User Guide

AWS Security Agent (now part of AWS Continuum)



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

AWS Security Agent (now part of AWS Continuum): 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

Introduction **1**
What is AWS Security Agent (now part of AWS Continuum)? 1
Key capabilities 1
Benefits 3
How AWS Security Agent works 3
Overview 4
Understand the resource hierarchy and lifecycle 7
What's shared across your organization 8
What's scoped per Agent Space 8
How GitHub repositories fit into the hierarchy 10
Key differences between security capabilities 10
Understanding resource relationships 12
Pricing and billing 12
Billable capabilities 12
What counts as a task hour 13
Task hours compared to duration 13
Find the task hours for a run 14
Service quotas are not spending limits 14
Control penetration testing costs 14
Pricing FAQ 15
Get started **16**
Sign up for AWS 16
Choose your path 16
Quickstart: Run a penetration test 17
Prerequisites 18
Step 1: Set up AWS Security Agent in the AWS console 18
Step 2: Enable penetration testing and verify your domain 19
Step 3: Connect source code (optional) 20
Step 4: Create and run a penetration test 20
Step 5: Review penetration test findings 21
Quickstart: Run a code review 21
Prerequisites 18
Step 1: Set up AWS Security Agent in the AWS console 18
Step 2: Enable and configure code review 23

Step 3: Create and run a code review ... 24
Step 4: Review code review findings ... 25
Quickstart: Run a threat model ... 25
Prerequisites ... 18
Step 1: Set up AWS Security Agent in the AWS console ... 18
Step 2: Connect source code ... 27
Step 3: Create and run a threat model ... 27
Step 4: Review threats ... 28
For administrators (console) ... 29
Example configuration ... 29
Set up and create Agent Spaces ... 29
Set up AWS Security Agent ... 29
Create an Agent Space ... 35
Connect integrations ... 37
How integrations work with Agent Spaces ... 37
Connect AWS Security Agent to GitHub repositories ... 42
Remove a GitHub integration ... 49
Connect AWS Security Agent to GitLab repositories ... 51
Connect AWS Security Agent to GitLab Self-Managed ... 57
Remove a GitLab integration ... 61
Connect AWS Security Agent to GitHub Enterprise ... 62
Remove a GitHub Enterprise integration ... 67
Connect AWS Security Agent to Azure DevOps repositories ... 68
Remove an Azure DevOps integration ... 76
Find your Atlassian installation ID ... 78
Connect AWS Security Agent to Bitbucket repositories ... 79
Remove a Bitbucket integration ... 84
Connect AWS Security Agent to Bitbucket Data Center ... 85
Connect AWS Security Agent to Confluence ... 93
Remove a Confluence integration ... 97
Connect AWS Security Agent to Jira ... 98
Remove a Jira integration ... 102
Connect to privately hosted source control ... 102
Connect agent to private VPC resources ... 109
Configure capabilities ... 112
Enable penetration test ... 113

Enable pull request code review for GitHub repositories ... 119
Enable code review ... 119
Configure pull request code review triggers ... 126
Enable threat modeling ... 129
Enable users to start remediation of penetration test and code review findings ... 133
Provide agent resources from an S3 bucket ... 135
Enable an application domain for penetration testing ... 136
Managed target domains used for penetration testing ... 139
Manage security requirements ... 140
Manage security requirements ... 140
AWS managed packs ... 147
Generate security requirements from documents ... 148
Prepare documents for requirement generation ... 151
Manage users and access ... 152
Grant users access to the AWS Security Agent web application ... 152
Create the IAM roles for AWS Security Agent ... 156
Customer managed keys ... 167
Troubleshooting ... 190
Access Denied: Incorrect GitHub account type selected or incorrect organization name specified ... 190
Access Denied: Insufficient permissions to install GitHub App into organization ... 191
Agent cannot connect to endpoint during a penetration test ... 192
Endpoint validation failed during a penetration test ... 192
Penetration test failed due to an IP allowlist ... 193
Penetration test login blocked by an identity provider ... 193
Getting additional help ... 194
For users and developers (web application and IDE) ... 195
Log in to the AWS Security Agent web application ... 195
Access methods ... 195
Log in with IAM Identity Center (SSO) ... 195
Log in with admin access (IAM) ... 197
Create a penetration test ... 198
Prerequisites ... 18
Start creating a penetration test ... 198
Name your penetration test ... 199
Choose a test scope ... 199

Configure penetration test scope ... 200
Configure IAM Role ... 202
Automatic code remediation ... 203
Configure VPC resources (optional) ... 203
Configure authentication credentials (optional) ... 205
Review network scope ... 208
Attach additional resources (optional) ... 209
Set a maximum task-hours limit ... 213
Exclude risk types (optional) ... 214
Configure best-effort resource cleanup (optional) ... 214
Create the penetration test ... 215
Review findings from a penetration test ... 215
Exporting a penetration test report ... 226
Revalidate penetration test findings ... 231
Provide authentication credentials for penetration testing ... 233
Provide trusted CA certificates for a penetration test ... 243
Provide client certificates for mutual TLS ... 245
Remediate a penetration test finding ... 252
Schedule a penetration test ... 254
Prerequisites ... 18
Create a schedule ... 254
View scheduled runs ... 255
Edit or remove a schedule ... 255
Run penetration tests from your CI/CD pipeline ... 256
Before you start ... 257
How the pipeline integration scopes the test ... 258
How the pipeline integration authenticates ... 259
Scope the trust policy to your pipeline ... 259
Grant least-privilege permissions ... 262
How the integration differs by provider ... 264
Defaults the integration ships with ... 265
Validate your setup before you trust the gate ... 266
Choose how the pipeline responds ... 266
Gate on scope conflicts ... 271
Cancellation and cleanup ... 271
Protect credentials on the runner ... 273

Worked examples by provider ... 273
Related security topics ... 274
Tutorial: Gate a GitHub Actions deployment with a scoped penetration test ... 274
Prerequisites ... 18
Step 1: Create the vulnerable sample application ... 276
Step 2: Set up GitHub OIDC and an IAM role ... 277
Step 3: Add the workflow ... 278
Step 4: Validate your setup with a dry run ... 281
Step 5: Trigger the gate and read the result ... 281
Step 6: Fix the finding and confirm the gate passes ... 282
Clean up ... 283
Related topics ... 129
Tutorial: Gate a GitLab CI/CD deployment with a scoped penetration test ... 284
Prerequisites ... 18
Step 1: Create the vulnerable sample application ... 276
Step 2: Set up GitLab OIDC and an IAM role ... 287
Step 3: Store the GitLab access token ... 289
Step 4: Add the component to your pipeline ... 290
Step 5: Validate your setup with a dry run ... 292
Step 6: Trigger the gate and read the result ... 292
Step 7: Fix the finding and confirm the gate passes ... 293
Clean up ... 283
Related topics ... 129
Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test ... 295
Prerequisites ... 18
Step 1: Create the vulnerable sample application ... 276
Step 2: Set up Bitbucket OIDC and an IAM role ... 298
Step 3: Add the pipeline ... 300
Step 4: Validate your setup with a dry run ... 281
Step 5: Trigger the gate and read the result ... 281
Step 6: Fix the finding and confirm the gate passes ... 282
Clean up ... 283
Related topics ... 129
Tutorial: Gate an Azure DevOps deployment with a scoped penetration test ... 306
Prerequisites ... 18
Step 1: Create the vulnerable sample application ... 276

Step 2: Set up the IAM role for Azure DevOps OIDC ... 309
Step 3: Create the AWS service connection ... 311
Step 4: Add the pipeline task ... 311
Step 5: Validate your setup with a dry run ... 292
Step 6: Trigger the gate and read the result ... 292
Step 7: Fix the finding and confirm the gate passes ... 293
Clean up ... 283
Related topics ... 129
Review code security findings in pull requests ... 316
How code review works in pull requests ... 317
Understanding code review results ... 317
Responding to security findings ... 318
Filtering code review findings ... 318
Next steps ... 34
Create a code review ... 321
Prerequisites ... 18
Access the code reviews page ... 321
Create a code review ... 322
Run a code review ... 327
Monitor a code review run ... 328
Next steps ... 34
Review findings from a code review ... 329
Remediate code review findings ... 336
Create Jira issues for findings ... 339
Creating issues for findings ... 340
Opening a linked issue in Jira ... 340
Unlinking an issue from a finding ... 340
Run a differential code scan with S3 ... 341
How differential scans work ... 341
Prerequisites ... 18
Step 1: Generate a diff file ... 342
Step 2: Upload the diff to S3 ... 342
Step 3: Start a differential code review job ... 343
Step 4: Monitor the scan ... 344
Step 5: Review findings ... 344
Quotas and limits ... 345

Next steps ... 34
Run code security scans from your IDE ... 345
How IDE integration works ... 345
Prerequisites ... 18
Install the MCP server ... 347
First-time setup ... 348
Run a full security scan ... 349
Run a differential scan ... 349
Run a threat model review ... 350
Review findings ... 350
Apply automated fixes ... 351
Automatic scan suggestions (Kiro) ... 351
Available scan types ... 352
Environment variables ... 352
Troubleshooting ... 353
Quotas and limits ... 345
Next steps ... 34
Create a threat model ... 354
Prerequisites ... 18
How AWS Security Agent runs a threat model ... 355
Access the threat models page ... 355
Create a threat model ... 356
Run a threat model ... 358
Monitor a threat model run ... 358
Edit a threat model ... 360
Delete a threat model ... 360
Next steps ... 34
Review threats from a threat model ... 361
Create a design review ... 366
Prerequisites ... 18
Step 1: Start creating a design review ... 367
Step 2: Name your design review ... 367
Step 3: Upload design files ... 368
Step 4: Initiate the design review ... 368
Next steps ... 34
Review findings from a design review ... 369

Security and reference 374
Severity ratings and CVSS metrics 374
Severity levels 374
CVSS metrics 374
Security 375
Security Considerations 376
AWS managed policies 384
Data protection 387
Identity and access management 391
Incident response 406
Compliance validation 408
Resilience 409
Infrastructure security 410
Interface VPC endpoints 411
Configuration and vulnerability analysis 415
Cross-service confused deputy prevention 415
Security best practices 416
IAM actions and resources migration 421
Logging and monitoring 425
Logging with CloudTrail 432
Service Quotas 434
Operations Quotas 434
Configuration Quotas 435
Document history 436

Introduction

Learn what AWS Security Agent (now part of AWS Continuum) does, how it works, and how its resources fit together.

What is AWS Security Agent (now part of AWS Continuum)?

AWS Security Agent (now part of AWS Continuum) is a frontier agent that proactively secures your applications throughout the development lifecycle. It conducts automated security reviews tailored to your organizational requirements, builds threat models to identify how your applications could be attacked, and delivers context-aware penetration testing on demand. By continuously validating security from design to deployment, AWS Security Agent helps prevent vulnerabilities early across all your environments.

Security teams define organizational security requirements once in the AWS Console: approved authorization libraries, logging standards, and data access policies. AWS Security Agent automatically enforces these security requirements throughout development, evaluating architectural documents and code against your standards and providing specific guidance when it detects violations. This delivers consistent security enforcement for design and code reviews across all teams.

For deployment validation, AWS Security Agent transforms penetration testing from a periodic bottleneck into an on-demand capability. Security teams provide target URLs, authentication details, source code and application documentation. AWS Security Agent develops deep application understanding and executes sophisticated attack chains to discover and validate vulnerabilities, enabling teams to test whenever needed.

Key capabilities

AWS Security Agent (now part of AWS Continuum) provides comprehensive security capabilities spanning the entire development lifecycle.

Continuum for penetration testing

AWS Security Agent (now part of AWS Continuum) delivers on-demand penetration testing by deploying specialized AI agents to discover, validate, report and remediate security vulnerabilities through tailored multi-step attack scenarios. The agent understands your application's context by analyzing source code and documentation to identify and exploit vulnerabilities that automated

security scanning tools cannot find. It documents findings with impact analysis, reproducible attack paths, and creates pull requests with ready-to-implement code fixes, transforming periodic assessments into continuous validation that scales across all applications rather than being limited to only critical ones.

