snapshots.
  - Use Amazon EC2 tagging actions to tag snapshots for scans with the `InspectorScan` tag key.
  - Use Amazon EC2 snapshot actions to create snapshots, tag them with the `InspectorScan` tag key, and then delete snapshots of Amazon EBS volumes that have been tagged with the `InspectorScan` tag key.
- Use Amazon EBS actions to retrieve information from snapshots tagged with the `InspectorScan` tag key.
- Use select AWS KMS decryption actions to decrypt snapshots encrypted with AWS KMS customer managed keys. Amazon Inspector does not decrypt snapshots when the KMS key used to encrypt them is tagged with the `InspectorEc2Exclusion` tag.

The role is configured with the following permissions policy.

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "InstanceIdentification",
   "Effect": "Allow",
```

```
    "Action": [
     "ec2:DescribeInstances",
     "ec2:DescribeVolumes",
     "ec2:DescribeSnapshots"
    ],
    "Resource": "*"
   },
   {
    "Sid": "GetSnapshotData",
    "Effect": "Allow",
    "Action": [
     "ebs:ListSnapshotBlocks",
     "ebs:GetSnapshotBlock"
    ],
    "Resource": "arn:aws:ec2:*:*:snapshot/*",
    "Condition": {
     "StringLike": {
      "aws:ResourceTag/InspectorScan": "*"
     }
    }
   },
   {
    "Sid": "CreateSnapshotsAnyInstanceOrVolume",
    "Effect": "Allow",
    "Action": "ec2:CreateSnapshots",
    "Resource": [
     "arn:aws:ec2:*:*:instance/*",
     "arn:aws:ec2:*:*:volume/*"
    ]
   },
   {
    "Sid": "DenyCreateSnapshotsOnExcludedInstances",
    "Effect": "Deny",
    "Action": "ec2:CreateSnapshots",
    "Resource": "arn:aws:ec2:*:*:instance/*",
    "Condition": {
     "StringEquals": {
      "ec2:ResourceTag/InspectorEc2Exclusion": "true"
     }
    }
   },
   {
    "Sid": "CreateSnapshotsOnAnySnapshotOnlyWithTag",
    "Effect": "Allow",
```

```
    "Action": "ec2:CreateSnapshots",
    "Resource": "arn:aws:ec2:*:*:snapshot/*",
    "Condition": {
     "Null": {
      "aws:TagKeys": "false"
     },
     "ForAllValues:StringEquals": {
      "aws:TagKeys": "InspectorScan"
     }
    }
   },
   {
    "Sid": "CreateOnlyInspectorScanTagOnlyUsingCreateSnapshots",
    "Effect": "Allow",
    "Action": "ec2:CreateTags",
    "Resource": "arn:aws:ec2:*:*:snapshot/*",
    "Condition": {
     "StringLike": {
      "ec2:CreateAction": "CreateSnapshots"
     },
     "Null": {
      "aws:TagKeys": "false"
     },
     "ForAllValues:StringEquals": {
      "aws:TagKeys": "InspectorScan"
     }
    }
   },
   {
    "Sid": "DeleteOnlySnapshotsTaggedForScanning",
    "Effect": "Allow",
    "Action": "ec2:DeleteSnapshot",
    "Resource": "arn:aws:ec2:*:*:snapshot/*",
    "Condition": {
     "StringLike": {
      "ec2:ResourceTag/InspectorScan": "*"
     }
    }
   },
   {
    "Sid": "DenyKmsDecryptForExcludedKeys",
    "Effect": "Deny",
    "Action": "kms:Decrypt",
    "Resource": "arn:aws:kms:*:*:key/*",
```

```
  "Condition": {
   "StringEquals": {
    "aws:ResourceTag/InspectorEc2Exclusion": "true"
   }
  }
 },
 {
  "Sid": "DecryptSnapshotBlocksVolContext",
  "Effect": "Allow",
  "Action": "kms:Decrypt",
  "Resource": "arn:aws:kms:*:*:key/*",
  "Condition": {
   "StringEquals": {
    "aws:ResourceAccount": "${aws:PrincipalAccount}"
   },
   "StringLike": {
    "kms:ViaService": "ec2.*.amazonaws.com",
    "kms:EncryptionContext:aws:ebs:id": "vol-*"
   }
  }
 },
 {
  "Sid": "DecryptSnapshotBlocksSnapContext",
  "Effect": "Allow",
  "Action": "kms:Decrypt",
  "Resource": "arn:aws:kms:*:*:key/*",
  "Condition": {
   "StringEquals": {
    "aws:ResourceAccount": "${aws:PrincipalAccount}"
   },
   "StringLike": {
    "kms:ViaService": "ec2.*.amazonaws.com",
    "kms:EncryptionContext:aws:ebs:id": "snap-*"
   }
  }
 },
 {
  "Sid": "DescribeKeysForEbsOperations",
  "Effect": "Allow",
  "Action": "kms:DescribeKey",
  "Resource": "arn:aws:kms:*:*:key/*",
  "Condition": {
   "StringEquals": {
    "aws:ResourceAccount": "${aws:PrincipalAccount}"
```

```
    },
    "StringLike": {
     "kms:ViaService": "ec2.*.amazonaws.com"
    }
   }
  },
  {
   "Sid": "ListKeyResourceTags",
   "Effect": "Allow",
   "Action": "kms:ListResourceTags",
   "Resource": "arn:aws:kms:*:*:key/*"
  }
 ]
}
```

Creating a service-linked role for agentless scanning

You don't need to manually create a service-linked role. When you activate Amazon Inspector in the AWS Management Console, the AWS CLI, or the AWS API, Amazon Inspector creates the service-linked role for you.

Editing a service-linked role for agentless scanning

Amazon Inspector does not allow you to edit the `AWSServiceRoleForAmazonInspector2Agentless` service-linked role. After a service-linked role is created, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role by using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for agentless scanning

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that isn't actively monitored or maintained.

> **Important**
>
> In order to delete the `AWSServiceRoleForAmazonInspector2Agentless` role, you must set your scan mode to agent-based in all Regions where agentless scanning is available.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonInspector2Agentless service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Service-linked role permissions for Amazon Inspector machine image scans

Amazon Inspector machine image scanning uses the service-linked role named `AWSServiceRoleForAmazonInspector2Ami`. This SLR allows Amazon Inspector to read the data from the existing Amazon EBS snapshots that back the machine images that your account owns. Unlike agentless Amazon EC2 scanning, machine image scanning doesn't create new snapshots. This service-linked role trusts the `ami.inspector2.amazonaws.com` service to assume the role.

The permissions policy for the role, which is named `AmazonInspector2AmiServiceRolePolicy`, allows Amazon Inspector to perform tasks such as:

- Use Amazon Elastic Compute Cloud (Amazon EC2) actions to retrieve information about the machine images that your account owns and the snapshots that back them.
- Use Amazon EBS actions to read the contents of the snapshots that back your machine images.
- Use select AWS KMS decryption actions to decrypt snapshots encrypted with AWS KMS customer managed keys.

Amazon Inspector doesn't use this role to scan machine images that you have excluded from scans. For more information, see Excluding copied and backup machine images.

For the permissions in this policy, see AmazonInspector2AmiServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

Creating a service-linked role for machine image scanning

You don't need to manually create the `AWSServiceRoleForAmazonInspector2Ami` service-linked role. When you enable machine image scanning in Amazon Inspector, Amazon Inspector automatically creates the service-linked role for you.

Amazon Inspector also creates this role automatically the first time you call the StartOnDemandScan API operation for a machine image, even if you haven't enabled machine image scanning for the account. To call StartOnDemandScan, the IAM identity must have permission for the `iam:CreateServiceLinkedRole` action. For more information, see Running on-demand machine image scans with Amazon Inspector.

Editing a service-linked role for machine image scanning

Amazon Inspector does not allow you to edit the `AWSServiceRoleForAmazonInspector2Ami` service-linked role. After a service-linked role is created, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role by using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for machine image scanning

If you no longer need to use machine image scanning, we recommend that you delete the `AWSServiceRoleForAmazonInspector2Ami` service-linked role. That way you don't have an unused entity that isn't actively monitored or maintained.

Important

In order to delete the `AWSServiceRoleForAmazonInspector2Ami` role, you must first disable machine image scanning in all Regions where it is enabled.

Important

Amazon Inspector also blocks deletion of the `AWSServiceRoleForAmazonInspector2Ami` role while an on-demand machine image scan is in progress in any AWS Region, so that the scan can finish. If you attempt to delete the role while a scan is in progress, the deletion fails. Wait for your on-demand scans to reach a terminal state, and then try the deletion again. To check whether a scan is still running, use the GetOnDemandScanStatus API operation.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonInspector2Ami service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Service-linked role permissions for Amazon Inspector multi-cloud scans

Amazon Inspector multi-cloud scanning uses the service-linked role named `AWSServiceRoleForAmazonInspector2ThirdParty`. This SLR allows Amazon Inspector to perform actions on your behalf for multi-cloud resource scanning, including managing cloud connectors, configuration recorders, and vulnerability assessments of resources in external cloud environments. This service-linked role trusts the `thirdparty.inspector2.amazonaws.com` service to assume the role.

The permissions policy for the role, which is named `AmazonInspector2ThirdPartyServiceRolePolicy`, allows Amazon Inspector to perform tasks such as:

- Use actions to discover multi-cloud resources through Inspector-owned service views.
- Use AWS Systems Manager actions to manage associations, automation executions, and cloud connectors for multi-cloud VM scanning.
- Use the IAM `PassRole` action to pass Amazon Inspector roles to AWS Systems Manager for automation execution, scoped to roles with names starting with `Inspector2SSM` or `Inspector2VmScanner`.
- Use the IAM `GetRole` action to verify that Amazon Inspector roles exist before passing them.
- Use the IAM `CreateServiceLinkedRole` action to create the AWS Systems Manager service-linked role when required by association creation.
- Use actions to manage service-linked configuration recorders and connectors for multi-cloud resource discovery.
- Use CloudWatch actions to retrieve health metrics for multi-cloud connectors.
- Use AWS STS `GetWebIdentityToken` action for OIDC federation into external cloud environments, scoped to the Azure AD token audience.

For the permissions in this policy, see AmazonInspector2ThirdPartyServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

Creating a service-linked role for multi-cloud scanning

You don't need to manually create the `AWSServiceRoleForAmazonInspector2ThirdParty` service-linked role. When you create a multi-cloud connector in Amazon Inspector, Amazon Inspector automatically creates the service-linked role for you.

Editing a service-linked role for multi-cloud scanning

Amazon Inspector does not allow you to edit the `AWSServiceRoleForAmazonInspector2ThirdParty` service-linked role. After a service-linked role is created, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role by using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for multi-cloud scanning

If you no longer need to use multi-cloud scanning, we recommend that you delete the `AWSServiceRoleForAmazonInspector2ThirdParty` service-linked role. That way you don't have an unused entity that isn't actively monitored or maintained.

> **Important**
>
> In order to delete the `AWSServiceRoleForAmazonInspector2ThirdParty` role, you must first delete all multi-cloud connectors in all Regions where multi-cloud scanning is enabled.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonInspector2ThirdParty service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Troubleshooting Amazon Inspector identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Inspector and IAM.

Topics

- I am not authorized to perform an action in Amazon Inspector

- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amazon Inspector resources

I am not authorized to perform an action in Amazon Inspector

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `inspector2:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 inspector2:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `inspector2:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon Inspector.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Inspector. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon Inspector resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Inspector supports these features, see How Amazon Inspector works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Monitoring Amazon Inspector

Monitoring is an important part of maintaining the availability, reliability, and performance of Amazon Inspector and other AWS solutions. AWS provides tools to monitor Amazon Inspector, report issues that occur, and take actions to remediate these issues:

- Amazon EventBridge is an AWS service that uses events to connect application components together, making it easier for you to build scalable event-driven applications. EventBridge delivers a stream of real-time data from your applications, Software-as-a-Service (SaaS) applications, and AWS services and routes, so you can monitor events that happen in services and build event-driven architectures.
- AWS CloudTrail is an AWS service that captures API calls and related events made by or on behalf of your AWS account. CloudTrail delivers the log files to an Amazon S3 bucket that you specify,

so you can identify which users and accounts called AWS, the source IP address from where calls were made, and when the calls occurred.

Logging Amazon Inspector API calls using AWS CloudTrail

Amazon Inspector is integrated with AWS CloudTrail, a service that provides a record of actions taken by an IAM user or role, or an AWS service, in Amazon Inspector. CloudTrail captures all API calls for Amazon Inspector as events. The calls captured include calls from the Amazon Inspector console and calls to the Amazon Inspector API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Inspector. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine:

- The request that was made to Amazon Inspector.
- The IP address from which the request was made.
- Who made the request.
- When the request was made.

To learn more about CloudTrail, see the *AWS CloudTrail User Guide*.