Continuum for code security review

AWS Security Agent (now part of AWS Continuum) proactively secures applications through two complementary approaches. First, you can create code reviews in the web application to perform comprehensive scans of your full source code from GitHub, GitLab, Bitbucket, or GitHub Enterprise Server repositories or S3 buckets, identifying security vulnerabilities and validating compliance with your organizational requirements across your entire codebase. Second, you can enable automated pull request analysis for connected repositories, where AWS Security Agent reviews code changes and posts security findings directly as pull request or merge request comments. In both cases, developers receive remediation guidance, and AWS Security Agent can automatically generate pull requests or merge requests with code fixes for identified vulnerabilities. This embeds security expertise across all repositories, reducing security-related delays in the development pipeline and scaling evaluation across all codebases.

Continuum for threat modeling

AWS Security Agent (now part of AWS Continuum) builds threat models of your applications to identify how they could be attacked. Provide technical design documents (scope docs) to define what the agent focuses on, source code as context for the agent to understand your existing system, or both. AWS Security Agent generates a system overview describing your application's architecture, components, trust boundaries, data flows, and security posture, along with a set of threats classified by STRIDE category (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) with severity levels and actionable recommendations. Threat models are reusable configurations that you can re-run as your code and design evolve, enabling iterative security assessment throughout the development lifecycle.

Continuum for design security review

AWS Security Agent (now part of AWS Continuum) shifts security left by providing real-time security feedback on design documents and assessing compliance with organizational security requirements before any code is written. Security teams upload documents through the web application and receive remediation guidance to prioritize findings, accelerating time-consuming manual reviews into focused analysis. By proactively embedding your security standards into

every design review, you reduce late-stage architectural rework and keep pace with multiple development teams.

Benefits

On-demand accessibility

AWS Security Agent provides immediate access to security expertise. Development teams can run design reviews, threat models, code analyses, and penetration tests on-demand whenever needed, matching the pace of modern development cycles and enabling proactive security at every stage.

Validate organizational requirements automatically

Define your organization's security requirements once in the AWS Console. AWS Security Agent automatically validates these requirements across all applications during every design and code security review, ensuring teams address your specific requirements rather than generic checklists.

Scale security expertise

AWS Security Agent scales security reviews to match development velocity by automating enforcement of organizational security requirements. Configure requirements centrally, conduct comprehensive reviews with findings analysis, and manage penetration testing scopes across your organization through the AWS Console.

Actionable fixes for confirmed vulnerabilities

AWS Security Agent validates security findings found during penetration testing through proof-based exploitation, delivering reproducible exploit paths, comprehensive impact analysis, and creates pull requests with ready-to-implement fixes in developer-friendly language. This helps teams focus on legitimate high-impact security risks without wasting time on false positives.

Multi and hybrid cloud support

AWS Security Agent operates across AWS, on-premise, hybrid, multicloud, and SaaS environments, ensuring consistent security guidance and testing regardless of your infrastructure or platform choices.

How AWS Security Agent works

AWS Security Agent operates across three interfaces to provide proactive application security throughout the development lifecycle. Security configurations are managed in the AWS

Management Console, security reviews are conducted in the Security Agent web application, and automated code and security finding remediation occur directly in code repository platforms like GitHub.

Overview

AWS Security Agent consists of three main components:

- **AWS Management Console** - Configure Agent Spaces, define security requirements, configure penetration testing, connect code repositories, and manage user access
- **Security Agent Web Application** - Execute penetration tests, create and run code reviews, create and run threat models, conduct design reviews, and review security findings within your assigned Agent Spaces
- **Code Repository Integration** - Receive automated code reviews on pull requests and penetration test remediation pull requests. Currently AWS Security Agent supports connecting to GitHub.

You work with AWS Security Agent in one of three roles, which you can identify by the interface you use:

Role	Where you work, and what you do
Administrator	Works in the **AWS Management Console** — sets up Agent Spaces, defines security requirements, configures capabilities, and manages user access.
User	Works in the **Security Agent Web Application** — runs penetration tests, code reviews, threat models, and design reviews, and reviews the resulting findings.
Developer	Works in **GitHub or an IDE** (such as Kiro or Claude Code) — receives automated security findings on pull requests and runs code scans without leaving the development environment.

We use these role names throughout to indicate who performs each task. A single person can hold more than one role.

For how AWS Security Agent organizes resources (what's shared across your organization versus scoped to each Agent Space) and how each capability is run and re-run, see the section called "Understand the resource hierarchy and lifecycle".

Console configuration

Administrators configure AWS Security Agent through the AWS Management Console.

Agent Spaces - When you create your first Agent Space in the AWS Management Console, AWS Security Agent creates the web application for your account. Each Agent Space you create represents a distinct application or project you want to secure. In the web application, users select which Agent Space to work in when conducting security assessments.

Security requirements - Define the security requirements that AWS Security Agent evaluates during design and code reviews, organized into security requirement packs. Enable AWS-managed packs based on industry standards, create custom packs for your organization's policies, or generate requirements by uploading your security documentation. Requirements apply across all Agent Spaces.

Penetration testing configuration - Configure penetration testing capabilities for each Agent Space by:

- Verifying target domains for testing through DNS or HTTP verification
- Configuring VPC access for testing private applications
- Setting up CloudWatch logging for penetration test runs
- Configuring AWS Secrets Manager or Lambda functions for test credentials
- Specifying S3 buckets for additional application context

Code review configuration - Configure code review capabilities for each Agent Space by:

- Connecting GitHub repositories or S3 buckets containing source code
- Selecting code review settings (security vulnerabilities, custom requirements, or both)
- Enabling pull request comments for automated review of code changes in GitHub
- Setting up CloudWatch logging for code review runs

Threat modeling configuration - Configure threat modeling capabilities for each Agent Space by:

- Connecting source code repositories or S3 buckets containing source code

- Adding scope docs (feature design documents) to focus the analysis on a specific feature
- Configuring CloudWatch logging for threat model runs
- Setting up a service role for AWS resource access

Integrations - Connect GitHub repositories to each Agent Space to enable key capabilities:

- Provide application context for more accurate penetration testing
- Enable code review for full repository scans and pull request analysis
- Enable automated code remediation through pull requests for code review and penetration test findings

User access - Manage how users access the Security Agent web application. If you've enabled IAM Identity Center (SSO), assign users or IAM Identity Center groups in the AWS Security Agent console to provide direct SSO access to the web application. If you're using IAM-only access, users with AWS Console access can launch the web application through the admin access link in the Console for any Agent Space.

Web application activities

Users access the Security Agent web application to conduct security assessments within their assigned Agent Spaces.

Select Agent Space - When logging into the web application, users select which Agent Space to work in. Users can only see and access Agent Spaces they've been assigned to, directly or through a group.

Continuum for penetration tests - Configure and execute penetration tests by providing target URLs, authentication details, and documentation. AWS Security Agent performs autonomous testing to discover exploitable vulnerabilities through multi-step attack scenarios.

Continuum for code reviews - Create and run code reviews that perform comprehensive static analysis across your full source code. Select GitHub repositories or S3 sources, configure scan settings, and review detailed security findings with remediation guidance. AWS Security Agent identifies security vulnerabilities and validates compliance with your organization's custom security requirements across your entire codebase.

Continuum for threat models - Create and run threat models by providing source code, technical design documents (scope docs), or both. Scope docs define what the agent focuses its analysis on;

source code provides context about your existing system. Each run produces a system overview describing your application's architecture, trust boundaries, data flows, and security posture, along with a set of threats classified by STRIDE category with severity levels and actionable recommendations.

Continuum for design reviews - Upload design documents and architecture specifications for analysis against organizational security requirements. Review findings with remediation guidance.

Review findings - Examine detailed security findings from penetration tests, code reviews, threat models, and design reviews, including impact analysis, reproducible attack paths, and remediation guidance.

Validate fixes - Re-run security assessments after implementing remediations to verify vulnerabilities have been addressed.

GitHub integration

AWS Security Agent integrates directly with GitHub to provide automated security feedback in developers' workflows.

Pull request comments - After administrators install the AWS Security Agent GitHub App and enable code review with Code review comments for an Agent Space, AWS Security Agent automatically analyzes pull requests in connected repositories. Developers receive security findings and remediation guidance directly in pull request comments, validating code changes against organizational security requirements and common vulnerabilities.

Automatic remediation - When users enable automatic code remediation for a code review, AWS Security Agent generates fixes for identified vulnerabilities and submits pull requests to the associated GitHub repositories.

Penetration test remediation - When administrators enable finding remediation in the Console, users can request automatic remediation for penetration test findings from the web application. AWS Security Agent opens a pull request to the associated GitHub repository with code fixes to address the vulnerability.

Understand the resource hierarchy and lifecycle

AWS Security Agent organizes security testing resources in a hierarchical structure that determines what's shared across your organization and what's scoped per application. Understanding this

structure helps you configure AWS Security Agent effectively and know where to find and manage different resources.

What's shared across your organization

Some resources in AWS Security Agent are configured once at the organizational level and apply across all your applications and Agent Spaces. These tenant-level resources provide consistency and reduce duplicate configuration work.

Resource	What it is	Why it's shared
Security requirements	Organizational security standards that define what AWS Security Agent validates during design and code reviews	Your security policies apply to all applications. Define them once and AWS Security Agent enforces them everywhere.
GitHub integrations	Registered GitHub organizations or user accounts authorized to connect with AWS Security Agent	Register your GitHub organization once, then connect specific repositories to any Agent Space as needed.
IAM Identity Center configurations	SSO settings that control how users access AWS Security Agent	Centralized identity management applies across all Agent Spaces in your organization.

> **Important**
>
> Changes to security requirements affect all future design reviews and code reviews across all Agent Spaces. Existing reviews are not affected.

What's scoped per Agent Space

Each Agent Space represents a distinct application or project you want to secure. Resources at the Agent Space level are scoped to that specific application, allowing different teams to work independently with their own configurations and assessments.

Resource	What it is	Why it's scoped per application
Penetration test configurations	Test configurations for specific features, API endpoints, or functiona lity within your application	Each application has unique targets, authentication methods, and scope boundaries specific to that applicati on.
Design reviews	Individual architectural security assessments of design documents	Each application has its own architecture and design documents that are assessed independently.
Threat models	Threat modeling assessments that build a system overview and identify threats from source code, design documents, or both	Each application has its own code and design, and is threat modeled independently. Threat models are reusable configurations that you can re-run as your code and design evolve.
Integrations	Source and documentation providers (GitHub, GitLab, Bitbucket, GitHub Enterprise Server, and Confluence) connected to this Agent Space	Different applications rely on different sources and documenta tion. Connecting them at the Agent Space level keeps application boundaries clear.
Code review settings	Configuration of code review capabilities including connected sources, scan settings, and PR comment enablement	Each application has its own repositories and security review needs configured independently.
Penetration test remediation settings	Configuration of which connected repositories can receive automated fix pull requests for penetration testing findings	Teams control where AWS Security Agent can submit code changes based on their application's workflow.
User and group assignments	Individual users and IAM Identity Center groups whose members can access this Agent Space	Teams only see security assessmen ts for applications they're responsib le for, keeping work organized and focused.

> **Tip**
>
> We recommend creating one Agent Space per application or project to maintain clear boundaries between teams and organize security assessments effectively.

How GitHub repositories fit into the hierarchy

GitHub repositories are integrated through a multi-step process that connects organizational resources to specific applications:

1. **Register at the tenant level** - Authorize the AWS Security Agent GitHub App for your GitHub organization or user account once
2. **Connect at the Agent Space level** - Select specific repositories to connect to each Agent Space
3. **Configure usage per repository** - Enable specific capabilities for each connected repository:
   - **Code review** - Full source code scanning and automated pull request analysis
   - **Penetration testing context** - Application understanding from source code during penetration tests
   - **Automatic code remediation** - Automated pull requests with vulnerability fixes for code review and penetration testing findings

A single repository can be connected to multiple Agent Spaces with different capabilities enabled in each one.

Key differences between security capabilities

Each security capability in AWS Security Agent follows a different workflow model based on how security teams use it.

Continuum for penetration testing: Reusable configurations with independent executions

Penetration tests use a configuration-and-run model that supports iterative security testing:

- **Create once, execute many times** - Define a configuration for a specific target (API endpoint, feature area) with scope boundaries, authentication, and test parameters

- **Independent executions** - Execute the same configuration multiple times as you improve security. Each execution generates its own findings, and first retests the findings from the most recent completed full run. For more information, see the section called “Retest of previous findings”.

This model supports continuous security validation as you develop and deploy improvements.

Continuum for design reviews: One-off assessments with cloning

Design reviews are independent assessments that don’t follow a reusable configuration model:

- **Single assessment** - Each design review analyzes uploaded documents once against your organization’s security requirements
- **Cannot re-run** - Design reviews are not reusable. You cannot re-run the same review
- **Clone for updates** - Clone an existing design review to create a new review with the original documents pre-loaded, allowing you to update documents and run a new analysis

This model supports point-in-time architectural security assessments.

Continuum for code reviews: Reusable configurations with on-demand scans and automatic PR analysis

Code reviews provide two modes of operation for securing your source code:

- **Full code reviews (web application)** - Create code review configurations that select GitHub repositories or S3 sources, then run comprehensive scans on demand. Each run performs static analysis across your full source code and generates findings with remediation guidance. You can re-run the same code review configuration as your code evolves.
- **Pull request comments (GitHub)** - Enable automated analysis for connected GitHub repositories. AWS Security Agent automatically reviews pull requests when they are marked as ready for review and posts security findings as comments directly in GitHub.

Both modes use your configured code review settings (security vulnerabilities, custom requirements, or both) and support automated code remediation through pull requests.

Continuum for threat models: Reusable configurations with on-demand runs

Threat models use a configuration-and-run model that supports iterative assessment of your architecture:

- **Create once, run many times** – Define a threat model by selecting source code as sources, uploading design documents as scope docs, or both. Run it on demand and re-run it as your code and design evolve.
- **Flexible inputs** – Run a threat model on source code only, design documents only, or both. Scope docs define what the agent focuses its analysis on; source code provides context about your existing system.
- **System overview and threats** – Each run produces a system overview describing your application's architecture, trust boundaries, data flows, and security posture, along with a set of threats classified by STRIDE category with severity, evidence, and actionable recommendations.

Understanding resource relationships

The hierarchy determines where each resource lives: the AWS Management Console for tenant-level resources and Agent Space settings, the Security Agent web application for assessments and findings, and GitHub for pull request findings and remediation. For the full breakdown of what you do in each interface and the roles involved, see the section called "How AWS Security Agent works".

Pricing and billing

With AWS Security Agent, you pay for penetration testing in *task hours*. Task hours measure the time the agent spends reasoning and acting, not the time you wait for a run to finish. This topic explains what accrues task hours so that you can predict and track what a penetration test costs.

For current rates and worked billing examples, see AWS Security Agent pricing.

Billable capabilities

The following table describes which capabilities AWS Security Agent bills for.

Capability	How AWS Security Agent charges you
Continuum for penetration testing	AWS Security Agent meters each task hour by the second of task time, except for continuous pentest runs while continuous pentesting is in preview
Continuum for code review	No charge while in preview
Continuum for design review	No charge while in preview
Continuum for threat modeling	No charge while in preview

AWS Security Agent bills penetration testing for each second of task time. It converts the total to hours on your bill, so you pay for partial hours.

While continuous pentesting is in preview, AWS doesn't charge for continuous pentest runs, however you start them. You still pay for the other AWS resources you use, such as hosting the application you test. For more information, see the section called "Run penetration tests from your CI/CD pipeline".

What counts as a task hour

A task hour measures the time of active work performed by AWS Security Agent within a task.

Task hours compared to duration

The run summary and the **All runs** table both report duration and task hours. They measure different things.

Duration is the elapsed time from when a run starts until it finishes. This represents how long you wait for results.

Task hours are the agent work inside that run. Task hours have no fixed relationship to duration.

Task hours exceed duration when the agent runs many tasks at the same time.

Find the task hours for a run

Open the penetration test in the AWS Security Agent web application. The **All runs** table reports task hours for every run alongside its duration and job type. Select a run to see the same values in its run summary.

To see which tasks accrued the task hours, select the **Logs** tab. Each task in the list reports its own task hours, and the task detail panel shows both its duration and its task hours.

To track charges across runs, use AWS Cost Explorer. To get notified when charges pass a threshold that you set, create a budget in AWS Budgets. For more information, see Analyzing your costs with AWS Cost Explorer and Managing your costs with AWS Budgets.

The `PentestExecutionDuration` metric that AWS Security Agent publishes to Amazon CloudWatch reports duration in seconds. For more information, see the section called "Logging and monitoring".

Service quotas are not spending limits

AWS Security Agent applies monthly quotas to design reviews and pull request code reviews. It also applies configuration quotas to resources such as Agent Spaces and penetration test projects. These quotas limit capacity, not spending. Reaching a monthly quota blocks additional reviews until the next month. It doesn't cap what you spend on penetration testing, and a quota increase doesn't change your rate. For more information, see the section called "Service Quotas".

Control penetration testing costs

Task hours depend on how much reasoning and acting you ask the agent to do. To reduce the task hours that a run consumes, do the following:

- **Narrow the test scope** – Target specific URL paths rather than an entire domain, and list URLs that you want to exclude. For more information, see the section called "Create a penetration test".
- **Select only the risk types you need** – Each additional risk type adds tasks to the run.
- **Stop a run that is off track** – Monitor the penetration test logs as a run progresses. If the logs show that the agent is exploring areas you don't care about, stop the run. For more information, see the section called "Review findings from a penetration test".

- **Reuse a configuration to build a baseline** – Penetration test configurations are reusable. Compare the task hours across runs of the same configuration to learn what a target typically costs.

Pricing FAQ

Why don't the task hours for my job match its duration?

A job's task hours reflect the total cumulative work done by AWS Security Agent across all concurrent tasks. Duration is the elapsed time from when a job starts until it finishes. Task hours can exceed duration when the agent runs many tasks at the same time.

Which capabilities does AWS Security Agent charge for?

Penetration testing. AWS doesn't charge for continuous pentest runs while continuous pentesting is in preview, or for code reviews, design reviews, or threat models while those capabilities are in preview. Quotas still apply to the capabilities that are in preview. For more information, see the section called "Service Quotas".

Does reaching a quota stop my charges?

No. Quotas limit how many resources you can create and how many reviews you can run. They don't limit how much you can spend. For more information, see the section called "Service Quotas".

Can I estimate what a penetration test will cost before I start it?

There is no precise estimation. Task hours depend on the breadth of the target application and the risk types you select. The agent also adapts its plan as it explores your application. If you have run the same configuration before, use the task hours from those runs as a baseline. For worked examples, see AWS Security Agent pricing.

If I stop a run before it finishes, am I charged?

Yes, the task hours that accrue until the job stops are charged. Continuous pentest runs are an exception. AWS doesn't charge for them while continuous pentesting is in preview.

Get started

Sign up, find the right starting point for your goal, and run your first assessment with a quickstart.

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

Choose your path

Find the right starting point for what you want to do with AWS Security Agent. Use the table below to match your goal to a task, then go directly to that page. If you're new to AWS Security Agent, read the section called "What is AWS Security Agent (now part of AWS Continuum)?" first for a quick overview.

I want to...	Who	Start here
Understand what AWS Security Agent does before setting it up	Everyone	the section called "What is AWS Security Agent (now part of AWS Continuum)?"
Set up the service and create an Agent Space	Admin	the section called "Set up AWS Security Agent"
Test my live or deployed application for exploitable vulnerabilities	User	the section called "Quickstart: Run a penetration test"
Scan my source code for vulnerabi lities and policy violations	User	the section called "Quickstart: Run a code review"
Model how my application could be attacked (STRIDE)	User	the section called "Quickstart: Run a threat model"
Get security feedback on a design before I write code	User	the section called "Create a design review"

I want to...	Who	Start here
Scan code without leaving my IDE (Kiro or Claude Code)	Developer	the section called “Run code security scans from your IDE”
Scan only my changed lines before merging	Developer	the section called “Run a differential code scan with S3”
Get automatic security comments on pull requests and merge requests	Admin	the section called “Enable code review”
Review findings and decide what to fix first	User	Penetration test, Code review, Threat model, Design review

> **Note**
>
> AWS Security Agent uses three roles, which you can identify by the interface you work in: **Admin** works in the AWS Management Console (setup and configuration), **User** works in the web application (running assessments and reviewing findings), and **Developer** works in GitHub or an IDE such as Kiro or Claude Code. A single person can hold more than one role. For full definitions, see the section called “How AWS Security Agent works”.

Quickstart: Run a penetration test

This quickstart walks you through running your first penetration test (pentest) with AWS Security Agent. A penetration test exercises your deployed application against a verified target domain and returns security findings, each with a severity rating and supporting evidence. It covers a publicly accessible application; to test one hosted in a private VPC, see the section called “Enable penetration test”.

> **Note**
>
> You need access to the AWS Management Console to set up AWS Security Agent and define the test scope, and access to the web application to create and run penetration tests.

Prerequisites

Before you begin, make sure you have:

- Access to the AWS Management Console with permissions to set up AWS Security Agent.
- A live target domain that hosts the application you want to test.
- The ability to add a DNS record for that domain, or a Route 53 hosted zone in the same AWS account, so you can verify ownership.
- (Optional) A source code repository (GitHub, GitLab, or Bitbucket) for your application, to give the agent more context.

Step 1: Set up AWS Security Agent in the AWS console

If you haven't already set up AWS Security Agent, complete the initial setup:

1. Navigate to AWS Security Agent in the AWS Management Console.
2. Select **Set up AWS Security Agent**.
3. Create an Agent Space. An Agent Space can be used by multiple users and should be specific for every application you want to secure. Enter a name and description for your first Agent Space. This name appears to users in the web application. The name should identify the application you want to secure.
4. Select **IAM-only access** under *User access configuration*.
   - This quickstart does not cover enabling single sign-on (SSO) with IAM Identity Center. With IAM Identity Center enabled, you can access the web application directly from the AWS Console.
   - To let users without AWS Management Console access use AWS Security Agent, enable the IAM Identity Center integration. For details, see the section called "Grant users access to the AWS Security Agent web application".
5. Choose **Set up AWS Security Agent**.

> **Note**
>
> When you choose Set up, AWS Security Agent creates your Agent Space and establishes a web application where users can run penetration tests, code reviews, threat models, and design reviews.

Step 2: Enable penetration testing and verify your domain

> **Note**
>
> In the AWS console you define the scope of what can be tested. Users then run specific penetration tests within that scope from the web application.

1. From the left sidebar, select **Agent Spaces**, then select the Agent Space you created in Step 1.
2. Select the **Penetration test** tab, then choose **Set up penetration test** to open the wizard.
3. **Configure domain** - Enter the target domain you want to test and select a verification method, **DNS TXT Record** or **HTTP Route**. The domain must be live and host the application you want to test. Choose **Next**.
4. **Verify domains** - Verify ownership of each domain in the **Target domains** table. AWS Security Agent runs tests only against verified domains. For detailed steps and the exact values to copy, see the section called “Enable an application domain for penetration testing”.
   - **Route 53 domains in the same AWS account** - Select the domain and choose **One-click verification**. AWS Security Agent creates the DNS record and completes verification for you.
   - **DNS TXT record (other DNS providers)** - In the **Target domains** table, copy the **Record name** and **Record value**, add them as a TXT record with your DNS provider, then select the domain and choose **Verify**. DNS changes can take time to propagate, so verification might not complete immediately.
   - **HTTP route** - Create a file at `.well-known/aws/securityagent-domain-verification.json` on your web server, add the verification token in the format `{"tokens": ["<token>"]}`, then select the domain and choose **Verify**.
5. (Optional) **Configure additional capabilities** - Add optional resources such as CloudWatch log groups and credentials. The **Service access** section is pre-configured: AWS Security Agent creates a service role with the required permissions unless you choose an existing IAM role.

Step 3: Connect source code (optional)

Connecting a source code provider gives AWS Security Agent context about your application and improves penetration testing coverage. This step is optional.

1. On the **Penetration test** tab, choose **Add** on the **Connect a source code provider for penetration testing** banner at the top of the page.
2. Register and connect your provider, then select the repositories to make available for penetration testing.