Amazon Inspector information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Inspector, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Amazon Inspector, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following topics:

- Overview for creating a trail
- CloudTrail supported services and integrations

- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple accounts
- Receiving CloudTrail log files from multiple regions

All Amazon Inspector actions are logged by CloudTrail. All actions that Amazon Inspector can make are documented in the Amazon Inspector API Reference. For example, calls to the `CreateFindingsReport`, `ListCoverage`, and `UpdateOrganizationConfiguration` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or a federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Amazon Inspector log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source. Events include information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

Amazon Inspector Scan information in CloudTrail

Amazon Inspector Scan is integrated with CloudTrail. All Amazon Inspector Scan API operations are logged as management events. For a list of the Amazon Inspector Scan API operations that Amazon Inspector logs to CloudTrail, see Amazon Inspector Scan in the Amazon Inspector API Reference.

The following example shows a CloudTrail log entry that demonstrates the `ScanSbom` action:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
```

```
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:akua_mansa",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/akua_mansa",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-10-17T15:22:59Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2023-10-17T16:02:34Z",
    "eventSource": "gamma-inspector-scan.amazonaws.com",
    "eventName": "ScanSbom",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-sdk-java/2.20.162 Mac_OS_X/13.5.2 OpenJDK_64-
Bit_Server_VM/17.0.8+7-LTS Java/17.0.8 vendor/Amazon.com_Inc. io/sync http/
UrlConnection cfg/retry-mode/legacy",
    "requestParameters": {
        "sbom": {
            "specVersion": "1.5",
            "metadata": {
                "component": {
                    "name": "debian",
                    "type": "operating-system",
                    "version": "9"
                }
            },
            "components": [
                {
                    "name": "packageOne",
                    "purl": "pkg:deb/debian/packageOne@1.0.0?arch=x86_64&distro=9",
                    "type": "application"
                }
```

```
            ],
            "bomFormat": "CycloneDX"
        }
    },
    "responseElements": null,
    "requestID": "f041a27f-f33e-4f70-b09b-5fbc5927282a",
    "eventID": "abc8d1e4-d214-4f07-bc56-8a31be6e36fe",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


Compliance validation for Amazon Inspector

Our new AWS sign-up experience is not designed for regulated workloads. If you're using our new AWS sign-up experience, but you want to use AWS for regulated workloads, you can sign up for AWS (advanced) or activate advanced features for your AWS environment.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in Amazon Inspector

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple, physically separated and isolated Availability Zones, which are connected to low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones

without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

Infrastructure security in Amazon Inspector

As a managed service, Amazon Inspector is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Inspector through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Incident response in Amazon Inspector

Security is the highest priority at AWS. As mentioned in the AWS shared responsibility model under "Security of the Cloud," AWS is responsible for protecting the infrastructure running all of the services in the AWS Cloud. AWS is also responsible for any incident response associated with the Amazon Inspector service.

As an AWS customer, you share a responsibility for maintaining security in the AWS Cloud. This means you control the security you choose to implement, which includes all of the AWS tools and features you access. It also means you're responsible for incident response on your side of the shared responsibility model.

By establishing a security baseline that meets all of the objectives for your applications running in the AWS Cloud, you can detect deviations you can respond to. Because incident response is a complex topic, review the following resources to better understand the impact of incident response and how your choices might influence your corporate goals: AWS Security Incident Response Guide, AWS Security Best Practices, and AWS Cloud Adoption Framework: Security Perspective.

Amazon Inspector Security Research

Amazon Inspector Security Research is a continuous monitoring program that identifies malicious packages published to public package registries. Software supply chain attacks targeting open-source ecosystems are a growing threat, and malicious packages are frequently removed from upstream registries only after they have been downloaded by unsuspecting developers. Amazon Inspector Security Research helps protect your applications by detecting these packages, publishing advisories, and incorporating this intelligence into Amazon Inspector findings so that workloads consuming a known-malicious package are surfaced automatically.

Supported public package registries:

- **npm** – the Node.js package registry (npmjs.com)
- **PyPI** – the Python Package Index (pypi.org)
- **RubyGems** – the Ruby package registry (rubygems.org)
- **crates.io** – the Rust package registry (crates.io)

Research methodology

The Amazon Inspector security research team combines automated detection pipelines with expert analyst review to identify malicious packages across supported registries. Each package that meets the Amazon Inspector criteria for a malicious determination receives an Amazon Inspector advisory with an `IN-MAL` identifier. Amazon Inspector findings use these advisories, so customers are notified when their workloads consume an affected package. No customer action is required to benefit from this research – detections are applied automatically.

Amazon Inspector is a contributing partner in the Open Source Security Foundation (OpenSSF) Malicious Packages Repository. Amazon Inspector also submits advisories to this open dataset, giving the broader open-source community access to the same threat intelligence that Amazon Inspector uses to protect AWS customers. Advisories that also meet the OpenSSF criteria are published in the repository with a `MAL` identifier, which Amazon Inspector records as an alias of the `IN-MAL` identifier. Advisories that meet only the Amazon Inspector criteria keep the `IN-MAL` identifier alone. Both kinds of advisory are used in Amazon Inspector findings.

As of 2026-11-11, Amazon Inspector is credited in 196,691 of the 240,400 records (81.8%) in the OpenSSF Malicious Packages Repository, making it the largest contributor to the dataset.

Detection summary

The following tables summarize the advisories issued by Amazon Inspector Security Research across all supported registries. Each package is counted once, whether its advisory has only an `IN-MAL` identifier or both an `IN-MAL` and a `MAL` identifier. Periods are based on the date Amazon Inspector issued the advisory.

Note

Last updated: 2026-11-11 22:04:14 UTC

Lifetime totals by registry

Registry	Lifetime packages identified
npm	196,132
PyPI	609
RubyGems	77
crates.io	1
Total	**196,819**

Recent activity by registry

Period	Window (UTC)	npm	PyPI	RubyGems	crates.io	Total
This week	2026-11-09 .. 2026-11-15	2	2	0	1	5
Last week	2026-11-02 ..	35	19	9	0	63

Period	Window (UTC)	npm	PyPI	RubyGems	crates.io	Total
	2026-11-08					
This month	2026-11	44	22	10	1	77
Last month	2026-10	518	148	67	0	733

Recent malicious package reports

The following table lists the ten most recent advisories issued by Amazon Inspector Security Research, ordered by the date Amazon Inspector issued the advisory. The advisory ID is the OpenSSF `MAL` identifier when the advisory has one, and the Amazon Inspector `IN-MAL` identifier otherwise.

Package name	Advisory ID	Registry	Published date
aurum-pay	MAL-2026-19147	PyPI	2026-11-11
mock-data-sdk	MAL-2026-19143	npm	2026-11-10
lwd-uri-util	MAL-2026-19058	crates.io	2026-11-10
logger-libs	MAL-2026-18778	PyPI	2026-11-09
@ethersphere/bee-js	MAL-2026-18773	npm	2026-11-09
bundyfed	MAL-2026-18770	PyPI	2026-11-08
super-color	MAL-2026-18768	PyPI	2026-11-08
supercolor-tui	MAL-2026-18767	PyPI	2026-11-08
fcefi-wdojs-util	MAL-2026-19142	npm	2026-11-08

Package name	Advisory ID	Registry	Published date
rcp-ndcg	MAL-2026-18658	PyPI	2026-11-08

Related resources

The following AWS Security Blog posts provide additional context on Amazon Inspector Security Research and recent supply chain threat campaigns:

- Amazon identifies North Korean hacker group behind open-source supply chain attacks
- Amazon Inspector detects over 150,000 malicious packages linked to token farming campaign
- Defending against supply chain attacks like Chalk/Debug and the Shai-Hulud worm
- What AWS Security learned from responding to recent npm supply chain threat campaigns

Access Amazon Inspector using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and Amazon Inspector. You can access Amazon Inspector as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access Amazon Inspector.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Amazon Inspector.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for Amazon Inspector

Before you set up an interface endpoint for Amazon Inspector, review Considerations in the *AWS PrivateLink Guide*.

Amazon Inspector supports making calls to all of its API actions through the interface endpoint.

VPC endpoint policies are not supported for Amazon Inspector. By default, full access to Amazon Inspector is allowed through the interface endpoint. Alternatively, you can associate a security group with the endpoint network interfaces to control traffic to Amazon Inspector through the interface endpoint.

Create an interface endpoint for Amazon Inspector

You can create an interface endpoint for Amazon Inspector using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

When you create an interface endpoint for Amazon Inspector, use one of the following service names:

```
com.amazonaws.region.inspector2
```

```
com.amazonaws.region.inspector-scan
```

Replace *region* with the AWS Region code for the applicable AWS Region.

If you enable private DNS for the interface endpoint, you can make API requests to Amazon Inspector using its default Regional DNS name, for example, `service-name.us-east-1.amazonaws.com` or `service-name.us-east-1.api.aws.com` for the US East (N. Virginia).

Amazon Inspector integrations

Amazon Inspector integrates with other AWS services. These services can ingest data from Amazon Inspector, so you can view your findings in different ways. Review the following integration options to learn more.

Using Amazon Inspector with AWS Organizations

AWS Organizations helps you centrally manage and govern your AWS environment. You can use AWS Organizations policies to enable and manage Amazon Inspector across multiple accounts in your organization automatically.

Amazon Inspector organization policies allow you to:

- Centrally enable Amazon Inspector scan types (EC2, ECR, Lambda, Code Repository) across your organization
- Automatically apply Amazon Inspector enablement to new accounts joining the organization
- Enforce consistent scanning coverage across organizational units
- Prevent member accounts from disabling required scanning

Organization policies control resource type enablement, while delegated administrators retain control over scan configuration settings. For information about how organization policies interact with delegated administrator and member account permissions, see Managing multiple accounts in Amazon Inspector with AWS Organizations. For detailed instructions on creating Amazon Inspector policies, see the AWS Organizations documentation for Amazon Inspector policies.

Integrating Amazon Inspector with Amazon ECR

Amazon Elastic Container Registry (Amazon ECR) is an AWS-managed container image registry that supports private registries. Amazon ECR private registries host container images in a highly-available and scalable architecture. You can use Amazon Inspector to scan container images residing in your Amazon ECR repository for vulnerable operating system packages and programming language packages. For more information, see Amazon Inspector integration with Amazon Elastic Container Registry (Amazon ECR).

Amazon Inspector integration with AWS Security Hub CSPM

AWS Security Hub CSPM provides a comprehensive view of your security state in AWS and helps you check your environment against security industry standards and best practices Security Hub CSPM collects security data from AWS accounts, services, and supported products. You can use Security Hub CSPM to ingest Amazon Inspector findings data and create a central location for findings in all of your integrated AWS services and AWS Partner Network products. For more information, see Amazon Inspector integration with AWS Security Hub CSPM.

Amazon Inspector integration with Amazon Elastic Container Registry (Amazon ECR)

Amazon Elastic Container Registry is a fully managed container registry that supports Docker and OCI images and AWS artifacts. If you use Amazon ECR, you can activate Enhanced Scanning for your container registry. When you activate enhanced scanning, Amazon Inspector automatically detects and scans your container images for vulnerable operating system and programming language packages. This integration allows you to view Amazon Inspector findings for container images and manage the frequency and scope of scans in the Amazon ECR console. For more information, see Scanning Amazon ECR container images with Amazon Inspector.

Activating the integration

You can activate the integration by activating Amazon Inspector scanning through the Amazon Inspector console or API, or by configuring your repository to use **Enhanced scanning** with Amazon Inspector through the Amazon ECR console or API.

For more information on activating the integration through Amazon Inspector, see Automated scan types in Amazon Inspector.

For information on activating and configuring **Enhanced scanning** in Amazon ECR, see Enhanced Scanning in the Amazon ECR user guide.

Using the integration with a multi-account environment

If you are a member in a multi-account environment, you can activate enhanced scanning through Amazon ECR. However, once activated, it can only be deactivated by your Amazon Inspector delegated administrator. If it is deactivated, it reverts to basic scanning. For more information, see Deactivating Amazon Inspector.

Amazon Inspector integration with AWS Security Hub CSPM

Security Hub CSPM provides a comprehensive view of your security state in AWS. This helps you check your environment against security industry standards and best practices. Security Hub CSPM collects security data from AWS accounts, services, and supported products. You can use this information to analyze security trends and identify security issues. When you activate the Amazon Inspector integration with Security Hub CSPM, Amazon Inspector can send findings to Security Hub CSPM, and Security Hub CSPM can analyze those findings as part of your security posture.

Security Hub CSPM tracks security issues as findings. Some findings can be a result of security issues detected in other AWS services or third-party products. Security Hub CSPM uses a set of rules to detect security issues and generate findings and provides tools, so you can manage findings. Security Hub CSPM archives Amazon Inspector findings once the findings have been closed in Amazon Inspector. You can also view a history of your findings and finding details, as well as track the status of an investigation into a finding.

Security Hub CSPM processes findings in the AWS Security Finding Format (ASFF). This format includes details such as unique identifiers, severity levels, affected resources, remediation guidance, workflow status, and contextual information.

> **Note**
>
> Security findings generated by Amazon Inspector Code Security are not available for this integration. However, you can access these particular findings in the Amazon Inspector console and through the Amazon Inspector API.