For the full integration flow, see Connect AWS Security Agent to GitHub repositories or the connect topic for your provider.

Step 4: Create and run a penetration test

> **Note**
>
> You create and run penetration tests in the AWS Security Agent web application. We recommend running tests against a test environment that closely mirrors production.

1. Launch the web application: select the **Web app** tab, then **Admin access**.
2. In the left sidebar, choose **Penetration tests**, then **Create a penetration test**.
3. **Penetration test details** - Set up the test scope and log source:
   a. Enter a **Pentest name**.
   b. Under **Target URLs**, select or enter a verified domain to test (for example, `https://example.com`). Only verified domains can be tested, and sub-domains are covered automatically.
   c. Under **Permissions**, select a **Service role** and, optionally, a **CloudWatch log group**. Choose **Save & Next**.
4. (Optional) **VPC Resources** - Configure a VPC if your target is on a private network that isn't publicly reachable. See the section called "Enable penetration test".
5. (Optional) **Credentials** - Add credentials so the agent can reach authenticated, non-public paths. We recommend adding these to broaden coverage and surface vulnerabilities behind a login. Choose **Test** on each credential to confirm the agent can sign in, and to discover the domains your application reaches while signed in.

6. **Network configuration** - Classify every domain the test may reach. Domains discovered while your credentials were signed in are listed as **Suggested**. Mark each one **Accessible** if the agent may reach it but must not attack it, or **Out of scope** if it must not be reached at all. Add anything the tests did not discover.
7. (Optional) **Additional configuration** - Add application context such as files, GitHub repositories, or S3 sources to improve finding quality. You can also enable automatic code remediation and set other run options here.
8. Choose **Create and execute** to start the test now, or **Create pentest** to save it and run it later.

Step 5: Review penetration test findings

1. A penetration test can take several hours to complete. Most complete within 16 hours, depending on the size and complexity of your application.
2. The run begins with a **Preflight** phase that validates the setup before testing starts: it sets up logging infrastructure, checks that your target endpoints are reachable, and prepares the testing environment. If a preflight check fails (for example, a target domain that can't be resolved), the run stops before testing begins. Open the **Preflight** tab to see which check failed, resolve it, and start a new run.
3. When the run completes, open it and review the **Run overview**, **Logs**, and **Findings** tabs.
4. On the **Findings** tab, select a finding to view its description, severity, risk type, and supporting evidence.

For more details, see the section called "Create a penetration test" and the section called "Review findings from a penetration test".

Quickstart: Run a code review

This quickstart walks you through running your first code review with AWS Security Agent. AWS Security Agent scans your source code repositories for security vulnerabilities and compliance with your organization's security requirements.

> **Note**
>
> You need access to the AWS Management Console to set up AWS Security Agent, and access to the web application to create and run code reviews.

Prerequisites

Before you begin, make sure you have:

- Access to the AWS Management Console with permissions to set up AWS Security Agent.
- A source code repository (GitHub, GitLab, or Bitbucket) or an Amazon S3 source that contains the code you want to review.

Step 1: Set up AWS Security Agent in the AWS console

If you haven't already set up AWS Security Agent, complete the initial setup:

1. Navigate to AWS Security Agent in the AWS Management Console.
2. Select **Set up AWS Security Agent**.
3. Create an Agent Space. An Agent Space can be used by multiple users and should be specific for every application you want to secure. Enter a name and description for your first Agent Space. This name appears to users in the web application. The name should identify the application you want to secure.
4. Select **IAM-only access** under *User access configuration*.
   - This quickstart does not cover enabling single sign-on (SSO) with IAM Identity Center. With IAM Identity Center enabled, you can access the web application directly from the AWS Console.
   - To let users without AWS Management Console access use AWS Security Agent, enable the IAM Identity Center integration. For details, see the section called "Grant users access to the AWS Security Agent web application".
5. Choose **Set up AWS Security Agent**.

> **Note**
>
> When you choose Set up, AWS Security Agent creates your Agent Space and establishes a web application where users can run penetration tests, code reviews, threat models, and design reviews.

Step 2: Enable and configure code review

> **Note**
>
> If you already have GitHub repositories or S3 buckets connected to your Agent Space (for example, through penetration testing setup), code review is already enabled. You can skip this step and go directly to the web application.

Open the code review setup wizard

1. From the left sidebar, select **Agent Spaces** and then select your Agent Space.
2. Select **Enable code review** from the header or the **Code review** tab.

Connect integrations, repositories, and buckets

1. **(If you don't have a GitHub integration yet)** Create a GitHub registration. If you already have one, skip to the next step.
   a. In the **Connected integrations** section, choose **Add** and then **Create new registration**.
   b. Select **GitHub** and choose **Next**.
   c. Choose **Install and authorize**, then complete installation in GitHub:
      i. Select the GitHub user or organization that owns the repository you want to review.
      ii. Select **All repositories** or **Only select repositories**.
      iii. Choose **Install & Authorize** and complete GitHub authentication.
   d. Back in the AWS Management Console, enter a **Registration name** and confirm the **Account type** matches where you installed the GitHub App.
   e. Choose **Connect** to save the registration.

   For the full GitHub integration flow, see Connect AWS Security Agent to GitHub repositories.
2. Connect GitHub repositories. In the **Connected integrations** section, choose **Add**, then select your GitHub registration. The two-step **Connect GitHub** wizard opens:
   a. On **Connect GitHub repositories**, select the repositories to include and choose **Next**.
   b. On **Manage capabilities**, toggle the following per repository:

- **Code review comments** – Let AWS Security Agent post security findings as comments on pull requests in the repository.
- **Automatic remediation** – Let users of the AWS Security Agent web application request pull requests that fix findings.

c. Choose **Save** to return to the setup wizard.

3. (Optional) Connect S3 sources. In the **S3 buckets** section, choose **Add S3 resource** and enter the S3 URI for a bucket containing source code, or choose **Browse** to pick one.
4. Select your **Code review settings**. The default, **Security requirements and vulnerability findings**, analyzes code for both compliance with the security requirements you've enabled and common vulnerabilities.
5. Choose **Next**.

Optional configurations

1. Configure optional CloudWatch log groups and service access. The default service role is pre-configured with the required permissions.
2. Choose **Save**.

Step 3: Create and run a code review

> **Note**
>
> You create and run code reviews only in the AWS Security Agent web application.

1. Select the **Web app** tab and then **Admin access** to launch the AWS Security Agent web application.
2. In the left sidebar, choose **Code reviews**.
3. Choose **Create code review**.
4. Configure the code review:
   a. Enter a **Title** that identifies the scope of this review (for example, "billing-service-security-review").
   b. Under **Sources**, select the GitHub repositories you connected to your Agent Space in Step 2, or enter the S3 sources you want to scan.

c. Select the **Service role** from your configured roles.

d. (Optional) Select **Enable automatic code remediation** to have AWS Security Agent automatically submit pull requests with fixes for all findings.

5. Choose **Create code review**.
6. On the code review detail page, choose **Start review**.

Step 4: Review code review findings

1. The code review typically takes 30–60 minutes depending on the size of your codebase.
2. The run begins with a **Preflight** phase that validates the setup before the scan starts: it confirms AWS Security Agent can pull your source code from your repository or S3 source and sets up the scanning environment. If a preflight check fails (for example, it can't access your source), the run stops before static analysis. Open the **Preflight** tab to see which check failed, resolve it, and start a new run.
3. Once complete, navigate to the completed run and select the **Findings** tab.
4. Review findings in the list-detail view:

   a. Select a finding from the left panel to view its details.

   b. Review the **Description**, **Code locations**, **Evidence**, and **Suggested fix** sections.

   c. Use **Remediate code** to generate a pull request with a fix, or review automatic remediation PRs if you enabled that option.

For more details, see the section called "Create a code review" and the section called "Review findings from a code review".

Quickstart: Run a threat model

This quickstart walks you through running your first threat model with AWS Security Agent. A threat model analyzes your application's architecture and produces a **system overview** (how AWS Security Agent understands your system) and a set of **threats** (how it could be attacked, each with a severity level, STRIDE classification, and recommendations). You can run a threat model on design documents (**scope docs**) to define the focus, source code (**sources**) to provide context about your existing system, or both.

> **Note**
>
> You need access to the AWS Management Console to set up AWS Security Agent, and access to the web application to create and run threat models.

Prerequisites

Before you begin, make sure you have:

- Access to the AWS Management Console with permissions to set up AWS Security Agent.
- At least one input for the agent to analyze: design documents to upload as scope docs, or a source code repository (GitHub, GitLab, or Bitbucket) or Amazon S3 source to provide as context.

Step 1: Set up AWS Security Agent in the AWS console

If you haven't already set up AWS Security Agent, complete the initial setup:

1. Navigate to AWS Security Agent in the AWS Management Console.
2. Select **Set up AWS Security Agent**.
3. Create an Agent Space. An Agent Space can be used by multiple users and should be specific for every application you want to secure. Enter a name and description for your first Agent Space. This name appears to users in the web application. The name should identify the application you want to secure.
4. Select **IAM-only access** under *User access configuration*.
   - This quickstart does not cover enabling single sign-on (SSO) with IAM Identity Center. With IAM Identity Center enabled, you can access the web application directly from the AWS Console.
   - To let users without AWS Management Console access use AWS Security Agent, enable the IAM Identity Center integration. For details, see the section called "Grant users access to the AWS Security Agent web application".
5. Choose **Set up AWS Security Agent**.

> **Note**
>
> When you choose Set up, AWS Security Agent creates your Agent Space and establishes a web application where users can run penetration tests, code reviews, threat models, and design reviews.

Step 2: Connect source code

> **Note**
>
> If you already have repositories or S3 buckets connected to your Agent Space (for example, through code review or penetration testing setup), you can skip this step and go directly to the web application. You can always run a threat model on uploaded scope docs without connecting any source code.

Connect repositories or S3 buckets that contain the source code you want the agent to use as context for understanding your existing system:

1. From the left sidebar, select **Agent Spaces** and then select your Agent Space.
2. Navigate to the **Integrations** section to connect your source code provider.
3. Alternatively, connect S3 buckets that contain your source code.

For the full integration flow, see Connect AWS Security Agent to GitHub repositories.

Step 3: Create and run a threat model

> **Note**
>
> You create and run threat models in the AWS Security Agent web application.

1. Launch the web application from the **Web app** tab in the AWS Management Console. Alternatively, if you have IAM Identity Center configured, log in directly.
2. In the left sidebar, choose **Threat models**.
3. Choose **Create threat model**.

4. Configure the threat model:
   a. Enter a **Title** that identifies the scope of this threat model (for example, "billing-service" or "checkout-api").
   b. Provide at least one input:
      - Under **Scope docs**, upload design documents (DOC, DOCX, JPEG, MD, PDF, PNG, or TXT), enter S3 URIs, or select Confluence pages.
      - Under **Sources**, select repositories from your connected integrations or enter S3 URIs containing source code.

      > **Tip**
      >
      > Provide **both** sources and scope docs to scope the threat model to a specific design (for example, a feature design document) while giving the agent your source code as context to understand your existing system.

   c. Select the **Service role** from your configured roles.
   d. (Optional) Select a **Log group** for CloudWatch logs.
5. Choose **Create threat model**.
6. On the threat model detail page, choose **Start run**.

Step 4: Review threats

1. The run progresses through its analysis tasks. You can monitor progress on the **Preflight** tab and view task details on the **Logs** tab.
2. Once complete, the run status changes to **Completed**.
3. Select the **Overview** tab to review the system overview and severity distribution.
4. Select the **Threats** tab to review identified threats:
   a. Select a threat from the list to view its details in the side panel.
   b. Review the **Statement**, **Severity**, **STRIDE categories**, **Recommendation**, **Evidence**, and other fields.
   c. Update the threat status to **Resolved** or **Dismissed** as you address or triage each threat.

For more details, see the section called “Create a threat model” and the section called “Review threats from a threat model”.

For administrators (console)

As an administrator, you set up AWS Security Agent in the AWS Management Console and configure Agent Spaces that users access through the AWS Security Agent web application. Each Agent Space represents a distinct environment with specific permissions and resources.

We recommend creating a unique Agent Space for each application you want to test. For example, if you have two internal projects—a billing application and a task tracking application—you should create two separate Agent Spaces.

Example configuration

Consider an administrator setting up an Agent Space to assess the security of an internal billing application. The administrator would:

- Verify the domain (such as `beta.billing.example.com`)
- Connect to GitHub and enable code review
- Configure network access by assigning an appropriate VPC, subnet, and security group for penetration testing

These pre-configured resources become the options available for penetration tests and code reviews in the Agent Space. Your guardrails apply to every assessment, and each one still has flexibility for its specific needs.

Set up and create Agent Spaces

Set up AWS Security Agent for your organization and create the Agent Spaces that scope each application's resources and assessments.

Set up AWS Security Agent

Configure AWS Security Agent for your organization by creating your first Agent Space and establishing how users will access the web application.

This initial setup enables administrators to configure security capabilities in the AWS Console and provides users with access to design review and penetration testing through the Security

Agent web application. After this one-time setup, you can create additional Agent Spaces with a simplified process.

Overview

Understanding Agent Spaces

An Agent Space is a dedicated workspace for securing a specific application or project. It contains all security reviews, optionally connected GitHub repositories, penetration test configurations, results, and findings for that application.

Agent Spaces help you organize security work by keeping each application's security assessments separate, allowing teams to focus on their specific application. We recommend creating one Agent Space per application or project to maintain clear boundaries.

Security requirements are defined at the organization level and apply across all Agent Spaces, while each Agent Space maintains its own design documents, code repositories, penetration testing configurations, penetration testing results and security findings.

What's included in an Agent Space

Each Agent Space contains:

- Optionally connected GitHub repositories associated with the application or project
- Previous design reviews for the application
- Penetration testing configurations and boundaries specific to the application
- Penetration testing test results and security findings

What you'll configure during setup

During this first-time setup, you'll make organizational decisions that apply to all Agent Spaces:

- **Access method** - Choose how users will access the Security Agent web application (IAM Identity Center SSO or IAM-only access through the AWS Console)
- **Permissions** - Configure the IAM role that the web application uses to access AWS services
- **First Agent Space** - Create your initial Agent Space with a name and description

> **Note**
>
> After completing this setup, creating additional Agent Spaces is simpler - just provide a name and description. See the section called “Create an Agent Space” for details on creating subsequent Agent Spaces.

Prerequisites

Before you begin, ensure you have:

- Permissions to create and manage AWS Security Agent resources
- Understanding of your organization’s identity management requirements
- (Optional) AWS account with permissions to create IAM roles and configure IAM Identity Center (if using SSO access)
- (Optional) Existing IAM role if you want to use a custom permissions configuration

Step 1: Create your first Agent Space

Define the basic properties of your first Agent Space that will be displayed to users in the web application.

1. On the **AWS Security Agent** console page, choose **Set up Security Agent**.
2. In the **Agent Space name** field, enter a name for your Agent Space.

   > **Note**
   >
   > The Agent Space name is displayed to users in the web application and helps identify which application or project this space represents.

3. (Optional) In the **Description** field, provide a description that assists in distinguishing the Agent Space purpose.

> **Tip**
>
> The description helps distinguish the Agent Space's purpose. We recommend describing the specific application or project this Agent Space will secure, such as "Customer portal web application" or "Payment processing microservices" or "Internal analytics platform."

Step 2: Choose your access method

Select how users will access the Security Agent web application. You choose between enabling SSO access through IAM Identity Center or providing access through the AWS Console.

> **Note**
>
> If you choose IAM-only access and later want to use IAM Identity Center, you'll need to delete your AWS Security Agent setup and complete the setup process again.

1. In the **Access method** section, select one of the following options:
   - **IAM Identity Center (SSO)** - Enable SSO access for your team through IAM Identity Center. This option is recommended for teams that need centralized user management and want users to access the web application directly without going through the AWS Console.
   - **IAM-only access** - Provide access through an admin access link in the AWS Console. This option is simpler to set up and suitable for teams that prefer console-based access or don't require SSO capabilities.
2. If you selected **IAM Identity Center (SSO)**, continue to Step 2a below.
3. If you selected **IAM-only access**, skip to Step 3.

Step 2a: Configure IAM Identity Center (SSO access only)

Complete these steps only if you selected IAM Identity Center (SSO) as your access method.

1. In the **Connect to IAM Identity Center** section, review the **AWS Region**.

> **Important**
>
> Your application must be configured in the same Region where you enable IAM Identity Center. The displayed Region is where your Agent Space will be created. IAM Identity Center must be enabled in the same Region where you create your Agent Space.

2. In the **IAM Identity Center** section, choose one of the following:
   - Choose **Create account instance** to create a new IAM Identity Center account instance
   - If an organization instance already exists in the same Region, AWS Security Agent will automatically connect to it

   > **Note**
   >
   > After you connect AWS Security Agent to IAM Identity Center, create and manage users and groups in IAM Identity Center. Assign users and groups to Agent Spaces in the AWS Security Agent console.

3. If you are connecting Active Directory or an external identity provider, review the informational alert.

   > **Important**
   >
   > If you already manage users in Active Directory or another identity source and plan to connect it to IAM Identity Center, cancel setup. Go to the IAM Identity Center console and choose the identity source before setting up AWS Security Agent. Changing identity sources later might remove existing user and group assignments.

Step 3: Configure permissions (Optional)

Configure the IAM role that your Security Agent web application uses to access AWS services, APIs, and accounts.

1. Locate the **Permissions configuration - optional** section.
2. If the section is collapsed, choose the **Permissions configuration** section to expand it.
3. Select one of the following options:

- **Create default role** - AWS Security Agent automatically creates a new IAM role with the necessary permissions for the web application
- **Use another role** - Select an existing IAM role from the available options

4. Review the role description:

> **Note**
>
> A default IAM role will be created for your web application to access other AWS services, APIs, and accounts. The role will be granted permissions required for security assessment operations.

Step 4: Complete setup

After configuring all required settings, complete the setup to create your first Agent Space and establish the Security Agent web application.

1. Review all configuration sections to ensure accuracy.
2. Choose **Set up**.
3. AWS Security Agent creates your Agent Space, establishes the web application, and configures the necessary AWS resources.

> **Note**
>
> After initial setup, you'll have the option to configure capabilities including penetration testing boundaries, security requirements, and GitHub integration.

Next steps

After setting up AWS Security Agent:

- Define security requirements for your organization
- Configure penetration testing capabilities including domain verification and VPC access for your Agent Space
- Connect GitHub repositories for code review and penetration testing context

- Assign users or groups to the Agent Space in the AWS Security Agent console if you use IAM Identity Center
- Launch the web application through the admin access link in the AWS Console if you use IAM-only access
- Create additional Agent Spaces for other applications (see the section called “Create an Agent Space”)

Create an Agent Space

Create Agent Spaces to secure applications or projects in your organization. Each Agent Space provides a workspace for security assessments, findings, and configurations specific to that application or project and can be accessed by multiple users.

This procedure assumes you have already completed the initial AWS Security Agent setup. If you haven’t set up AWS Security Agent yet, see the section called “Set up AWS Security Agent”.

Overview

Understanding Agent Spaces

An Agent Space is a dedicated workspace for securing a specific application or project. It contains all security reviews, optionally connected code repositories, penetration test configurations, results, and findings for that application.

Agent Spaces help you organize security work by keeping each application’s security assessments separate, allowing teams to focus on their specific application. We recommend creating one Agent Space per application or project to maintain clear boundaries.

For a comprehensive explanation of Agent Spaces and how they fit into your organization’s security structure, see the section called “Understand the resource hierarchy and lifecycle”.

What’s included in an Agent Space

Each Agent Space contains:

- Optionally connected code repositories associated with the application or project
- Previous design reviews for the application or project
- Penetration testing configurations and boundaries specific to the application

- Penetration testing test results and security findings

Create an Agent Space

Create a new Agent Space for an application or project you want to secure.

1. In the AWS Security Agent console, navigate to the **Agent Spaces** page.
2. Choose **Create Agent Space**.
3. In the **Agent Space name** field, enter a name for your Agent Space.

> **Note**
>
> The Agent Space name is displayed to users in the web application and helps identify which application or project this space represents.

4. (Optional) In the **Description** field, provide a description that assists in distinguishing the Agent Space purpose.

> **Tip**
>
> The description helps distinguish the Agent Space's purpose. We recommend describing the specific application or project this Agent Space will secure, such as "Customer portal web application" or "Payment processing microservices" or "Internal analytics platform."

5. Choose **Create**.

> **Note**
>
> AWS Security Agent will create your Agent Space. You can now configure capabilities and connect resources specific to this application.

Next steps

After creating your Agent Space:

- Connect source code repositories (GitHub, GitLab, Bitbucket, or GitHub Enterprise Server) for code review and penetration testing context

- Connect Confluence for documentation context in design reviews and penetration testing
- Enable code review capability for connected repositories (see the section called "Enable pull request code review for GitHub repositories")
- Configure penetration testing capabilities including domain verification
- **(If using IAM Identity Center)** Assign users or groups to this Agent Space on the **Web app** tab of the Agent Space page. (see the section called "Grant users access to the AWS Security Agent web application")
- **(If using IAM-only access)** Users with console access can launch the web application through the admin access link for this Agent Space

Connect integrations

Connect and manage source code providers (GitHub, GitLab, Azure DevOps, Bitbucket, and GitHub Enterprise Server), documentation providers (Confluence), and issue trackers (Jira), along with private network connectivity, that your Agent Spaces use for code review, penetration testing, and threat modeling.

How integrations work with Agent Spaces

AWS Security Agent connects to third-party source code and documentation providers so the agent can analyze your code and documentation during security assessments. Before you connect a specific provider, it helps to understand how integrations relate to Agent Spaces and capabilities.

Register once, reuse everywhere

Connecting a provider to AWS Security Agent involves two distinct steps that happen at different levels:

1. **Register the integration (account level).** You register a provider one time from the **Integrations** page in the AWS Security Agent Management Console. A registration represents the authorized connection to a provider account, such as a GitHub organization, a GitLab group, a Bitbucket workspace, or a Confluence site. Registrations are account-level resources.
2. **Connect resources to an Agent Space (Agent Space level).** From within an Agent Space, you connect resources from a registered integration — repositories for source code providers, or pages for Confluence — and configure how the agent uses them. This step is specific to each Agent Space.

A single registration is reused across every Agent Space in your account. If you registered a provider while setting up one Agent Space, that same registration is available when you connect resources to another Agent Space — you do not register the same provider again.

One connection, shared across capabilities

When you connect a resource to an Agent Space, that resource is shared across the agent's capabilities. You do not connect a repository separately for each capability.

- **Read access is granted by connecting the resource.** When you connect a repository, AWS Security Agent can read it for full code scans (code review), penetration testing context, and threat modeling. When you connect a Confluence page, AWS Security Agent can read it for documentation context in design reviews, threat modeling, and penetration tests.
- **Write actions are opt-in, per repository.** When you connect source code repositories to an Agent Space, you can additionally enable write actions for each repository:
  - **Code review comments** - AWS Security Agent posts review findings as comments on pull requests (or merge requests in GitLab).
  - **Code remediation** - AWS Security Agent opens pull requests (or merge requests) with fixes for discovered vulnerabilities.

    These write actions are not available for public repositories. Read-only analysis still applies.

> **Note**
>
> Confluence is a documentation provider rather than a source code provider. AWS Security Agent reads connected Confluence pages to provide context for design reviews, threat modeling, and penetration tests. Selecting a page grants read access. You can also enable create and update capabilities on a connected space to publish reports from AWS Security Agent to Confluence. The create and update capabilities are both opt-in and disabled by default.

> **Note**
>
> Jira is an issue tracker. After you connect a Jira project to an Agent Space, you can create a Jira issue for a finding so your team can track it in Jira. For each connected project, you choose whether AWS Security Agent can create issues and comment on them. For more

information about connecting a Jira project, see the section called "Connect AWS Security Agent to Jira".