Topics

- Viewing Amazon Inspector findings in AWS Security Hub CSPM
- Activating and configuring the Amazon Inspector integration with Security Hub CSPM
- Activating Amazon Inspector from Security Hub CSPM using organization policy
- Disabling the flow of findings from an integration
- Viewing security controls for Amazon Inspector in Security Hub CSPM

Viewing Amazon Inspector findings in AWS Security Hub CSPM

You can view Amazon Inspector Classic and Amazon Inspector findings in Security Hub CSPM.

> **Note**
>
> To filter on Amazon Inspector findings only, add `"aws/inspector/ProductVersion": "2"` to the filter bar. This filter excludes Amazon Inspector Classic findings from the Security Hub CSPM dashboard.

Example finding from Amazon Inspector


```
{
  "SchemaVersion": "2018-10-08",
  "Id": "arn:aws:inspector2:us-east-1:123456789012:finding/FINDING_ID",
  "ProductArn": "arn:aws:securityhub:us-east-1::product/aws/inspector",
  "ProductName": "Inspector",
  "CompanyName": "Amazon",
  "Region": "us-east-1",
  "GeneratorId": "AWSInspector",
  "AwsAccountId": "123456789012",
  "Types": [
    "Software and Configuration Checks/Vulnerabilities/CVE"
  ],
  "FirstObservedAt": "2023-01-31T20:25:38Z",
  "LastObservedAt": "2023-05-04T18:18:43Z",
  "CreatedAt": "2023-01-31T20:25:38Z",
  "UpdatedAt": "2023-05-04T18:18:43Z",
  "Severity": {
    "Label": "HIGH",
    "Normalized": 70
  },
  "Title": "CVE-2022-34918 - kernel",
  "Description": "An issue was discovered in the Linux kernel through 5.18.9. A type
confusion bug in nft_set_elem_init (leading to a buffer overflow) could be used by a
local attacker to escalate privileges, a different vulnerability than CVE-2022-32250.
(The attacker can obtain root access, but must start with an unprivileged user
namespace to obtain CAP_NET_ADMIN access.) This can be fixed in nft_setelem_parse_data
in net/netfilter/nf_tables_api.c.",
  "Remediation": {
    "Recommendation": {
      "Text": "Remediation is available. Please refer to the Fixed version in the
vulnerability details section above. For detailed remediation guidance for each of the
affected packages, refer to the vulnerabilities section of the detailed finding JSON."
    }
```

```
  },
  "ProductFields": {
    "aws/inspector/FindingStatus": "ACTIVE",
    "aws/inspector/inspectorScore": "7.8",
    "aws/inspector/resources/1/resourceDetails/awsEc2InstanceDetails/platform":
 "AMAZON_LINUX_2",
    "aws/inspector/ProductVersion": "2",
    "aws/inspector/instanceId": "i-0f1ed287081bdf0fb",
    "aws/securityhub/FindingId": "arn:aws:securityhub:us-east-1::product/aws/inspector/
arn:aws:inspector2:us-east-1:123456789012:finding/FINDING_ID",
    "aws/securityhub/ProductName": "Inspector",
    "aws/securityhub/CompanyName": "Amazon"
  },
  "Resources": [
    {
      "Type": "AwsEc2Instance",
      "Id": "arn:aws:ec2:us-east-1:123456789012:i-0f1ed287081bdf0fb",
      "Partition": "aws",
      "Region": "us-east-1",
      "Tags": {
        "Patch Group": "SSM",
        "Name": "High-SEv-Test"
      },
      "Details": {
        "AwsEc2Instance": {
          "Type": "t2.micro",
          "ImageId": "ami-0cff7528ff583bf9a",
          "IpV4Addresses": [
            "52.87.229.97",
            "172.31.57.162"
          ],
          "KeyName": "ACloudGuru",
          "IamInstanceProfileArn": "arn:aws:iam::123456789012:instance-profile/
AmazonSSMRoleForInstancesQuickSetup",
          "VpcId": "vpc-a0c2d7c7",
          "SubnetId": "subnet-9c934cb1",
          "LaunchedAt": "2022-07-26T21:49:46Z"
        }
      }
    }
  ],
  "WorkflowState": "NEW",
  "Workflow": {
    "Status": "NEW"
```

```
  },
  "RecordState": "ACTIVE",
  "Vulnerabilities": [
    {
      "Id": "CVE-2022-34918",
      "VulnerablePackages": [
        {
          "Name": "kernel",
          "Version": "5.10.118",
          "Epoch": "0",
          "Release": "111.515.amzn2",
          "Architecture": "X86_64",
          "PackageManager": "OS",
          "FixedInVersion": "0:5.10.130-118.517.amzn2",
          "Remediation": "yum update kernel"
        }
      ],
      "Cvss": [
        {
          "Version": "2.0",
          "BaseScore": 7.2,
          "BaseVector": "AV:L/AC:L/Au:N/C:C/I:C/A:C",
          "Source": "NVD"
        },
        {
          "Version": "3.1",
          "BaseScore": 7.8,
          "BaseVector": "CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H",
          "Source": "NVD"
        },
        {
          "Version": "3.1",
          "BaseScore": 7.8,
          "BaseVector": "CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H",
          "Source": "NVD",
          "Adjustments": []
        }
      ],
      "Vendor": {
        "Name": "NVD",
        "Url": "https://nvd.nist.gov/vuln/detail/CVE-2022-34918",
        "VendorSeverity": "HIGH",
        "VendorCreatedAt": "2022-07-04T21:15:00Z",
        "VendorUpdatedAt": "2022-10-26T17:05:00Z"
```

```
      },
      "ReferenceUrls": [
        "https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?
id=7e6bc1f6cabcd30aba0b11219d8e01b952eacbb6",
        "https://lore.kernel.org/netfilter-devel/cd9428b6-7ffb-dd22-d949-
d86f4869f452@randorisec.fr/T/",
        "https://www.debian.org/security/2022/dsa-5191"
      ],
      "FixAvailable": "YES"
    }
  ],
  "FindingProviderFields": {
    "Severity": {
      "Label": "HIGH"
    },
    "Types": [
      "Software and Configuration Checks/Vulnerabilities/CVE"
    ]
  },
  "ProcessedAt": "2023-05-05T20:28:38.822Z"
}
```


Activating and configuring the Amazon Inspector integration with Security Hub CSPM

You can activate the Amazon Inspector integration with AWS Security Hub CSPM by enabling Security Hub CSPM. After you enable Security Hub CSPM, the Amazon Inspector integration with AWS Security Hub CSPM is automatically activated, and Amazon Inspector begins sending all of its findings to Security Hub CSPM using the AWS Security Finding Format (ASFF).

Activating Amazon Inspector from Security Hub CSPM using organization policy

You can manage Amazon Inspector activation across your organization using AWS Organizations policies directly from the Security Hub CSPM console. This centralized approach allows you to enable Amazon Inspector scanning for multiple accounts simultaneously through organization-level policy management.

For detailed instructions on managing Amazon Inspector activation through Security Hub CSPM using organization policies, see Managing delegated administrator accounts for Security Hub CSPM in the *AWS Security Hub CSPM User Guide*.

Disabling the flow of findings from an integration

To stop Amazon Inspector from sending findings to Security Hub CSPM, you can use the Security Hub CSPM console or API and AWS CLI..

Viewing security controls for Amazon Inspector in Security Hub CSPM

Security Hub CSPM analyzes findings from supported AWS and third-party products and runs automated and continuous security checks against rules to generate findings of its own. The rules are represented by security controls, which help you determine whether requirements in a standard are being met.

Amazon Inspector uses security controls to check whether Amazon Inspector features are or should be enabled. These features include the following:

- Amazon EC2 scanning
- Amazon ECR scanning
- Lambda standard scanning
- Lambda code scanning

For more information, see Amazon Inspector controls in the *AWS Security Hub CSPM User Guide*.

Supported operating systems and programming languages for Amazon Inspector

Amazon Inspector can scan software applications that are installed on the following:

- Amazon Elastic Compute Cloud (Amazon EC2) instances

> **Note**
>
> For Amazon EC2 instances, Amazon Inspector can scan for package vulnerabilities in operating systems that support agent-based scanning. Amazon Inspector can also scan for package vulnerabilities in operating systems and programming languages that support hybrid scanning. Amazon Inspector does not scan for toolchain vulnerabilities. The version of the programming language compiler used to build the application introduces these vulnerabilities.

- Machine images (AMIs) that your account owns

> **Note**
>
> For machine images, Amazon Inspector can scan for operating system and programming language package vulnerabilities. For the operating systems that Amazon Inspector supports for machine image scanning, see Supported operating systems: machine image scanning. Amazon Inspector does not scan for toolchain vulnerabilities. The version of the programming language compiler used to build the application introduces these vulnerabilities.

- Container images stored in Amazon Elastic Container Registry (Amazon ECR) repositories

> **Note**
>
> For ECR container images, Amazon Inspector can scan for operating system and programming language package vulnerabilities. Amazon Inspector also supports scanning hardened container images. For more information, see Supported hardened container images. Amazon Inspector does not scan for toolchain vulnerabilities in Rust

——the version of the programming language compiler used to build the application introduces these vulnerabilities.

- AWS Lambda functions

> **Note**
>
> For Lambda functions, Amazon Inspector can scan for programming language package vulnerabilities and code vulnerabilities. Amazon Inspector does not scan for toolchain vulnerabilities. The version of the programming language compiler used to build the application introduces these vulnerabilities.

When Amazon Inspector scans resources, Amazon Inspector sources more than 50 data feeds to generate findings for common vulnerabilities and exposures (CVEs). Examples of these sources include vendor security advisories, data feeds, and threat intelligence feeds, as well as the National Vulnerability Database (NVD) and MITRE. Amazon Inspector updates vulnerability data from source feeds at least once daily.

For Amazon Inspector to scan a resource, the resource must be running a supported operating system or using a supported programming language. The topics in this section list the operating systems, programming languages, and runtimes Amazon Inspector supports for different resources and scan types. They also list discontinued operating systems.

> **Note**
>
> Amazon Inspector can provide only limited support for an operating system after a vendor discontinues support for the operating system.

Topics

- Supported operating systems
- Discontinued operating systems
- Supported programming languages
- Supported runtimes

Supported operating systems

This section lists the operating systems Amazon Inspector supports. Each table specifies the advisory source and advisory status for each operating system.

The **Advisory status** column lists the vendor advisory statuses that Amazon Inspector uses to generate findings for an operating system. The column can contain the following values:

Fixed

The vendor has released a fix for the vulnerability.

Open

The vendor has acknowledged the vulnerability but hasn't released a fix.

Active

The vendor is tracking the vulnerability, and a fix isn't available yet.

Pending

The vendor is preparing a fix that hasn't been released yet.

For some operating systems, the **Advisory source** includes additional repositories:

RHEL

The advisory source includes the BaseOS repository and the EUS, E2S, and E4S repositories. For more information, see Red Hat Enterprise Linux Life Cycle on the Red Hat website.

Ubuntu

The advisory source includes Ubuntu Pro (esm-infra and esm-apps). For more information, see Ubuntu Security Notices and Expanded Security Maintenance on the Ubuntu website.

Supported operating systems: Amazon EC2 scanning

The following table lists the operating systems Amazon Inspector supports for the scanning of Amazon EC2 instances. It specifies the advisory source and advisory status for each operating system and which operating systems support agent-based scanning and agentless scanning.

When using the agent-based scanning method, you configure the SSM agent to perform continuous scans on all eligible instances. Amazon Inspector recommends that you configure a version of the SSM agent that's greater than 3.2.2086.0. For more information, see Working with the SSM Agent in the *Amazon EC2 Systems Manager User Guide*.

Linux operating system detections are supported only for the default package manager repository (rpm and dpkg). Detections don't include third-party applications, extended support repositories, or optional repositories (application streams) unless otherwise specified below. Amazon Inspector scans the running kernel for vulnerabilities. For some operating systems, like Ubuntu, a reboot is required for upgrades to show in active findings.

For definitions of the advisory status values and details about advisory sources, see Supported operating systems.

Operating system	Version	Advisory source	Advisory status	Agentless scan support	Agent-based scan support
AlmaLinux	8	CVE Errata	Fixed	Yes	Yes
AlmaLinux	9	CVE Errata	Fixed	Yes	Yes
AlmaLinux	10	CVE Errata	Fixed	Yes	Yes
Amazon Linux 2023 (AL2023)	AL2023	CVE Errata	Fixed	Yes	Yes
Bottlerocket	1.7.0 and later	CVE Errata	Fixed	Yes	Yes
Debian Server (Bookworm)	12	DSA CVE	Fixed, Open	Yes	Yes
Debian Server (Trixie)	13	DSA CVE	Fixed, Open	Yes	Yes
Fedora	43	CVE Errata	Fixed	Yes	Yes

Operating system	Version	Advisory source	Advisory status	Agentless scan support	Agent-based scan support
Fedora	44	CVE Errata	Fixed	Yes	Yes
Oracle Linux	8	CVE Errata	Fixed	Yes	Yes
Oracle Linux	9	CVE Errata	Fixed	Yes	Yes
Oracle Linux	10	CVE Errata	Fixed	Yes	Yes
Red Hat Enterprise Linux (RHEL)	8	RHEL VEX CVE	Fixed	Yes	Yes
Red Hat Enterprise Linux (RHEL)	9	RHEL VEX CVE	Fixed	Yes	Yes
Red Hat Enterprise Linux (RHEL)	10	RHEL VEX CVE	Fixed	Yes	Yes
Rocky Linux	8	CVE Errata	Fixed	Yes	Yes
Rocky Linux	9	CVE Errata	Fixed	Yes	Yes
Rocky Linux	10	CVE Errata	Fixed	Yes	Yes
SUSE Linux Enterprise Server (SLES)	15.7	CVE Errata	Fixed	Yes	Yes
SUSE Linux Enterprise Server (SLES)	16.0	CVE Errata	Fixed	Yes	Yes
Ubuntu (Bionic)	18.04	USN	Fixed, Active, Pending	Yes	Yes

Operating system	Version	Advisory source	Advisory status	Agentless scan support	Agent-based scan support
Ubuntu (Focal)	20.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Jammy)	22.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Noble)	24.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Resolute)	26.04	USN	Fixed, Active, Pending	Yes	Yes
Windows Server	2016	MSKB	Fixed	Yes	Yes
Windows Server	2019	MSKB	Fixed	Yes	Yes
Windows Server	2022	MSKB	Fixed	Yes	Yes
Windows Server	2025	MSKB	Fixed	Yes	Yes
macOS (Mojave)	10.14	APPLE-SA	Fixed	No	Yes
macOS (Catalina)	10.15	APPLE-SA	Fixed	No	Yes
macOS (Big Sur)	11	APPLE-SA	Fixed	No	Yes
macOS (Monterey)	12	APPLE-SA	Fixed	No	Yes

Operating system	Version	Advisory source	Advisory status	Agentless scan support	Agent-based scan support
macOS (Ventura)	13	APPLE-SA	Fixed	No	Yes
macOS (Sonoma)	14	APPLE-SA	Fixed	No	Yes
macOS (Sequoia)	15	APPLE-SA	Fixed	No	Yes
macOS (Tahoe)	26	APPLE-SA	Fixed	No	Yes

Supported operating systems: machine image scanning

The following table lists the operating systems Amazon Inspector supports for the scanning of machine images (AMIs). It specifies the advisory source and advisory status for each operating system.

Machine image scanning doesn't use an agent. Amazon Inspector reads the software inventory directly from the Amazon EBS snapshots that back the image, so you don't need to install the SSM Agent or launch an instance from the image. For more information, see Scanning machine images with Amazon Inspector.

Linux operating system detections are supported only for the default package manager repository (rpm and dpkg). Detections don't include third-party applications, extended support repositories, or optional repositories (application streams) unless otherwise specified below.

The snapshots that back the machine image must use a supported file system format: `ext3`, `ext4`, `ntfs`, or `xfs`. For more information about supported file system formats, see Eligible machine images.

> **Note**
>
> Amazon Inspector doesn't support machine image scanning for macOS. Amazon Inspector reports a scan status of `UNSUPPORTED_OS` for a machine image that runs an operating

system that isn't listed in the following table, and for a machine image whose snapshots use a configuration that Amazon Inspector can't read, such as Logical Volume Manager (LVM).

For definitions of the advisory status values and details about advisory sources, see Supported operating systems.

Supported operating systems for machine image scanning

Operating system	Version	Advisory source	Advisory status
AlmaLinux	8	CVE Errata	Fixed
AlmaLinux	9	CVE Errata	Fixed
AlmaLinux	10	CVE Errata	Fixed
Amazon Linux 2023 (AL2023)	AL2023	CVE Errata	Fixed
Bottlerocket	1.7.0 and later	CVE Errata	Fixed
Debian Server (Bookworm)	12	DSA CVE	Fixed, Open
Debian Server (Trixie)	13	DSA CVE	Fixed, Open
Fedora	43	CVE Errata	Fixed
Fedora	44	CVE Errata	Fixed
Oracle Linux	8	CVE Errata	Fixed
Oracle Linux	9	CVE Errata	Fixed
Oracle Linux	10	CVE Errata	Fixed
Red Hat Enterprise Linux (RHEL)	8	RHEL VEX CVE	Fixed

Operating system	Version	Advisory source	Advisory status
Red Hat Enterprise Linux (RHEL)	9	RHEL VEX CVE	Fixed
Red Hat Enterprise Linux (RHEL)	10	RHEL VEX CVE	Fixed
Rocky Linux	8	CVE Errata	Fixed
Rocky Linux	9	CVE Errata	Fixed
Rocky Linux	10	CVE Errata	Fixed
SUSE Linux Enterprise Server (SLES)	15.7	CVE Errata	Fixed
SUSE Linux Enterprise Server (SLES)	16.0	CVE Errata	Fixed
Ubuntu (Bionic)	18.04	USN	Fixed, Active, Pending
Ubuntu (Focal)	20.04	USN	Fixed, Active, Pending
Ubuntu (Jammy)	22.04	USN	Fixed, Active, Pending
Ubuntu (Noble)	24.04	USN	Fixed, Active, Pending
Ubuntu (Resolute)	26.04	USN	Fixed, Active, Pending
Windows Server	2016	MSKB	Fixed
Windows Server	2019	MSKB	Fixed
Windows Server	2022	MSKB	Fixed
Windows Server	2025	MSKB	Fixed

Supported operating systems: Amazon ECR scanning with Amazon Inspector

The following table lists the operating systems Amazon Inspector supports for the scanning of container images in Amazon ECR repositories. It also specifies the advisory source and advisory status for each operating system.

For definitions of the advisory status values and details about advisory sources, see Supported operating systems.

Operating system	Version	Advisory source	Advisory status	Basic scanning	Enhanced scanning
AlmaLinux	8	CVE Errata	Fixed	Yes	Yes
AlmaLinux	9	CVE Errata	Fixed	Yes	Yes
AlmaLinux	10	CVE Errata	Fixed	Yes	Yes
Alpine Linux	3.22	CVE Errata	Fixed	Yes	Yes
Alpine Linux	3.23	CVE Errata	Fixed	Yes	Yes
Alpine Linux	3.24	CVE Errata	Fixed	Yes	Yes
Amazon Linux 2023 (AL2023)	AL2023	CVE Errata	Fixed	Yes	Yes
Azure Linux	3	CVE Errata	Fixed	Yes	Yes
BusyBox	–	MITRE CVE	Fixed	Yes	Yes
Debian Server (Bookworm)	12	DSA CVE	Fixed, Open	Yes	Yes
Debian Server (Trixie)	13	DSA CVE	Fixed, Open	Yes	Yes

Operating system	Version	Advisory source	Advisory status	Basic scanning	Enhanced scanning
Fedora	43	CVE Errata	Fixed	Yes	Yes
Fedora	44	CVE Errata	Fixed	Yes	Yes
Fedora	45	CVE Errata	Fixed	Yes	No
Oracle Linux	8	CVE Errata	Fixed	Yes	Yes
Oracle Linux	9	CVE Errata	Fixed	Yes	Yes
Oracle Linux	10	CVE Errata	Fixed	Yes	Yes
Photon OS	4	CVE Errata	Fixed	Yes	Yes
Photon OS	5	CVE Errata	Fixed	Yes	Yes
Red Hat Enterprise Linux (RHEL)	8	RHEL VEX CVE	Fixed	Yes	Yes
Red Hat Enterprise Linux (RHEL)	9	RHEL VEX CVE	Fixed	Yes	Yes
Red Hat Enterprise Linux (RHEL)	10	RHEL VEX CVE	Fixed	Yes	Yes
Rocky Linux	8	CVE Errata	Fixed	Yes	Yes
Rocky Linux	9	CVE Errata	Fixed	Yes	Yes
Rocky Linux	10	CVE Errata	Fixed	Yes	Yes
SUSE Linux Enterprise Server (SLES)	15.7	CVE Errata	Fixed	Yes	Yes

Operating system	Version	Advisory source	Advisory status	Basic scanning	Enhanced scanning
SUSE Linux Enterprise Server (SLES)	16.0	CVE Errata	Fixed	Yes	Yes
Ubuntu (Bionic)	18.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Focal)	20.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Jammy)	22.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Noble)	24.04	USN	Fixed, Active, Pending	Yes	Yes
Ubuntu (Resolute)	26.04	USN	Fixed, Active, Pending	Yes	Yes
Windows Server	2019	MSKB	Fixed	Yes	No
Windows Server	2022	MSKB	Fixed	Yes	No
Windows Server	2025	MSKB	Fixed	Yes	No

Supported hardened container images

The following table lists the hardened container images that Amazon Inspector supports for Amazon ECR scanning.

Operating system	Version	Advisory source	Advisory status	Basic scanning	Enhanced scanning
Chainguard	–	OSV	Fixed	Yes	Yes
CleanStart	–	OSV	Fixed	Yes	No
Echo	2	CVE Errata	Fixed	Yes	Yes
Hummingbird OS	–	RHEL VEX CVE	Fixed	Yes	No
Wolfi	–	OSV	Fixed	Yes	Yes

Supported operating systems: CIS scanning

The following table lists the operating systems Amazon Inspector supports for CIS scans. It also specifies the CIS benchmark version for each operating system.

> **Note**
>
> CIS standards are intended for x86_64 operating systems. Some checks might not be evaluated or might return invalid remediation instructions on ARM-based resources.

Operating system	Version	CIS benchmark version
Amazon Linux 2023	AL2023	1.0.0
Red Hat Enterprise Linux (RHEL)	8	4.0.0
Red Hat Enterprise Linux (RHEL)	9	2.0.0
Rocky Linux	8	3.0.0
Rocky Linux	9	2.0.0

Operating system	Version	CIS benchmark version
SUSE Linux Enterprise Server	15	2.0.1
Ubuntu (Focal)	20.04	3.0.0
Ubuntu (Jammy)	22.04	3.0.0
Ubuntu (Noble)	24.04	2.0.0
Windows Server	2016	4.0.0
Windows Server	2019	5.0.0
Windows Server	2022	5.0.0
Windows Server	2025	2.0.0

Supported operating systems: Amazon Inspector Scan API

The following table lists the supported operating systems for the Amazon Inspector Scan API. For more information, see ScanSbom in the *Amazon Inspector V2 API Reference*.

Operating system	Version
AlmaLinux	8
AlmaLinux	9
AlmaLinux	10
Alpine Linux	3.22
Alpine Linux	3.23
Alpine Linux	3.24
Amazon Linux 2023 (AL2023)	AL2023
Azure Linux	3

Operating system	Version
Azure Linux	4
Bottlerocket	–
BusyBox	1.36.0+
Chainguard	–
CleanStart	–
Debian Server (Bookworm)	12
Debian Server (Trixie)	13
Debian Sid	–
Echo	2
Fedora	43
Fedora	44
Fedora	45
Hummingbird OS	–
macOS	11+
Oracle Linux	8
Oracle Linux	9
Oracle Linux	10
Photon OS	4
Photon OS	5
Red Hat Enterprise Linux (RHEL)	8

Operating system	Version
Red Hat Enterprise Linux (RHEL)	9
Red Hat Enterprise Linux (RHEL)	10
Rocky Linux	8
Rocky Linux	9
Rocky Linux	10
SUSE Linux Enterprise Server (SLES)	15.7
SUSE Linux Enterprise Server (SLES)	16.0
Ubuntu (Bionic)	18.04
Ubuntu (Focal)	20.04
Ubuntu (Jammy)	22.04
Ubuntu (Noble)	24.04
Ubuntu (Resolute)	26.04
Ubuntu (Stonking)	26.10
Wolfi	–
Windows Server	2016
Windows Server	2019
Windows Server	2022
Windows Server	2025

Discontinued operating systems

The following table lists operating systems that have been discontinued and when they were discontinued.

Even though Amazon Inspector doesn't provide full support for discontinued operating systems, Amazon Inspector continues to scan Amazon EC2 instances and Amazon ECR container images running them. As a security best practice, Amazon Inspector generates a CRITICAL finding for resources using a discontinued operating system and recommends moving to a supported version. Findings that Amazon Inspector generates for discontinued operating systems should be used for informational purposes only.

In accordance with vendor policy, discontinued operating systems no longer receive patch updates or security advisories. Vendors can also remove existing security advisories and detections from their feeds for operating systems that reach the end of standard support. As a result, Amazon Inspector stops generating findings for discontinued operating systems 12 months after the associated dates listed below.