Supported providers

AWS Security Agent supports the following providers:

- **Source code** - GitHub (cloud-hosted GitHub and cloud-hosted GitHub Enterprise), GitHub Enterprise Server (self-hosted), GitLab (cloud-hosted), GitLab Self-Managed (self-hosted), Azure DevOps, Bitbucket Cloud, and Bitbucket Data Center (self-hosted).
- **Documentation** - Confluence Cloud.
- **Issue tracking** – Jira Cloud.

For self-hosted providers that are not reachable over the public internet, you can route the agent's traffic through a private connection. For more information, see the section called "Connect to privately hosted source control".

AWS Security Agent IP addresses

AWS Security Agent connects to your source code repositories from a fixed set of IP addresses, one set per AWS Region. The same IP addresses are used for every supported source code provider: GitHub, GitLab, Azure DevOps, and Bitbucket. This includes self-hosted GitHub Enterprise Server, GitLab Self-Managed, and Bitbucket Data Center instances that are reachable over the public internet.

Important

If your repository provider or the network in front of it restricts access with an IP allow list, add the AWS Security Agent IP addresses for your Agent Space's AWS Region to that allow list. Wait a few minutes for the change to take effect, then register the integration. Examples include a GitHub organization IP allow list, a GitLab allowed IP range, Bitbucket workspace IP allowlisting, or a firewall in front of a self-hosted instance.

The following IP addresses are used to access your connected repositories:

- US East (N. Virginia) (us-east-1)

- 34.228.181.128
- 44.219.176.187
- 54.226.244.221

- US East (Ohio) (us-east-2)
  - 16.58.244.57
  - 18.221.155.24
  - 52.15.165.125
- US West (Oregon) (us-west-2)
  - 34.212.16.133
  - 52.89.67.212
  - 54.187.135.61
- Canada (Central) (ca-central-1)
  - 3.96.5.29
  - 3.99.39.12
  - 99.79.90.221
- Asia Pacific (Mumbai) (ap-south-1)
  - 13.126.209.199
  - 13.234.6.24
  - 35.154.102.216
- Asia Pacific (Seoul) (ap-northeast-2)
  - 3.34.151.190
  - 43.201.166.78
  - 43.202.86.139
- Asia Pacific (Singapore) (ap-southeast-1)
  - 18.139.13.125
  - 47.130.240.215
  - 54.179.238.173
- Asia Pacific (Sydney) (ap-southeast-2)
  - 13.237.95.197
  - 13.238.84.102

  - 52.64.174.242
- Asia Pacific (Tokyo) (ap-northeast-1)
  - 13.192.12.233
  - 35.74.181.230
  - 57.183.50.158
- Europe (Frankfurt) (eu-central-1)
  - 18.158.110.140
  - 52.57.96.160
  - 52.59.55.56
- Europe (Ireland) (eu-west-1)
  - 34.251.85.24
  - 52.30.157.157
  - 52.51.192.222
- Europe (London) (eu-west-2)
  - 13.42.228.66
  - 16.60.62.58
  - 35.176.240.10
- Europe (Paris) (eu-west-3)
  - 15.224.33.200
  - 15.224.52.205
  - 35.181.29.248
- Europe (Stockholm) (eu-north-1)
  - 13.49.123.102
  - 13.49.215.189
  - 16.170.2.94
- South America (São Paulo) (sa-east-1)
  - 54.94.247.213
  - 54.207.222.14
  - 54.232.201.242

Next steps

- Register a provider from the **Integrations** page. See the connect topic for your provider, such as the section called “Connect AWS Security Agent to GitHub repositories”.
- Connect resources to an Agent Space and configure capabilities. See the section called “Enable code review” and the section called “Enable penetration test”.

Connect AWS Security Agent to GitHub repositories

Connect your AWS Security Agent to GitHub repositories to enable code review, threat modeling, penetration testing, and automated remediation capabilities. AWS Security Agent supports both cloud-hosted GitHub and cloud-hosted GitHub Enterprise. Before you begin, review the section called “How integrations work with Agent Spaces” to understand how a registration is reused across Agent Spaces and shared across capabilities.

GitHub integration serves multiple purposes:

> **Note**
>
> This page covers cloud-hosted GitHub (github.com) and cloud-hosted GitHub Enterprise. For self-hosted GitHub Enterprise Server, see the section called “Connect AWS Security Agent to GitHub Enterprise”.

- **Continuum for code review** - Automatically analyze the code changes in each pull request against your organizational security requirements, and run on-demand full-repository scans
- **Continuum for threat modeling** - Provide application understanding by analyzing source code, data flows, and architecture
- **Continuum for penetration testing context** - Provide application understanding for penetration testing by analyzing source code
- **Continuum for automated remediation** - Submit pull requests with fixes for vulnerabilities discovered during security assessments

Connecting GitHub to AWS Security Agent requires authorizing the AWS Security Agent GitHub App for your GitHub organization or user account, then registering the connection in the AWS Console. When you authorize the app, you choose the permission level that the GitHub App

requests: **Read and write** or **Read-only**. For more information, see the section called "Authorize and register the AWS Security Agent GitHub App".

How GitHub integration works

Pull request analysis happens within GitHub. After you authorize the GitHub App, connect repositories, and enable code review comments in the AWS Management Console, AWS Security Agent automatically scans the changes in each new pull request (a differential scan of the changed code, which can also flag existing code that the changes make reachable) and posts findings as pull request comments.

You create and run **full code reviews** — which scan a repository's entire codebase — in the AWS Security Agent web application, not in GitHub.

Penetration testing and **threat modeling** are initiated within the AWS Security Agent web application. Users select connected repositories to provide application context, and specify target domains for penetration testing. If you enable automated remediation, users can request AWS Security Agent to fix findings by opening pull requests to connected repositories.

Prerequisites

Before you begin, ensure you have:

- A GitHub account that has the `Owner` role in the GitHub organization you want to connect, or that owns the GitHub user account you want to connect. The repository `Admin` role and the GitHub App manager role aren't sufficient. For detailed app permissions, see the section called "Authorize and register the AWS Security Agent GitHub App".
- Permissions to configure integrations for your Agent Space in the AWS Management Console
- Understanding of which repositories you want to connect for code review, threat modeling, and penetration testing

> **Important**
>
> A GitHub organization owner must authorize the AWS Security Agent GitHub App and register the connection in the AWS Management Console. You don't need to be the owner who installed the app. During registration, AWS Security Agent uses the authorizing account to list the apps installed on the organization. In GitHub, only organization owners can do that. If a member authorizes the app, registration fails with an access denied error.

This happens even when an owner already installed the app or approved the member's request. If your organization enforces Security Assertion Markup Language (SAML) single sign-on, the owner's GitHub account must also be authorized for single sign-on in the organization. For more information, see the section called "Troubleshooting".

Important

Multiple AWS accounts can connect to the same GitHub organization or user account. Each account gets an independent integration. However, only one account can enable **Code review comments** or **Code remediation** per repository. If a second account tries to enable these features for a repository already owned by another account, it receives the error "Code review comments or code remediation is already enabled for this repository by another account."

Important

Your GitHub organization might use an IP allow list. If so, add the AWS Security Agent IP addresses for your AWS Region to the IP allow list. Wait a few minutes for GitHub to apply them, then register the integration. For the IP addresses, see the section called "AWS Security Agent IP addresses".

Do not rely on the GitHub **Enable IP allow list configuration for installed GitHub Apps** setting. This setting adds IP addresses only for app-installation requests, not for the requests that AWS Security Agent makes during registration. As a result, registration fails even when those addresses appear in your IP allow list.

For more information about enabling allowed IP addresses for your GitHub organization, see Enabling allowed IP addresses in the GitHub documentation.

Authorize and register the AWS Security Agent GitHub App

Authorize the AWS Security Agent GitHub App to access your GitHub organization or user account, then register the connection in the AWS Console.

The following table describes each permission the AWS Security Agent GitHub App requests and why it is needed.

If you choose **Read-only**, the GitHub App requests only read-level access for each permission in the table. With **Read-only** permissions, the GitHub App cannot perform the write-level actions listed in the **Purpose** column.

Permission	Access level	Purpose
Contents	Read and write	Read repository source code for code review, threat modeling, and penetration testing. Write access lets AWS Security Agent open a new branch to stage remediation fixes.
Pull requests	Read and write	Read pull request details to run automated code reviews. Write access lets AWS Security Agent post review comments with findings and open remediation pull requests.
Administration	Read	Read repository settings to validate the connection and repository configuration.
Metadata	Read	Read basic repository information. GitHub requires this permission for all GitHub Apps.
Organization administration	Read	Read the list of applications installed in the target organization to verify that the AWS Security Agent GitHub App is installed.

Important

Complete all steps in this process without closing your browser or navigating away. If the registration process is interrupted, you may need to uninstall the GitHub App and start over.

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **GitHub**.
4. Choose **Next**.

5. Under **GitHub App permissions**, choose **Read and write** or **Read-only**.
6. Choose **Install and authorize**.

   You are redirected to GitHub to complete the authorization. Ensure you're logged into GitHub with an account that is an owner of the organization, or that owns the user account, you want to connect.
7. In GitHub, select the account or organization where you want to install the AWS Security Agent GitHub App.
8. Select which repositories AWS Security Agent can access:
   - **All repositories** - Grant access to all current and future repositories in the organization or user account
   - **Only select repositories** - Choose specific repositories from the dropdown. You can select multiple repositories one at a time.

   > **Note**
   >
   > You can modify repository access at any time by visiting the GitHub App settings in your GitHub organization or user account settings.
9. Choose **Install and authorize**.
10. You are redirected back to the AWS Management Console to complete the registration.
11. In the **Registration details** section, configure the following fields:
    a. **Registration name** - Enter a descriptive name for this GitHub connection. Use a name that identifies the GitHub organization or user account, such as "Acme-Corp-Org" or "Production-Repos".
    b. **Account type** - Select one of the following from the dropdown:
       - **Organization** - If you connected a GitHub organization account
       - **User** - If you connected a personal GitHub user account
    c. **Organization name** (appears only if you selected Organization) - Enter the exact name of your GitHub organization as it appears in GitHub.
12. Choose **Connect**.
13. You see a confirmation message and return to the Integrations page, where your new GitHub connection appears with its registration name. The **Access** column shows the permission level of the connection: **Read and write** or **Read-only**.

To connect additional GitHub organizations or user accounts, repeat this process by choosing **Add integration** again.

Troubleshoot GitHub integration

If you encounter issues during the GitHub integration process, use the following guidance to resolve common problems.

Unable to complete registration

If you were unable to complete the registration process (for example, your browser was closed, you navigated away from the registration page, or you encountered a session interruption), the AWS Security Agent GitHub App may be installed in your GitHub organization but not registered in the AWS Console.

Symptoms:

- When you try to authorize the GitHub App again, GitHub shows "Configure" instead of "Install"
- You cannot complete the registration in the AWS Console
- The integration does not appear in your Integrations list

Resolution:

1. Uninstall the AWS Security Agent GitHub App from your GitHub organization or user account.
2. Return to the AWS Security Agent console and start the integration process again from the beginning.

Registration fails when you enable an IP allow list

Symptoms

Registration fails with an error such as "Security token validation error. Please try again." or "Access Denied." The failure occurs even when the AWS Security Agent IP addresses already appear in your IP allow list. The GitHub **Enable IP allow list configuration for installed GitHub Apps** setting added them automatically.

Resolution

Manually add the AWS Security Agent IP addresses for your AWS Region (see the section called "AWS Security Agent IP addresses") to your organization's IP allow list. Wait a few minutes for GitHub to apply them, then register the integration again.

Multiple AWS accounts connecting to the same GitHub organization

Multiple AWS accounts can connect to the same GitHub organization or user account. When a second account connects to a GitHub organization where the app is already installed, GitHub shows only the authorization page—no re-installation is needed. The user authorizes and receives an independent integration.

However, only one account can enable **Code review comments** or **Code remediation** per repository. If a second account tries to enable these features for a repository that is already owned by another account, it receives an error: "Code review comments or code remediation is already enabled for this repository by another account."

Resolution:

- On-demand features (code review, penetration testing, threat modeling) work for all connected accounts regardless of ownership.
- To transfer automated scanning ownership, the current owner must disable code review comments for the repository. The new account can then enable it.
- All accounts can list and access repositories from the shared GitHub organization independently.