Operating system	Version	Discontinued
Alpine Linux	3.2	May 1, 2017
Alpine Linux	3.3	November 1, 2017
Alpine Linux	3.4	May 1, 2018
Alpine Linux	3.5	November 1, 2018
Alpine Linux	3.6	May 1, 2019
Alpine Linux	3.7	November 1, 2019
Alpine Linux	3.8	May 1, 2020
Alpine Linux	3.9	November 1, 2020
Alpine Linux	3.10	May 1, 2021
Alpine Linux	3.11	November 1, 2021

Operating system	Version	Discontinued
Alpine Linux	3.12	May 1, 2022
Alpine Linux	3.13	November 1, 2022
Alpine Linux	3.14	May 1, 2023
Alpine Linux	3.15	November 1, 2023
Alpine Linux	3.16	May 23, 2024
Alpine Linux	3.17	November 22, 2024
Alpine Linux	3.18	May 9, 2025
Alpine Linux	3.19	November 1, 2025
Alpine Linux	3.20	April 1, 2026
Alpine Linux	3.21	November 1, 2026
Amazon Linux (AL1)	2012	December 31, 2021
Amazon Linux 2 (AL2)	AL2	June 30, 2026
CentOS Linux (CentOS)	7	June 30, 2024
CentOS Linux (CentOS)	8	December 31, 2021
Debian Server (Jessie)	8	June 30, 2020
Debian Server (Stretch)	9	June 30, 2022
Debian Server (Buster)	10	June 30, 2024
Debian Server (Bullseye)	11	August 31, 2026
Fedora	33	November 30, 2021
Fedora	34	June 7, 2022

Operating system	Version	Discontinued
Fedora	35	December 13, 2022
Fedora	36	May 16, 2023
Fedora	37	December 15, 2023
Fedora	38	May 21, 2024
Fedora	39	November 26, 2024
Fedora	40	May 13, 2025
Fedora	41	November 19, 2025
Fedora	42	May 13, 2026
MinimOS	–	October 22, 2026
OpenSUSE Leap	15.2	December 1, 2021
OpenSUSE Leap	15.3	December 1, 2022
OpenSUSE Leap	15.4	December 7, 2023
OpenSUSE Leap	15.5	December 31, 2024
OpenSUSE Leap	15.6	April 30, 2026
Oracle Linux	6	March 1, 2021
Oracle Linux	7	December 31, 2024
Photon OS	2	December 2, 2021
Photon OS	3	March 1, 2024
Red Hat Enterprise Linux (RHEL)	6	June 30, 2020
Red Hat Enterprise Linux (RHEL)	7	June 30, 2024

Operating system	Version	Discontinued
SUSE Linux Enterprise Server (SLES)	12	June 30, 2016
SUSE Linux Enterprise Server (SLES)	12.1	May 31, 2017
SUSE Linux Enterprise Server (SLES)	12.2	March 31, 2018
SUSE Linux Enterprise Server (SLES)	12.3	June 30, 2019
SUSE Linux Enterprise Server (SLES)	12.4	June 30, 2020
SUSE Linux Enterprise Server (SLES)	12.5	October 31, 2024
SUSE Linux Enterprise Server (SLES)	15	December 31, 2019
SUSE Linux Enterprise Server (SLES)	15.1	January 31, 2021
SUSE Linux Enterprise Server (SLES)	15.2	December 31, 2021
SUSE Linux Enterprise Server (SLES)	15.3	December 31, 2022
SUSE Linux Enterprise Server (SLES)	15.4	December 31, 2023
SUSE Linux Enterprise Server (SLES)	15.5	December 31, 2024
SUSE Linux Enterprise Server (SLES)	15.6	December 31, 2025
Ubuntu (Precise)	12.04	April 28, 2017
Ubuntu (Trusty)	14.04	April 1, 2024
Ubuntu (Xenial)	16.04	April 1, 2026
Ubuntu (Groovy)	20.10	July 22, 2021
Ubuntu (Hirsute)	21.04	January 20, 2022
Ubuntu (Impish)	21.10	July 31, 2022
Ubuntu (Kinetic)	22.10	July 20, 2023

Operating system	Version	Discontinued
Ubuntu (Lunar)	23.04	January 25, 2024
Ubuntu (Mantic)	23.10	July 11, 2024
Ubuntu (Oracular)	24.10	July 10, 2025
Ubuntu (Plucky)	25.04	January 15, 2026
Ubuntu (Questing)	25.10	July 9, 2026
Windows Server	2012	October 10, 2023
Windows Server	2012 R2	October 10, 2023

Supported programming languages

This section lists the programming languages Amazon Inspector supports.

Supported programming languages: Amazon EC2 agentless scanning

Amazon Inspector currently supports the following programming languages when performing agentless scans on eligible Amazon EC2 instances. For more information, see agentless scanning.

> **Note**
>
> Amazon Inspector doesn't scan for toolchain vulnerabilities in Go and Rust. The version of the programming language compiler used to build the application introduces these vulnerabilities.

- C#
- Go
- Java
- JavaScript
- PHP

- Python
- Ruby
- Rust

Supported programming languages: Amazon Inspector VM Scanner

The Amazon Inspector VM Scanner supports the same programming languages as Amazon EC2 agentless scanning. For the list of supported languages, see Supported programming languages: Amazon EC2 agentless scanning.

Supported programming languages: Amazon Inspector SSM plugin

Amazon Inspector currently supports the following programming languages when performing deep inspection scans on Amazon EC2 Linux instances using the Amazon Inspector SSM plugin. Deep inspection through the Amazon Inspector SSM plugin supports a subset of the programming languages supported by the Amazon Inspector VM Scanner. For more information, see Amazon Inspector deep inspection for Linux-based Amazon EC2 instances.

The languages listed here apply to deep inspection through the Amazon Inspector SSM plugin on Linux instances. For deep inspection with Enhanced EC2 Scanning (the Amazon Inspector VM Scanner), see Amazon Inspector VM Scanner. The Amazon Inspector VM Scanner supports Linux, Windows, and macOS instances.

- Java (.ear, .jar, .par, and .war archive formats)
- JavaScript
- Python

Amazon Inspector uses Systems Manager Distributor to deploy the plugin for deep inspection of your Amazon EC2 instance.

> **Note**
>
> Deep inspection is not supported for Bottlerocket operating systems.

To perform deep inspection scans, Systems Manager Distributor and Amazon Inspector must support your Amazon EC2 instance operating system. For information about supported operating

systems in Systems Manager Distributor, see Supported package platforms and architectures in the *Systems Manager User Guide*.

Supported programming languages: Amazon ECR scanning

Amazon Inspector currently supports the following programming languages when scanning container images in Amazon ECR repositories:

> **Note**
>
> Amazon Inspector doesn't scan for toolchain vulnerabilities in Rust. The version of the programming language compiler used to build the application introduces these vulnerabilities. For Python, Java, and JavaScript applications using Chainguard Libraries or Echo Libraries, Amazon Inspector recognizes the back-ported security fixes and excludes them from findings. For more information, see Chainguard Libraries on the Chainguard website and Echo Libraries on the Echo website.

- C#
- Go
- Go toolchain
- Java (including Chainguard and Echo Libraries)
- Java JDK
- JavaScript (including Echo Libraries)
- PHP
- Python (including Chainguard and Echo Libraries)
- Ruby
- Rust

Supported runtimes

This section lists the runtimes Amazon Inspector supports.

Supported runtimes: Amazon Inspector Lambda standard scanning

Amazon Inspector Lambda standard scanning currently supports the following runtimes for the programming languages it can use when scanning Lambda functions for vulnerabilities in third-party software packages:

> **Note**
>
> Amazon Inspector doesn't scan for toolchain vulnerabilities in Rust. The version of the programming language compiler used to build the application introduces these vulnerabilities.

- Go
  - go1.x
- Java
  - java8
  - java8.al2
  - java11
  - java17
  - java21
  - java25
- .NET
  - .NET 6
  - .NET 8
  - .NET 10
- Node.js
  - nodejs12.x
  - nodejs14.x
  - nodejs16.x
  - nodejs18.x
  - nodejs20.x
  - nodejs22.x

- nodejs24.x
- Python
  - python3.7
  - python3.8
  - python3.9
  - python3.10
  - python3.11
  - python3.12
  - python3.13
  - python3.14
- Ruby
  - ruby2.7
  - ruby3.2
  - ruby3.3
- Custom runtimes
  - AL2
  - AL2023

Supported runtimes: Amazon Inspector Lambda code scanning

Amazon Inspector Lambda code scanning currently supports the following runtimes for the programming languages it can use when scanning Lambda functions for vulnerabilities in code:

- Java
  - java8
  - java8.al2
  - java11
  - java17
  - java21
  - java25
- .NET
  - .NET 6

- .NET 8
- Node.js
  - nodejs12.x
  - nodejs14.x
  - nodejs16.x
  - nodejs18.x
  - nodejs20.x
  - nodejs22.x
- Python
  - python3.7
  - python3.8
  - python3.9
  - python3.10
  - python3.11
  - python3.12
  - python3.13
  - python3.14
- Ruby
  - ruby2.7
  - ruby3.2
  - ruby3.3

Deactivating Amazon Inspector

You can deactivate Amazon Inspector in the Amazon Inspector console or with the Amazon Inspector API. If you deactivate all scan types for an account;, Amazon Inspector is deactivated for that account automatically.

If you deactivate Amazon Inspector for an account, all scan types are deactivated for that account. Additionally, all Amazon Inspector scan settings, inclduing filters, suppression rules, and findings are deleted for the account.

When you deactivate Amazon Inspector Amazon EC2 scanning,Amazon Inspector deletes the following SSM associations:

- `InspectorDistributor-do-not-delete`
- `InspectorInventoryCollection-do-not-delete`
- `InvokeInspectorSsmPlugin-do-not-delete`. Additionally, the Amazon Inspector SSM plugin installed through this association is removed from all of your Windows hosts. For more information, see Scanning Windows EC2 instance.

> **Note**
>
> Once you deactivate Amazon Inspector, you no longer incur service charges. However, you can reactivate Amazon Inspector at any time.

For information about how to deactivate scan types for different resources, see Deactivating a scan type.

Prerequisites

Depending on the account type, consider the following:

- If your account is a standalone Amazon Inspector account, you can deactivate Amazon Inspector at any time.
- If your account is a member account in a multi-account environment, you cannot deactivate Amazon Inspector. You must contact the delegated administrator for your organization to deactivate Amazon Inspector.

- If you're the delegated administrator for an organization, you must disassociate all of your member accounts before you deactivate Amazon Inspector.
- If your account's Amazon Inspector enablement is managed by AWS Organizations policies, you cannot deactivate policy-managed scan types through the Amazon Inspector console or API. To deactivate Amazon Inspector scan types, you must modify the organization policy to explicitly disable them through the AWS Organizations console or API. You can deactivate scan types that are not managed by organization policies through the Amazon Inspector console or API.

> **Note**
>
> When you deactivate Amazon Inspector as the delegated administrator, you deactivate the auto-activate feature for your organization.

Deactivating Amazon Inspector managed by organization policies

If Amazon Inspector is enabled in your accounts through AWS Organizations policies, you must use the AWS Organizations console or API to disable Inspector. Member accounts and delegated administrators cannot disable policy-managed scan types through the Amazon Inspector console or API.

To deactivate Amazon Inspector for policy-managed accounts:

To deactivate policy-managed Amazon Inspector enablement

1. Sign in to the AWS Organizations management account or policy administrator account.
2. Modify the organization policy to explicitly set the scan types to disabled in the regions where you want to disable Inspector. You must update the policy content to specify disabled regions for the scan types you want to deactivate.
3. AWS Organizations will automatically apply the policy changes, and Amazon Inspector will disable the specified scan types in the affected accounts.

For detailed instructions on modifying or detaching organization policies, see the AWS Organizations documentation for Amazon Inspector policies.

> **Note**
>
> When you detach an organization policy from accounts, those accounts retain their current Amazon Inspector settings (enabled or disabled based on the last applied policy). The accounts are no longer managed by the policy and can then manage their Amazon Inspector settings independently or through the delegated administrator.

Deactivate Amazon Inspector

> **Note**
>
> Before you deactivate Amazon Inspector, consider exporting your findings.

Console

To deactivate Amazon Inspector

1. Sign in using your credentials, and then open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. By using the AWS Region selector in the upper-right corner of the page, choose the Region in which you want to deactivate Amazon Inspector.
3. In the navigation pane, choose **General settings**.
4. Choose **Deactivate Inspector**.
5. When prompted for confirmation, enter **deactivate** in the text box, and then choose **Deactivate Inspector**.
6. (Recommended) Repeat these steps in each Region for which you want to deactivate Amazon Inspector.

API

Run the Disable API operation. In the request, provide the account IDs you are deactivating, and `EC2, ECR, LAMBDA` for `resourceTypes` to deactivate all scans, which will deactivate the account.

Amazon Inspector quotas

This section lists Amazon Inspector quotas per AWS Region.

Resource	Default	Comments
Member accounts	10,000	The maximum number of member accounts associated with an Amazon Inspector delegated administrator account. The limit is based on Quotas for AWS Organizations.
Suppression rules	500	The maximum number of saved suppression rules per AWS account per Region. You cannot request a quota increase.
Amazon EC2 network findings	10,000	The maximum number of Amazon EC2 network findings per AWS account. You cannot request a quota increase.
CIS scan configurations	500	The maximum number of CIS scan configurations. You cannot request a quota increase.

For a list of quotas associated with Amazon Inspector Classic, see Amazon Inspector Classic service quotas in the *AWS General Reference*. For a list of quotas associated with AWS Organizations, see AWS Organizations service quotas in the *AWS General Reference*.

Regions and endpoints

This topic includes tables that show endpoints for Amazon Inspector and Amazon Inspector Scan. It also includes tables that show which AWS Regions support Amazon Inspector features. To view the AWS Regions where Amazon Inspector is available, see Amazon Inspector endpoint and quotas in the *Amazon Web Services General Reference*.

Service endpoints for Amazon Inspector

The following table shows the service endpoints for Amazon Inspector. The naming convention for Amazon Inspector endpoints is `inspector2.Region.amazonaws.com`.

Region Name	Region	Endpoint	Protocol
US East (Ohio)	us-east-2	inspector2.us-east-2.amazonaws.com	HTTPS
		inspector2-fips.us-east-2.amazonaws.com	HTTPS
US East (N. Virginia)	us-east-1	inspector2.us-east-1.amazonaws.com	HTTPS
		inspector2-fips.us-east-1.amazonaws.com	HTTPS
US West (N. California)	us-west-1	inspector2.us-west-1.amazonaws.com	HTTPS
		inspector2-fips.us-west-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	inspector2.us-west-2.amazonaws.com	HTTPS
		inspector2-fips.us-west-2.amazonaws.com	HTTPS
Africa (Cape Town)	af-south-1	inspector2.af-south-1.amazonaws.com	HTTPS
Asia Pacific	ap-east-1	inspector2.ap-east-1.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
(Hong Kong)			
Asia Pacific (Hyderabad)	ap-south-2	inspector2.ap-south-2.amazonaws.com	HTTPS
Asia Pacific (Jakarta)	ap-southeast-3	inspector2.ap-southeast-3.amazonaws.com	HTTPS
Asia Pacific (Malaysia)	ap-southeast-5	inspector2.ap-southeast-5.amazonaws.com	HTTPS
Asia Pacific (Melbourne)	ap-southeast-4	inspector2.ap-southeast-4.amazonaws.com	HTTPS
Asia Pacific (Mumbai)	ap-south-1	inspector2.ap-south-1.amazonaws.com	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	inspector2.ap-northeast-3.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	inspector2.ap-northeast-2.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Singapore)	ap-southeast-1	inspector2.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	inspector2.ap-southeast-2.amazonaws.com	HTTPS
Asia Pacific (Taipei)	ap-east-2	inspector2.ap-east-2.amazonaws.com	HTTPS
Asia Pacific (Thailand)	ap-southeast-7	inspector2.ap-southeast-7.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	inspector2.ap-northeast-1.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	inspector2.ca-central-1.amazonaws.com	HTTPS
Canada West (Calgary)	ca-west-1	inspector2.ca-west-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	inspector2.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	inspector2.eu-west-1.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Europe (London)	eu-west-2	inspector2.eu-west-2.amazonaws.com	HTTPS
Europe (Milan)	eu-south-1	inspector2.eu-south-1.amazonaws.com	HTTPS
Europe (Paris)	eu-west-3	inspector2.eu-west-3.amazonaws.com	HTTPS
Europe (Spain)	eu-south-2	inspector2.eu-south-2.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	inspector2.eu-north-1.amazonaws.com	HTTPS
Europe (Zurich)	eu-central-2	inspector2.eu-central-2.amazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	inspector2.il-central-1.amazonaws.com	HTTPS
Mexico (Central)	mx-central-1	inspector2.mx-central-1.amazonaws.com	HTTPS
Middle East (Bahrain)	me-south-1	inspector2.me-south-1.amazonaws.com	HTTPS
Middle East (UAE)	me-central-1	inspector2.me-central-1.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
South America (São Paulo)	sa-east-1	inspector2.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-East)	us-gov-east-1	inspector2.us-gov-east-1.amazonaws.com inspector2-fips.us-gov-east-1.amazonaws.com	HTTPS HTTPS
AWS GovCloud (US-West)	us-gov-west-1	inspector2.us-gov-west-1.amazonaws.com inspector2-fips.us-gov-west-1.amazonaws.com	HTTPS HTTPS

Endpoints for Amazon Inspector Scan API

The following table shows the Regional endpoints that can be used when calling the Amazon Inspector Scan API. When using the API, you must supply the endpoint and it's corresponding Region for the AWS Region you're currently authenticated to.

The naming convention for Amazon Inspector Scan endpoints is `inspector-scan.`*`region`*`.amazonaws.com`. For example, if you are authenticated in `us-west-2`, you would use the endpoint `inspector-scan.us-west-2.amazonaws.com` to call the `inspector-scan` API.

Region Name	Region	Endpoint	Protocol
US East (Ohio)	us-east-2	inspector-scan.us-east-2.amazonaws.com inspector-scan-fips.us-east-2.amazonaws.com	HTTPS HTTPS

Region Name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	inspector-scan.us-east-1.amazonaws.com inspector-scan-fips.us-east-1.amazonaws.com	HTTPS HTTPS
US West (N. California)	us-west-1	inspector-scan.us-west-1.amazonaws.com inspector-scan-fips.us-west-1.amazonaws.com	HTTPS HTTPS
US West (Oregon)	us-west-2	inspector-scan.us-west-2.amazonaws.com inspector-scan-fips.us-west-2.amazonaws.com	HTTPS HTTPS
Africa (Cape Town)	af-south-1	inspector-scan.af-south-1.amazonaws.com	HTTPS
Asia Pacific (Hong Kong)	ap-east-1	inspector-scan.ap-east-1.amazonaws.com	HTTPS
Asia Pacific (Hyderabad)	ap-south-2	inspector-scan.ap-south-2.amazonaws.com	HTTPS
Asia Pacific (Jakarta)	ap-southeast-3	inspector-scan.ap-southeast-3.amazonaws.com	HTTPS
Asia Pacific (Malaysia)	ap-southeast-5	inspector-scan.ap-southeast-5.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Melbourne)	ap-southeast-4	inspector-scan.ap-southeast-4.amazonaws.com	HTTPS
Asia Pacific (Mumbai)	ap-south-1	inspector-scan.ap-south-1.amazonaws.com	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	inspector-scan.ap-northeast-3.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	inspector-scan.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	inspector-scan.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	inspector-scan.ap-southeast-2.amazonaws.com	HTTPS
Asia Pacific (Taipei)	ap-east-2	inspector-scan.ap-east-2.amazonaws.com	HTTPS
Asia Pacific (Thailand)	ap-southeast-7	inspector-scan.ap-southeast-7.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Tokyo)	ap-northeast-1	inspector-scan.ap-northeast-1.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	inspector-scan.ca-central-1.amazonaws.com	HTTPS
Canada West (Calgary)	ca-west-1	inspector-scan.ca-west-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	inspector-scan.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	inspector-scan.eu-west-1.amazonaws.com	HTTPS
Europe (London)	eu-west-2	inspector-scan.eu-west-2.amazonaws.com	HTTPS
Europe (Milan)	eu-south-1	inspector-scan.eu-south-1.amazonaws.com	HTTPS
Europe (Paris)	eu-west-3	inspector-scan.eu-west-3.amazonaws.com	HTTPS
Europe (Spain)	eu-south-2	inspector-scan.eu-south-2.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	inspector-scan.eu-north-1.amazonaws.com	HTTPS
Europe (Zurich)	eu-central-2	inspector-scan.eu-central-2.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Israel (Tel Aviv)	il-central-1	inspector-scan.il-central-1.amazonaws.com	HTTPS
Mexico (Central)	mx-central-1	inspector-scan.mx-central-1.amazonaws.com	HTTPS
Middle East (Bahrain)	me-south-1	inspector-scan.me-south-1.amazonaws.com	HTTPS
Middle East (UAE)	me-central-1	inspector-scan.me-central-1.amazonaws.com	HTTPS
South America (São Paulo)	sa-east-1	inspector-scan.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-East)	us-gov-east-1	inspector-scan.us-gov-east-1.amazonaws.com inspector-scan-fips.us-gov-east-1.amazonaws.com	HTTPS HTTPS
AWS GovCloud (US-West)	us-gov-west-1	inspector-scan.us-gov-west-1.amazonaws.com inspector-scan-fips.us-gov-west-1.amazonaws.com	HTTPS HTTPS

Region-specific feature availability

This section describes the availability of Amazon Inspector features by AWS Region.

Agentless EC2 scanning for Amazon EC2 Regions

The following table shows the AWS Regions where agentless scanning for Amazon EC2 is currently available.

Region Name	Region code
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Malaysia)	ap-southeast-5
Asia Pacific (Thailand)	ap-southeast-7
Canada (Central)	ca-central-1

Region Name	Region code
Canada West (Calgary)	ca-west-1
Europe (Stockholm)	eu-north-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Israel (Tel Aviv)	il-central-1
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1
Mexico (Central)	mx-central-1
South America (São Paulo)	sa-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Lambda code scanning Regions

The following table shows the AWS Regions where Lambda code scanning is currently available.

Region Name	Region code
US East (N. Virginia)	us-east-1

Region Name	Region code
US West (Oregon)	us-west-2
US East (Ohio)	us-east-2
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Stockholm)	eu-north-1
Asia Pacific (Singapore)	ap-southeast-1

Important

If you try to enable Lambda code scanning with the Amazon Inspector Enable API in an AWS Region where Lambda code scanning isn't available, you receive the following access denied error:

```
An error occurred (AccessDeniedException) when calling the Enable operation:
 Lambda code scanning is not supported in unsupported-AWS Region
```

Amazon Inspector Code Security Regions

The following table shows the AWS Regions where Amazon Inspector Code Security is currently available.

Region Name	Region code
US East (N. Virginia)	us-east-1

Region Name	Region code
US West (Oregon)	us-west-2
US East (Ohio)	us-east-2
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Stockholm)	eu-north-1
Asia Pacific (Singapore)	ap-southeast-1

Machine image scanning Regions

Amazon Inspector supports machine image (AMI) scanning in the commercial AWS Regions and the AWS GovCloud (US) Regions, but not in the China Regions. The following table shows where continuous machine image scanning and on-demand machine image scans are available.

Feature	Commercial AWS Regions	AWS GovCloud (US) Regions	China Regions
Continuous machine image scanning	Available	Available	Not available
On-demand machine image scans	Available	Available	Not available

AWS GovCloud (US) Regions

For the latest information, see Amazon Inspector in the *AWS GovCloud (US) User Guide*.

Document history

The following table describes important changes in each release of the *Amazon Inspector User Guide*, beginning in November 2021. To receive notifications about documentation updates, you can subscribe to an RSS feed.

Amazon Inspector Product Updates

Change	Description	Date
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerability finding for CVE-2026-97054. This finding is resolved by upgrading the Amazon Inspector SBOM Generator to version 1.17.7 or later.	November 12, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerabi lity finding for CVE-2026-116591. This finding is resolved by upgrading the Amazon Inspector SBOM Generator to version 1.17.6 or later.	November 9, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerabi lity finding for CVE-2026-	November 6, 2026