Next steps

After connecting GitHub to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories
- Choose **Enable code review** or **Setup penetration testing** to connect specific repositories to your Agent Space and configure their usage (see the section called "Enable code review" and the section called "Enable penetration test")
- Enable **Code review comments** to have AWS Security Agent analyze each pull request and post findings in GitHub (see the section called "Review code security findings in pull requests")
- Enable **Code remediation** to allow AWS Security Agent to submit pull requests with vulnerability fixes (see the section called "Enable users to start remediation of penetration test and code review findings")

- Create threat models from connected repositories in the web application (see the section called "Enable threat modeling")
- Review GitHub App permissions and repository access in your GitHub organization settings

Remove a GitHub integration

Remove a GitHub integration when you no longer need AWS Security Agent to access repositories from a specific GitHub organization or user account. You must first uninstall the AWS Security Agent GitHub App from GitHub before removing the integration in the AWS Console.

Prerequisites for removal

Before removing a GitHub integration, ensure you have:

- Checked which Agent Spaces have repositories connected from this integration
- Understood the impact: Removing this integration will break repository connections for code review, penetration testing context, and penetration test remediation across all Agent Spaces using repositories from this integration
- The `Owner` role in the GitHub organization, or ownership of the GitHub user account. You need this role to uninstall the GitHub App.

Step 1: Uninstall the AWS Security Agent GitHub App from GitHub

First, uninstall the AWS Security Agent GitHub App from your GitHub organization or user account.

For GitHub Organizations:

1. Go to github.com and open your organization page.
2. In the left sidebar, choose **Settings**.
3. Under **Code, planning, and automation**, choose **Installed GitHub Apps**.
4. Locate the **AWS Security Agent** app in the list.
5. Choose **Configure** next to the AWS Security Agent app.
6. Scroll to the bottom of the configuration page and choose **Uninstall**.
7. Confirm the uninstallation when prompted.

For GitHub User Accounts:

1. Go to github.com.
2. Choose your profile picture.
3. Choose **Settings**.
4. In the left sidebar, select **Applications**.
5. Open the **Installed GitHub Apps** tab.
6. Locate the **AWS Security Agent** app in the list.
7. Choose **Configure** next to the AWS Security Agent app.
8. Scroll to the bottom of the configuration page and choose **Uninstall**.
9. Confirm the uninstallation when prompted.

Step 2: Remove the integration from AWS Security Agent

After uninstalling the GitHub App, remove the integration registration from the AWS Console.

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Locate the GitHub integration you want to remove in the integrations list.
3. Select the integration by clicking on it.
4. Choose **Remove**.
5. Review the confirmation dialog, which warns you about the impact:

> ⚠ **Warning**
>
> Removing this integration will affect all Agent Spaces that have repositories connected from this GitHub organization or user account. This will break:
>
> - Code review functionality for connected repositories
> - Penetration testing context from connected repositories
> - Penetration test remediation capabilities for connected repositories
>
> Ensure you have uninstalled the AWS Security Agent GitHub App from GitHub before proceeding.

6. If you have not yet uninstalled the GitHub App from GitHub, you'll receive a warning. Return to Step 1 to complete the uninstallation first.
7. If you have uninstalled the GitHub App and understand the impact, choose **Confirm removal**.

8. The integration is removed from your integrations list. Any Agent Spaces with repositories from this integration no longer have access to those repositories for code review, penetration testing context, or automated remediation.

Connect AWS Security Agent to GitLab repositories

Connect your AWS Security Agent to GitLab Cloud repositories to enable code review, threat modeling, penetration testing, and automated remediation capabilities. Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces and shared across capabilities.

GitLab integration serves multiple purposes:

- **Continuum for code review** - Automatically analyze the code changes in each merge request against your organizational security requirements, and run on-demand full-repository scans
- **Continuum for threat modeling** - Provide application understanding by analyzing source code, data flows, and architecture
- **Continuum for penetration testing context** - Provide application understanding for penetration testing by analyzing source code
- **Continuum for automated remediation** - Submit merge requests with fixes for vulnerabilities discovered during security assessments

Connecting GitLab to AWS Security Agent requires providing a GitLab access token with the appropriate permissions, then registering the connection in the AWS Console.

How GitLab integration works

Merge request analysis happens within GitLab. After you provide your access token, connect projects, and enable code review comments in the AWS Management Console, AWS Security Agent automatically scans the changes in each new merge request (a differential scan of the changed code, which can also flag existing code that the changes make reachable) and posts findings as merge request comments.

> **Note**
>
> The webhooks that trigger merge request analysis are created best-effort when you register the connection. With a personal access token, AWS Security Agent creates a

webhook on each project the token can access. With a group access token, it creates a single webhook on the group.

You create and run **full code reviews** — which scan a project's entire codebase — in the AWS Security Agent web application, not in GitLab.

Penetration testing and **threat modeling** are initiated within the AWS Security Agent web application. Users specify target domains and select connected repositories to provide application context. If you enable automated remediation, users can request AWS Security Agent to fix findings by opening merge requests to connected repositories.

Note

Automated remediation is not available for public GitLab repositories to avoid disclosing vulnerabilities before they are fixed.

Prerequisites

Before you begin, ensure you have:

- A GitLab.com account with Maintainer or Owner access to the projects you want to connect
- A GitLab access token with the scopes required for your connection type:
  - **Personal** – A personal access token with all read permissions and the `api` permission.
  - **Group** – A group access token with at least the Owner role, all read permissions, and the `api` permission.
- Permissions to configure integrations in the AWS Security Agent Management Console

Important

If your GitLab group or project restricts access with an allowed IP range, or a network firewall sits in front of GitLab, add the AWS Security Agent IP addresses for your AWS Region before you register the integration. For the IP addresses, see the section called "AWS Security Agent IP addresses".

Note

Keep the following in mind when you create your GitLab access token:

- **Scope** – The `read_api` scope is sufficient only for read-only operations, such as listing repositories. The full `api` scope is required for any operation that writes to GitLab, including posting code review comments on merge requests and automated remediation (creating merge requests). The `read_repository` and `write_repository` scopes grant Git-over-HTTPS access only. They do **not** grant REST API access, and a token with only those scopes returns `401 Unauthorized` for API operations.
- **Expiration** – Set the token expiration to a maximum of 365 days after the current date.
- **Reuse across accounts** – GitLab Personal Access Tokens can be used across multiple AWS accounts.

Register a GitLab connection

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **GitLab**, then choose **Next**.
4. Under **Choose an account type**, select one of the following:
   - **Personal** - Connect your individual GitLab user account.
   - **Group** - Connect a GitLab group that contains multiple projects. Enter your **Group ID**.
5. In the **Access token** field, paste your GitLab access token.
6. In the **Registration name** field, enter a descriptive name for this connection, such as `Engineering-Team-GitLab`. Valid characters are letters, numbers, periods, underscores, and hyphens.
7. Choose **Connect**.

   You return to the **Integrations** page, where the new connection appears with its registration name.

Troubleshoot GitLab integration

If you encounter issues during the GitLab integration process, use the following guidance to resolve common problems.

Invalid or expired token

Symptoms

- Integration fails to connect during registration
- Registration succeeds, but a later operation (such as listing repositories, `ListResourcesFromIntegration`, code review, or remediation) fails
- A previously working integration stops functioning
- An error that includes `GITLAB authentication failed: Invalid or expired token`, or the GitLab API response `{"message":"401 Unauthorized"}`

Resolution

1. In GitLab, navigate to your user settings and select **Access Tokens**.
2. Verify the token has not expired. GitLab returns `401 Unauthorized` for expired, revoked, or malformed tokens.
3. Verify the token has the scopes required for the operations you use (see the section called "Missing or insufficient token scope").
4. If the token has expired or was revoked, create a new token and update the integration in the AWS Console.

> **Tip**
>
> To confirm whether a token is valid independently of AWS Security Agent, call the GitLab REST API directly. A valid token returns HTTP 200 with a JSON list; an invalid, expired, or wrong-scope token returns `{"message":"401 Unauthorized"}`.
>
> ```
> curl --header "PRIVATE-TOKEN: <your-token>" https://gitlab.example.com/api/v4/
> projects
> ```

Missing or insufficient token scope

A token can be valid and unexpired but still return `401 Unauthorized` if it lacks the scopes required for the operation. This most commonly happens when a repository-only token (created for Git operations) is reused for API operations.

Symptoms

- Registration or `ListResourcesFromIntegration` returns `401 Unauthorized` even though the token is current
- Reading repositories works, but posting code review comments or creating merge requests fails

Resolution

Select scopes based on the capabilities you need:

Capability	Minimum scope	Notes
List and read repositories (threat modeling, penetration testing context)	`read_api`	Read-only access to the REST API.
Post code review comments and automated remediation (create merge requests, push fix branches)	`api`	Full read/write API access. The `api` scope is a superset that also covers Git read/write over HTTPS.

> **Important**
>
> The `read_repository` and `write_repository` scopes grant Git-over-HTTPS access only. They do **not** grant REST API access, so a token with only those scopes returns `401 Unauthorized` for API operations such as listing resources. Use `read_api` or `api` for API access.

Group access token requires a paid GitLab.com tier

On GitLab.com, group access tokens are available only with a Premium or Ultimate subscription, and not on a trial. The integration supports the **Personal** and **Group** account types only. It does

not support project access tokens, which on GitLab.com also require a Premium or Ultimate subscription.

Symptoms

- You cannot create a group access token in GitLab on a free GitLab.com namespace.
- Registration fails when you select the **Group** account type with a free GitLab.com namespace.

Resolution

- On a free GitLab.com namespace, select the **Personal** account type and use a personal access token, which is available on all tiers.
- To use the **Group** account type, upgrade the group's namespace to Premium or Ultimate. Then create a group access token with at least the Owner role, all read permissions, and the `api` permission.
- On GitLab self-managed, group access token availability depends on your instance's license tier.

Rate limiting

Symptoms

- Intermittent failures during code review
- Delayed merge request analysis

Resolution

- GitLab applies rate limits to API requests. If you have a large number of repositories or frequent merge requests, some requests may be throttled.
- Wait for the rate limit window to reset, or contact GitLab support to increase your rate limits.

Next steps

After connecting GitLab to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories.

- Choose **Enable code review** or **Setup penetration testing** to connect specific projects to your Agent Space and configure their usage (see the section called "Enable code review" and the section called "Enable penetration test").
- Enable **Code review comments** to have AWS Security Agent analyze each merge request and post findings in GitLab (see the section called "Review code security findings in pull requests").
- Enable **Code remediation** to allow AWS Security Agent to submit merge requests with vulnerability fixes (see the section called "Enable users to start remediation of penetration test and code review findings").
- Create threat models from connected projects in the web application (see the section called "Enable threat modeling").
- For privately hosted GitLab instances, see the section called "Connect AWS Security Agent to GitLab Self-Managed".

Connect AWS Security Agent to GitLab Self-Managed

Connect your AWS Security Agent to a GitLab Self-Managed instance to enable code review, threat modeling, penetration testing, and automated remediation capabilities for repositories hosted on your own infrastructure.

GitLab Self-Managed integration works the same as GitLab Cloud (see the section called "Connect AWS Security Agent to GitLab repositories") with additional configuration for network connectivity to your private instance. Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces and shared across capabilities.

> **Note**
>
> GitLab Self-Managed is registered through the **GitLab** integration, not a separate integration type. In the registration flow you select **Use GitLab self-hosted endpoint** and provide your instance URL. The cloud-hosted GitLab flow is described in the section called "Connect AWS Security Agent to GitLab repositories".

Prerequisites

Before you begin, ensure you have:

- A GitLab Self-Managed instance that is either:
  - Publicly accessible over the internet, OR
  - Accessible via a private connection (see the section called "Connect to privately hosted source control")
- A GitLab access token with the scopes required for your connection type:
  - **Personal** – A personal access token with all read permissions and the `api` permission.
  - **Group** – A group access token with at least the Owner role, all read permissions, and the `api` permission.
- Maintainer or Owner access to the projects you want to connect
- Your GitLab instance must serve HTTPS traffic with a minimum TLS version of 1.2

> **Important**
>
> If your GitLab Self-Managed instance is reachable over the public internet and restricts inbound access with an IP allow list or firewall, add the AWS Security Agent IP addresses for your AWS Region before you register the integration. For the IP addresses, see the section called "AWS Security Agent IP addresses". Instances reached through a private connection do not need this step. That traffic arrives over the private network path, not from these public IP addresses.