	116591. This finding will be resolved in the next release.	
Added a supported hardened container images table for Amazon ECR scanning	Moved Chainguard, Echo, Hummingbird OS, and Wolfi to a new table of supported hardened container images for Amazon ECR scanning, and added CleanStart support for basic scanning. For more information, see Supported hardened container images.	November 6, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerabi lity findings for CVE-2026-78661 and CVE-2026-97045. These findings are resolved by upgrading the Amazon Inspector SBOM Generator to version 1.17.4 or later.	November 3, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2026-78661 and CVE-2026-97045. It was confirmed that the Amazon Inspector SBOM Generator is not impacted by these vulnerabilities. These findings will be resolved in the next release.	October 28, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerability finding for CVE-2026-111943. This finding is resolved by upgrading the Amazon Inspector SBOM Generator to version 1.17.3 or later.	October 22, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerability finding for CVE-2026-111943. This finding will be resolved in the next release.	October 21, 2026
Updated default Amazon ECR re-scan duration	Amazon Inspector now uses a default Amazon ECR re-scan duration of 7 days when you activate Amazon ECR scanning. For more information about configuring the Amazon ECR re-scan duration, see Configuring the Amazon ECR re-scan duration.	October 21, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerability finding for CVE-2026-85747. It was confirmed that the Amazon Inspector SBOM Generator is not impacted by this vulnerability. This finding is resolved by upgrading the Amazon Inspector SBOM Generator to version 1.17.2 or later.	October 19, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate a vulnerability finding for CVE-2026-85747. It was confirmed that the Amazon Inspector SBOM Generator is not impacted by this vulnerability. This finding will be resolved in the next release of the Amazon Inspector SBOM Generator. In the mean time, customers may safely ignore or suppress this vulnerability.	October 16, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2026-56866, CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78667, CVE-2026-78669, CVE-2026-94439, CVE-2026-94440, CVE-2026-94448, CVE-2026-97030, CVE-2026-97031, and CVE-2026-97032. It was confirmed that the Amazon Inspector SBOM Generator is not impacted by these vulnerabilities. These findings are resolved by upgrading the Amazon Inspector SBOM Generator to version 1.17.1 or later.	October 9, 2026
Amazon Inspector SBOM Generator 1.17.0 release	Released Amazon Inspector SBOM Generator 1.17.0. For download links and release details, see Previous versions of the Amazon Inspector SBOM Generator.	October 6, 2026

Amazon Inspector adds on-demand machine image scanning	You can now scan a single machine image (AMI) on demand without activating Amazon Inspector. Start a scan with the StartOnDemandScan API operation or from the Amazon EC2 console, track it with GetOnDemandScanStatus and ListOnDemandScans, and receive findings and scan completion notifications through Amazon EventBridge. For more information, see Running on-demand machine image scans with Amazon Inspector.	September 17, 2026
Amazon Inspector adds support for scanning machine images	Amazon Inspector can now continuously scan the machine images (AMIs) that your account owns for operating system and programming language package vulnerabilities. You control which machine images are in scope by using scan eligibility windows, a scan mode, and exclusion options for copied and backup images. For more information, see Scanning machine images with Amazon Inspector and Configuring machine image scanning.	September 8, 2026

Amazon Inspector SBOM Generator 1.16.0 release	The Amazon Inspector SBOM Generator 1.16.0 adds support for detecting Metabase installations and Mozilla Firefox, Firefox ESR, and Thunderbird installations on Linux, Windows, and macOS. This release also adds identific ation of Red Hat Hardened Images, emits an `amazon:inspector:sbom_generator:image:base_layer` property on the image metadata component during container scans, and adds the `sbomgen.get_home_dirs()` function to the Lua plugin API. For more informati on, see Previous versions of the Amazon Inspector SBOM Generator.	September 8, 2026

Updates for the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner may generate vulnerability findings for CVE-2026-56854. It was confirmed that the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner are not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner.	August 31, 2026
Updated supported CIS benchmark versions	Amazon Inspector has updated the supported benchmarks for Rocky 8/9, RHEL 8, Ubuntu 22.04/24.04, and Windows Server 2016/2019/2022/2025. For more information, see Supported operating systems and programming languages for Amazon Inspector	August 24, 2026

Updates for the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner may generate vulnerability findings for CVE-2026-61711 and CVE-2026-61712. It was confirmed that the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner are not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner.	August 19, 2026

Updates for the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner may generate vulnerability findings for CVE-2026-39821, CVE-2026-46600, CVE-2026-33818, CVE-2026-56859, CVE-2026-56858, CVE-2026-56864, CVE-2026-56853, CVE-2026-56860, CVE-2026-56865, and CVE-2026-56862. It was confirmed that the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner are not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner.	August 14, 2026

Updated guidance for Amazon EC2 scanning	Clarified guidance for Amazon EC2 scanning across several topics. We now recommend Enhanced EC2 Scanning with the Amazon Inspector VM Scanner over the legacy Amazon Inspector SSM plugin, and note that it produces more consisten t findings – especially on Windows. Documented that deep inspection custom paths are supported on Linux, Windows, and macOS when you use Enhanced EC2 Scanning, and clarified the default scan locations by operating system. Added guidance that edits to the Amazon Inspector VM Scanner `systemd` unit files are lost on package updates, and documented how to make persistent overrides . For more information, see Amazon Inspector VM Scanner and Custom paths for Amazon Inspector deep inspection.	July 28, 2026

Updates for the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner may generate vulnerability findings for CVE-2026-46600 and CVE-2026-56852. It was confirmed that the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner are not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner.	July 24, 2026

Updates for the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner may generate vulnerability findings for CVE-2026-39822 and CVE-2026-42505. It was confirmed that the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner are not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner.	July 9, 2026
Updated functionality	Amazon Inspector now supports 3-day and 7-day Amazon ECR re-scan duration settings for both image re-scan duration and image push date duration. For more information, see Configuring the Amazon ECR re-scan duration.	July 9, 2026

New content	Added content that lists all the programming languages that the Amazon Inspector VM Scanner currently supports. For more informati on, see Supported programmi ng languages: Amazon Inspector VM Scanner.	June 16, 2026
Updates for the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner may generate vulnerability findings for CVE-2026-27145, CVE-2026-42504, and CVE-2026-42507. It was confirmed that the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner are not impacted by these vulnerabi lities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin, Amazon Inspector SBOM Generator, and Amazon Inspector VM Scanner.	June 3, 2026

New feature	To perform enhanced vulnerability assessments of Amazon Elastic Compute Cloud instances, you can now use the Amazon Inspector VM Scanner instead of the Amazon Inspector SSM plugin. The Amazon Inspector VM Scanner provides several advantages, such as more granular package collection, and it uses fewer compute resources. For more informati on, see Amazon Inspector VM Scanner.	May 29, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerabi lity findings for CVE-2026-39821, CVE-2026-39827, and CVE-2026-39824. It was confirmed that Amazon Inspector is not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SBOM Generator.	May 28, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2026-29181. It was confirmed the Amazon Inspector SBOM Generator is not impacted by this vulnerability. This finding can be resolved by upgrading the Amazon Inspector SBOM Generator to version 1.12.2 to later.	April 27, 2026
Updates for the Amazon Inspector SSM plugin	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin might generate vulnerability findings for CVE-2026-32283, CVE-2026-33810, CVE-2026-32289, CVE-2026-32281, CVE-2026-32288, and CVE-2026-32282. It was confirmed that the Amazon Inspector SSM plugin is not impacted by these vulnerabilities. These findings will be resolved in the next release of the Amazon Inspector SSM plugin.	April 13, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2026-32280. This finding is resolved by upgrading to Amazon Inspector SBOM Generator version 1.12.1.	April 9, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2026-33997, CVE-2026-34040, CVE-2026-33747 and CVE-2026-33748. It was confirmed the Amazon Inspector SBOM Generator is not impacted by these vulnerabilities. These findings will be resolved in Amazon Inspector SBOM Generator version 1.12.0.	March 26, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2026-25679, CVE-2026-27142, and CVE-2026-27139. It was confirmed the Amazon Inspector SBOM Generator is not impacted by these vulnerabilities. This vulnerability can be resolved by upgrading the Amazon Inspector SBOM Generator version to 1.11.2 or later.	March 11, 2026
Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for CVE-2025-15558 . It was confirmed the Amazon Inspector SBOM Generator is not impacted by CVE-2025-15558. This vulnerability can be resolved by upgrading the Amazon Inspector SBOM Generator version to 1.11.1 or later.	March 5, 2026

Updates for the Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for `CVE-2025-68121` . It was confirmed the Amazon Inspector SBOM Generator is not impacted by CVE-2025-68121. This vulnerability can be resolved by upgrading the Amazon Inspector SBOM Generator version to 1.11.0 or later.	March 2, 2026
New managed policy	Amazon Inspector has released a new managed policy `AmazonInspector2ManagedTelemetryPolicy` that grants permissions for Amazon Inspector telemetry operations, allowing the service to collect and transmit package inventory data for vulnerability scanning. For information, see Amazon Inspector updates to AWS managed policies.	February 5, 2026

Updated policy	Amazon Inspector adds new permissions to the service-linked role named `AmazonInspector2ServiceRolePolicy`. Amazon Inspector has added a new permission that allows Amazon Inspector to describe firewall metadata for network reachability analysis. Additionally, Amazon Inspector has added additional resource scoping to allow Amazon Inspector to create, update, and start SSM associations with SSM document `AWS-ConfigureAWSPackage`. For more information, see Service-linked role permissions for Amazon Inspector.	February 3, 2026
Updates for the Amazon Inspector SSM plugin and Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin and Amazon Inspector SBOM Generator may generate vulnerability findings for `CVE-2025-61728`, `CVE-2025-61730`, and `CVE-2025-61726`. These vulnerabilities can be resolved by upgrading the Amazon Inspector SSM plugin version to 1.0.2327.0, or Amazon Inspector SBOM Generator 1.10.1 or later.	January 29, 2026

Updates for the Amazon Inspector SSM plugin and Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SSM plugin and Amazon Inspector SBOM Generator might generate vulnerability findings for `CVE-2025-61729` . It was confirmed these applications are not impacted by this CVE. We are presently working on improvements to resolve this detection. In the mean time, customers may safely ignore or suppress this vulnerability.	December 3, 2025
Updates for Amazon Inspector SBOM Generator	Amazon Inspector is aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerabi lity findings for `CVE-2025-47914` and `CVE-2025-58181` . It was confirmed the Amazon Inspector SBOM Generator is not impacted by these CVEs. We are presently working on improvements to resolve these detections. In the mean time, customers may safely ignore or suppress these vulnerabilities.	November 20, 2025

New feature	Amazon Inspector now supports AWS Organizations policies for centralized enablement and governance across organization accounts. Organization policies allow you to automatically enable Amazon Inspector scan types across your organization and prevent unauthorized modifications. For more information, see Getting started tutorial and Managing multiple accounts.	November 19, 2025
Updates for Amazon Inspector SBOM Generator	Amazon Inspector is made aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for `CVE-2025-47913` . It was confirmed the Amazon Inspector SBOM Generator is not impacted by this CVE, and an update was deployed to resolve this detection.	November 14, 2025

Updated policies	Amazon Inspector adds new permissions to the managed policies `AmazonInspector2FullAccess_v2` and `AmazonInspector2ReadOnlyAccess`. The permissions allow viewing of Amazon Inspector organizational policies and delegation configurations established through AWS Organizations policies. For more information, see AWS managed policies for Amazon Inspector.	November 14, 2025
Updates for Amazon Inspector SBOM Generator	Amazon Inspector updates Amazon Inspector SBOM Generator version. For more information, see Previous versions of the Amazon Inspector SBOM Generator.	November 11, 2025
Updated policy	Amazon Inspector adds new permissions to the service-linked role named `AmazonInspector2ServiceRolePolicy`. The permissions allow the Amazon Inspector AWS Organizations policy to enforce the enablement and disablement of Amazon Inspector. For more information, see Service-linked role permissions for Amazon Inspector.	November 10, 2025

Updates for Amazon Inspector SBOM Generator	Amazon Inspector is made aware of a scenario where the Amazon Inspector SBOM Generator might generate vulnerability findings for `CVE-2025-58188` and `CVE-2025-61725` . It was confirmed the Amazon Inspector SBOM Generator is not impacted by these CVEs, and Amazon Inspector updates Amazon Inspector SBOM Generator version. For more information, see Previous versions of the Amazon Inspector SBOM Generator.	November 4, 2025
Update for plugin	Amazon Inspector is made aware of a scenario where the Amazon Inspector SSM plugin might generate vulnerabi lity findings for `CVE-2025-58188` and `CVE-2025-61725` . It was confirmed the Amazon Inspector SSM plugin is not impacted by these CVEs, and an update was deployed to resolve this detection.	November 3, 2025

Update for plugin	Amazon Inspector is made aware of a scenario where the Amazon Inspector SSM plugin might generate a vulnerability finding for `CVE-2025-47907` . It was confirmed the Amazon Inspector SSM plugin is not impacted by these CVEs, and an update was deployed to resolve this detection.	August 8, 2025
New policy	Amazon Inspector adds a new managed policy that provides full access to Amazon Inspector and access to other related services. For more information, see AWS managed policies for Amazon Inspector.	July 3, 2025
Updated functionality	Amazon Inspector is now available in new AWS Regions. For more information, see Regions and endpoints.	July 1, 2025
Updated functionality	Amazon Inspector updates its retention period for closed findings. Amazon Inspector removes findings after 3 days if associated resources are deleted, terminated, or no longer eligble for scanning. For more information, see Understanding Amazon Inspector findings.	June 25, 2025

Updated functionality	Amazon Inspector updates its supported operating systems for Amazon EC2 scanning and Amazon ECR scanning. Amazon EC2 scanning now supports Fedora version 42 and Ubuntu version 25.04. Amazon ECR scanning now supports Alpine version 3.22, Fedora version 42, and Ubuntu version 25.04. For more information, see Supported operating systems and programming languages for Amazon Inspector.	June 18, 2025
New feature	Amazon Inspector now scans first-party application source code, third-party application dependencies, and Infrastructure as Code for vulnerabilities. For more information, see Amazon Inspector Code Security.	June 17, 2025

Update for plugin	Amazon Inspector is made aware of a scenario where the Amazon Inspector SSM plugin might generate a vulnerability finding for `CVE-2025-0913` and `CVE-2025-4673`. It was confirmed the Amazon Inspector SSM plugin is not impacted by these CVEs, and an update was deployed to resolve this detection.	June 13, 2025
New feature	Amazon Inspector can now show actively used container images and when container images were last used on a cluster. For more information, see Mapping container images to running containers.	May 16, 2025
Updates to supported operating systems	Amazon Inspector add support for BusyBox For more information, see Supported operating systems and programming languages for Amazon Inspector.	May 13, 2025

Updated policy	Amazon Inspector adds a new permission to the service-linked role named `AmazonInspector2ServiceRolePolicy`. This permission allows you to describe IP addresses and internet gateways. For more information, see AWS managed policies for Amazon Inspector.	April 29, 2025
Update for plugin	Amazon Inspector is made aware of a scenario where the Amazon Inspector SSM plugin might generate a vulnerability finding for `CVE-2025-22871`. It was confirmed the Amazon Inspector SSM plugin is not impacted by these CVEs, and an update was deployed to resolve this detection.	April 21, 2025
Update for plugin	Amazon Inspector is made aware of a scenario where the Amazon Inspector SSM plugin might generate a vulnerability finding for `CVE-2020-8911` ,`CVE-2020-8912` , and `CVE-2024-45337` . It was confirmed that Amazon Inspector is not impacted by these CVEs and an update was deployed to resolve this detection.	April 18, 2025

Updates to Amazon Inspector SBOM Generator chapter	Amazon Inspector updates Amazon Inspector SBOM Generator version. For more information, see Previous versions of the Amazon Inspector SBOM Generator.	April 16, 2025
Updates to Amazon Inspector SBOM Generator chapter	Amazon Inspector adds new topic to Amazon Inspector SBOM Generator chapter. This topic describes how the Sbomgen tracks license information in a software bill of materials. For more information, see Amazon Inspector SBOM Generator license collection.	April 16, 2025
Updates to managed policies	Amazon Inspector adds permissions that allow read-only access to Amazon ECS and Amazon EKS actions. For more information, see Service-linked role permissions for Amazon Inspector.	March 25, 2025
Updates to supported operating systems	Amazon Inspector no longer supports SUSE Linux Enterprise Server 12.5 as part of scanning for Amazon EC2 and Amazon ECR. For more information, see Supported operating systems and programming languages for Amazon Inspector.	March 21, 2025

Updates to supported operating systems	Amazon Inspector adds support for Chainguard and Wolfi to Amazon ECR scanning. For more informati on, see Supported operating systems and programmi ng languages for Amazon Inspector.	March 21, 2025
Updates to table of contents	Amazon Inspector adds chapter about tagging Amazon Inspector resources . For more information, see Tagging Amazon Inspector resources.	February 25, 2025
Updates to table of contents	Amazon Inspector adds new topic to Amazon Inspector SBOM Generator chapter. For more information, see Amazon Inspector SBOM Generator comprehensive operating system collection.	January 28, 2025
Updated functionality	Amazon Inspector adds nodejs202.x and python3.1 3 to its list of supported runtimes for Lambda standard scanning. For more information, see Supported operating systems and programming languages for Amazon Inspector.	January 24, 2025

Updated functionality	Amazon Inspector removes Oracle Linux (Oracle) 7 and SUSE Linux Enterprise Server (SLES) 15.5 from its list of supported operating systems for Amazon EC2 and Amazon ECR. For more information, see Supported operating systems and programming languages for Amazon Inspector.	December 31, 2024
Updated functionality	Amazon Inspector adds Ubuntu 24.10 to its list of supported operating systems for Amazon EC2 and Amazon ECR. For more information, see Supported operating systems and programming languages for Amazon Inspector.	December 12, 2024
Updates to table of contents	Amazon Inspector adds new topics to the Amazon Inspector SBOM Generator chapter. For more information, see Amazon Inspector SBOM Generator.	December 9, 2024
Updated functionality	Amazon Inspector updates the `amazon:inspector:sbom_generator` table to add and remove namespaces. For more information, see Using CycloneDX namespaces with Amazon Inspector.	December 9, 2024

Updated functionality	Amazon Inspector updates its CI/CD integration feature to support scan actions with CodePipeline. For more information, see Using Amazon Inspector Scan actions with CodePipeline.	November 26, 2024
Updates to table of contents	Amazon Inspector reorganizes the table of contents to include a chapter for the Amazon Inspector SBOM Generator. For more information, see Amazon Inspector SBOM Generator.	November 22, 2024
Updated functionality	Amazon Inspector removes Fedora 39 from its list of supported operating systems for Amazon EC2 and Amazon ECR. For more information, see Supported operating systems and programming languages for Amazon Inspector.	November 22, 2024
Updated functionality	Amazon Inspector removes Alpine 3.17 from its list of supported operating systems for Amazon ECR. For more information, see Supported operating systems and programming languages for Amazon Inspector.	November 22, 2024

Updated functionality	Amazon Inspector adds Sbomgen versions to Previous versions of the Amazon Inspector SBOM Generator.	November 19, 2024
Updated functionality	Amazon Inspector adds AL2 as a supported runtime. For more information, see Supported operating systems and programming languages for Amazon Inspector.	August 26, 2024
Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ServiceRolePolicy policy. The new statement allows Amazon Inspector to return function tags in AWS Lambda.	July 31, 2024
Updated functionality	Amazon Inspector releases new security controls. For more information, see Amazon Inspector controls in the *AWS Security Hub CSPM User Guide*.	July 11, 2024
Updated functionality	The Amazon Inspector SBOM Generator now scans Dockerfiles and Docker container images for misconfigurations that can introduce security vulnerabilities. For more information, see Amazon Inspector Dockerfile checks.	June 10, 2024

Updated functionality	Amazon Inspector updates its CI/CD integration feature to support CodeCatalyst actions, so you can add Amazon Inspector vulnerability scans to your CodeCatalyst workflows. For more information, see Using CodeCatalyst actions.	June 7, 2024
Updated functionality	Amazon Inspector includes an option to download a CSV file of CIS scan results. For more information, see Viewing and downloading CIS scan results in Center for Internet Security (CIS) scans for Amazon EC2 instances.	May 3, 2024
Updated functionality	Amazon Inspector updates its CI/CD integration feature to support GitHub Actions, so you can add Amazon Inspector vulnerability scans to your GitHub workflows . For more information, see Using Amazon Inspector with GitHub Actions.	April 29, 2024

Updated functionality	Amazon Inspector updates the managed policy `AmazonInspector2FullAccess`, so it creates the service-linked role `AWSServiceRoleForAmazonInspector2AgentlessScanning`. This allows users to perform agent-based scanning and agentless scanning when they enable Amazon Inspector.	April 24, 2024
Updated functionality	Amazon Inspector updates retention period for closed findings from 30 days to 7 days. For more information, see Understanding findings in Amazon Inspector.	February 12, 2024
Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ServiceRolePolicy policy. The new statement allows Amazon Inspector to start CIS scans for your instance.	January 23, 2024
New Policy	Amazon Inspector has added a new policy, AmazonInspector2ManagedCisPolicy policy, that you can use as part of in an instance profile to allow CIS scans on an instance.	January 23, 2024

New Feature	Amazon Inspector will now refresh the **ECR re-scan duration** of container images when you pull them. To change your re-scan duration based on push or pull dates see Configuring the ECR re-scan duration.	January 23, 2024
New Feature	Amazon Inspector can now run Center for Internet Security (CIS) scans on EC2 instances. For more informati on, see Amazon Inspector CIS scans.	January 23, 2024
New Feature	Amazon Inspector can now scan container images in your CI/CD pipelines. For more information, see CI/CD integration with Amazon Inspector.	November 30, 2023
New Policy	Amazon Inspector has added a new policy that allows Amazon Inspector to scan Amazon EBS snapshots from your EC2 instance for agentless scanning. For more information on the policy, see Agentless scanning.	November 27, 2023

New Feature	Amazon Inspector now supports scanning supported Linux Amazon EC2 instances without SSM agents through agentless scanning. For more information see Agentless scanning.	November 27, 2023
New supported resources	Amazon Inspector now supports scanning of MacOS Amazon EC2 instances. See Supported operating systems: Amazon EC2 scanning for supported MacOS versions.	October 5, 2023
New Regions	Amazon Inspector is now available in Asia Pacific (Jakarta), Africa (Cape Town), Asia Pacific (Osaka), and Europe (Zurich).	September 29, 2023
New feature	You can now exclude EC2 instances from Amazon Inspector scans using exclusion tags.	September 14, 2023
New feature	Amazon Inspector has added new permissions that allow Amazon Inspector to scan network configurations of Amazon EC2 instances that are part of Elastic Load Balancing target groups.	August 31, 2023

New feature	Amazon Inspector now provides vulnerability intelligence details for package vulnerability findings.	July 31, 2023
Updated functionality	Amazon Inspector has added new permissions that allow read-only users to export Software Bill of Materials (SBOM) for their resources.	June 29, 2023
New feature	You can now export SBOM for resources being scanned by Amazon Inspector.	June 13, 2023
New feature	Lambda code scanning is now generally available . New features have been added that allow you to encrypt code identified in your Lambda code scanning findings. Additionally Lambda code scanning now provides suggested remediation rewrites of your code.	June 13, 2023
Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ReadOnlyAccess policy. The new statements allows read-only users to retrieve details of Lambda code scanning status and findings for their account.	May 2, 2023

New feature	Amazon Inspector has added Vulnerability database search which allows you to check if Amazon Inspector covers a specific CVE.	May 1, 2023
Updated functionality	Amazon Inspector has added new permissions to the AmazonInspector2ServiceRolePolicy policy that allow Amazon Inspector to create AWS CloudTrail service-linked channels in your account when you activate Lambda scanning. This allows Amazon Inspector to monitor CloudTrail events in your account.	April 30, 2023
Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2FullAccess policy. The new statement allows users to retrieve details of code vulnerability findings from Lambda code scanning.	April 17, 2023

Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ServiceRolePolicy policy. The new statement allows Amazon Inspector to send information to Amazon EC2 Systems Manager about the custom paths you have defined for Amazon EC2 deep inspection.	April 17, 2023
New feature	Amazon Inspector adds additional support for Linux EC2 instances in the form of Amazon Inspector deep inspection, which scans your instances for package vulnerabilities in application programming language packages.	April 17, 2023

Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ServiceRolePolicy policy. The new statements allows Amazon Inspector to request scans of the developer code in AWS Lambda functions , and receive scan data from Amazon CodeGuru Security. Additionally Amazon Inspector has added permissions to review IAM policies. Amazon Inspector uses this information to scan Lambda functions for code vulnerabilities.	February 28, 2023
New feature	Amazon Inspector adds additional support for Lambda functions in the form of Lambda code scanning, which scan the developer code of your Lambda functions for security vulnerabilities.	February 28, 2023

Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ServiceRolePolicy policy. The new statement allows Amazon Inspector to retrieve information from CloudWatch about when an AWS Lambda function was last invoked. uses this information to focus scans on the Lambda functions in your environment that have been active in the last 90 days.	February 20, 2023
Updated functionality	Amazon Inspector added a new statement to the AmazonInspector2ServiceRolePolicy policy. The new statement allows Amazon Inspector to retrieve information about your AWS Lambda functions. Amazon Inspector uses this information to scan your Lambda functions for security vulnerabilities.	November 28, 2022
New feature	Amazon Inspector adds support for Scanning AWS Lambda functions.	November 28, 2022

Updated content	Added procedures, policy examples, and tips for exporting findings reports from Amazon Inspector to an Amazon Simple Storage Service (Amazon S3) bucket.	October 14, 2022
New content	Added information about assessing Amazon Inspector coverage of your AWS environment by using the Amazon Inspector console. The information includes descriptions of **Status** values for individual resources in your environment.	October 7, 2022
New feature	Amazon Inspector now provides additional details about how to remediate package vulnerabilities. New fields have been added to finding details. The new fields provide context about whether a fix is available through a package update. If a fix is available, the **Suggested remediation** section of a finding shows the commands that you can run to make the fix.	September 2, 2022

Updated functionality	Amazon Inspector added a new action to the AmazonInspector2ServiceRolePolicy policy. The new action allows Amazon Inspector to describe SSM association executions. Amazon Inspector also added additional resource scoping to allow Amazon Inspector to create, update, delete, and start SSM associations with `AmazonInspector2` owned SSM documents.	August 31, 2022
New feature	Amazon Inspector now supports scans for Windows instances. Amazon Inspector can now scan SSM managed instances running supported Windows operating systems. Scans of Windows hosts are performed by the Amazon Inspector SSM plugin, which is installed and invoked through new SSM associations automatically created by Amazon Inspector.	August 31, 2022
Updated functionality	Amazon Inspector updated the resource scoping of the AmazonInspector2ServiceRolePolicy policy to allow Amazon Inspector to collect software inventory in other AWS partitions.	August 12, 2022

Updated functionality	In the AmazonInspector2ServiceRolePolicy policy, Amazon Inspector restructured the resource scoping of the actions allowing Amazon Inspector to create, delete, and update SSM associations.	August 10, 2022
New feature	Amazon Inspector now supports changing your ECR automated re-scan duration setting. The Amazon ECR automated re-scan duration setting determines how long Amazon Inspector continuously monitors images pushed into repositories. When an image is older than the scan duration, Amazon Inspector will no longer scan the image and close all existing findings for it. All new accounts will automatically have their ECR automated re-scan duration set to lifetime. Previously created accounts had an ECR automated re-scan duration of 30 days, but you can now choose from 30-day, 180-day, or lifetime durations for scans.	June 25, 2022

New functionality	Amazon Inspector added a new AWS managed policy, the AmazonInspector2ReadOnlyAccess policy, to allow read-only access to Amazon Inspector functionality.	January 21, 2022
General availability	This is the initial public release of the *Amazon Inspector User Guide*.	November 29, 2021

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.