> **Note**
>
> If your GitLab Self-Managed instance uses TLS certificates issued by a private certificate authority, you can provide the PEM-encoded public key of the certificate when creating a private connection. This allows AWS Security Agent to trust the TLS connection to your instance.

Register a GitLab Self-Managed connection

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **GitLab**, then choose **Next**.

4. Under **Choose an account type**, select **Personal** or **Group**. If you select **Group**, enter your **Group ID**.
5. Select **Use GitLab self-hosted endpoint**.
6. In the **GitLab self-hosted endpoint URL** field, enter the URL of your instance, for example `https://gitlab.example.com`.
7. If your instance is not publicly accessible, select **Connect to endpoint using a private connection**, then choose an existing private connection or create a new one. See the section called “Connect to privately hosted source control”.
8. In the **Access token** field, paste your GitLab access token.
9. In the **Registration name** field, enter a descriptive name for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
10. Choose **Connect**.

    You return to the **Integrations** page, where the new connection appears with its registration name.

Private connectivity

If your GitLab Self-Managed instance is not publicly accessible, you must create a private connection before registering the integration. See the section called “Connect to privately hosted source control” for detailed instructions.

> **Important**
>
> Service-managed private connections require the GitLab Self-Managed instance to be running in the **same AWS account** where the Agent Space is created. For cross-account access, use a self-managed private connection where you provide your own VPC Lattice resource configuration.

Troubleshoot GitLab Self-Managed integration

In addition to the troubleshooting steps in the section called “Connect AWS Security Agent to GitLab repositories”, the following issues are specific to self-managed instances:

401 Unauthorized through a private connection

Symptoms

- An operation such as `ListResourcesFromIntegration` returns `GITLAB authentication failed: Invalid or expired token` with the body `{"message":"401 Unauthorized"}`

Resolution

- A GitLab `401` response means the request **reached** your GitLab instance and GitLab rejected the credentials. This confirms that DNS resolution, TLS, and the private connection (VPC Lattice resource gateway and resource configuration) are all working. The problem is the access token, not connectivity, so do not troubleshoot the private connection for this error.
- Resolve it as a token issue. See the **Invalid or expired token**, **Missing or insufficient token scope**, and **Group access token requires a paid GitLab.com tier** sections in the section called "Connect AWS Security Agent to GitLab repositories".
- By contrast, a connection **timeout** or a **TLS error** (rather than a GitLab JSON response) indicates a network or certificate problem. See **Instance unreachable** and **TLS certificate errors** below.

Instance unreachable

Symptoms

- Connection fails with timeout or network error
- Integration was previously working but stops functioning

Resolution

- Verify your GitLab instance is running and accessible
- If using a private connection, verify the VPC Lattice resource gateway is healthy and the ENIs have network connectivity to your instance
- Verify security groups allow traffic on the configured port
- Verify TLS certificate is valid and not expired

TLS certificate errors

Symptoms

- Connection fails with SSL/TLS error

Resolution

- Verify your instance serves HTTPS with TLS 1.2 or higher
- If using a private certificate authority, ensure the PEM-encoded public key was provided during private connection setup
- Verify the certificate is not expired

Next steps

After connecting GitLab Self-Managed to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories
- Choose **Enable code review** or **Setup penetration testing** to connect specific projects (see the section called "Enable code review" and the section called "Enable penetration test")
- Enable **Code review comments** to have AWS Security Agent analyze each merge request and post findings in GitLab (see the section called "Review code security findings in pull requests")
- Enable **Code remediation** for merge request-based fixes (see the section called "Enable users to start remediation of penetration test and code review findings")
- Create threat models from connected projects in the web application (see the section called "Enable threat modeling")

Remove a GitLab integration

Remove a GitLab integration when you no longer need AWS Security Agent to access repositories from a specific GitLab account or group.

Prerequisites for removal

Before removing a GitLab integration, ensure you have:

- Checked which Agent Spaces have repositories connected from this integration

- Understood the impact: Removing this integration will break repository connections for code review, penetration testing context, threat modeling, and automated remediation across all Agent Spaces using repositories from this integration

Remove the integration from AWS Security Agent

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Locate the GitLab integration you want to remove.
3. Select the integration by clicking on it.
4. Choose **Remove**.
5. Review the confirmation dialog and choose **Confirm removal**.

> **Note**
>
> After removing the integration, you may also want to revoke the Personal Access Token in GitLab to prevent further access. Navigate to your GitLab user settings and delete or revoke the token.

The same process applies to GitLab Self-Managed integrations.

Connect AWS Security Agent to GitHub Enterprise

Connect AWS Security Agent to GitHub Enterprise (GHE) to enable code review, threat modeling, penetration testing, and automated remediation for your repositories.

GitHub Enterprise integration provides the same capabilities as cloud-hosted GitHub (see Connect AWS Security Agent to GitHub repositories). Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces and shared across capabilities.

> **Note**
>
> GitHub Enterprise is registered through the **GitHub** integration, not a separate integration type. In the registration flow, you choose **GitHub Enterprise** as the instance type. This instance type covers both a GitHub Enterprise Server instance that you operate and a GitHub Enterprise Cloud with data residency (a `.ghe.com` host). For the cloud-

hosted GitHub.com flow, see the section called "Connect AWS Security Agent to GitHub repositories".

How GitHub Enterprise integration works

Pull request analysis happens within GitHub Enterprise. After you authorize the connection, connect repositories, and enable code review comments in the AWS Management Console, AWS Security Agent automatically scans the changes in each new pull request (a differential scan of the changed code, which can also flag existing code that the changes make reachable) and posts findings as pull request comments.

You create and run **full code reviews** — which scan a repository's entire codebase — in the AWS Security Agent web application, not in GitHub Enterprise.

Penetration testing and **threat modeling** are initiated within the AWS Security Agent web application. Users specify target domains and select connected repositories to provide application context. If you enable automated remediation, users can request AWS Security Agent to fix findings by opening pull requests to connected repositories.

GitHub Enterprise Cloud with data residency (.ghe.com)

GitHub Enterprise Cloud with data residency serves your organization from a dedicated, GitHub-operated tenant on a `.ghe.com` host rather than from `github.com`. Connect it through the same **GitHub Enterprise** instance type you use for GitHub Enterprise Server, following the steps below. A data residency tenant installs the AWS Security Agent GitHub App under an organization, so you choose the **Organization** account type during registration.

Important

AWS Security Agent processes your repository content in the Region of the AWS Security Agent instance you connect from. If that Region differs from your tenant's data residency Region, AWS Security Agent processes your content outside that Region. See the section called "Cross-Region data processing".

Prerequisites

Before you begin, ensure you have:

- Permissions to configure integrations in the AWS Security Agent Management Console

For a GitHub Enterprise Server instance:

- An instance that is either publicly accessible over the internet, or reachable through a private connection (see the section called “Connect to privately hosted source control”)
- Site administrator or organization administrator access on the instance
- The instance must serve HTTPS traffic with a minimum TLS version of 1.2

For GitHub Enterprise Cloud with data residency:

- An organization on your `.ghe.com` tenant, and permission to install a GitHub App in it

> **Note**
>
> GitHub Enterprise integrations can be used across multiple AWS accounts.

Register a GitHub Enterprise connection

Registering a GitHub Enterprise connection uses an OAuth-based authorization flow.

> **Important**
>
> Complete all steps in this process without closing your browser or navigating away. If the registration process is interrupted, you may need to restart from the beginning.

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **GitHub**, then choose **Next**.
4. Under **Instance type**, select **GitHub Enterprise**.
5. In the **GitHub Enterprise URL** field, enter the HTTPS URL of your instance. For a GitHub Enterprise Server instance, enter its host, for example `https://github.example.com`. For GitHub Enterprise Cloud with data residency, enter your `.ghe.com` host, for example `https://acme.ghe.com`. Do not include an `api.` prefix or additional subdomain levels.

6. Under **GitHub App permissions**, choose **Read and write** or **Read-only**. With **Read-only**, AWS Security Agent creates a GitHub App on your instance that requests only read permissions, and **Code review comments** and **Code remediation** are unavailable for the connection. For more information, see the section called “Authorize and register the AWS Security Agent GitHub App”.
7. (GitHub Enterprise Server only) If your instance is not publicly accessible, select **Connect to endpoint using a private connection**, then choose an existing private connection or create a new one. See the section called “Connect to privately hosted source control”.
8. In the **Register details** section, configure the following fields:
   a. **Registration name** - Enter a descriptive name for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
   b. **GitHub account type** - Select **Organization** or **User**. For a GitHub Enterprise Cloud with data residency, you must select **Organization**; **User** accounts are not supported.
   c. **Organization name** (appears only if you selected Organization) - Enter the exact name of your GitHub Enterprise organization. Names are case sensitive.
9. Choose **Connect**.

> **Note**
>
> AWS Security Agent redirects you away from the console to complete authorization with your GitHub Enterprise instance. After authorization completes, you return to the console and the new connection appears on the **Integrations** page.

Private connectivity

If your GitHub Enterprise Server instance is not publicly accessible, you must create a private connection before registering the integration. See the section called “Connect to privately hosted source control” for detailed instructions.

> **Important**
>
> Service-managed private connections require the GHE instance to be running in the **same AWS account** where the Agent Space is created. For cross-account access, use a self-managed private connection.

Note

If your GHE instance uses TLS certificates issued by a private certificate authority, provide the PEM-encoded public key of the certificate when creating the private connection.

Troubleshoot GitHub Enterprise integration

If you encounter issues connecting AWS Security Agent to GitHub Enterprise, use the following guidance to diagnose and resolve common problems.

OAuth redirect failure

Symptoms

- Browser redirects fail during the authorization flow
- Error page displayed after authorizing on GHE

Resolution

- Verify your GHE instance is accessible from your browser
- Ensure the OAuth callback URL is correctly configured
- Restart the integration process from the beginning

Instance unreachable

Symptoms

- Connection fails with timeout or network error

Resolution

- Verify your GHE instance is running and accessible
- If using a private connection, verify VPC Lattice connectivity
- Verify security groups allow traffic on the configured port
- Verify TLS certificate is valid (TLS 1.2 minimum)

Next steps

After connecting GitHub Enterprise to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories
- Choose **Enable code review** or **Setup penetration testing** to connect specific repositories (see the section called "Enable code review" and the section called "Enable penetration test")
- Enable **Code review comments** to have AWS Security Agent analyze each pull request and post findings in GitHub Enterprise (see the section called "Review code security findings in pull requests")
- Enable **Code remediation** to allow AWS Security Agent to submit pull requests with vulnerability fixes (see the section called "Enable users to start remediation of penetration test and code review findings")
- Create threat models from connected repositories in the web application (see the section called "Enable threat modeling")

Remove a GitHub Enterprise integration

Remove a GitHub Enterprise integration when you no longer need AWS Security Agent to access repositories from a specific GHE instance.

Prerequisites for removal

Before removing a GHE integration:

- Check which Agent Spaces have repositories connected from this integration
- Understand the impact: Removing will break code review, penetration testing context, threat modeling, and automated remediation for all connected repositories

Remove the integration

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Locate the GitHub Enterprise integration you want to remove.
3. Select the integration.
4. Choose **Remove**.

5. Review the confirmation dialog and choose **Confirm removal**.

> **Note**
>
> After removal, you may also want to revoke the OAuth application on your GHE instance. Navigate to your GHE organization settings and remove the authorized application.

Connect AWS Security Agent to Azure DevOps repositories

Connect your AWS Security Agent to Azure DevOps repositories to enable code review, threat modeling, penetration testing, and automated remediation capabilities. Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces and shared across capabilities.

Azure DevOps integration serves multiple purposes:

- **Continuum for code review** – Automatically analyze the code changes in each pull request against your organizational security requirements, and run on-demand full-repository scans
- **Continuum for threat modeling** – Provide application understanding by analyzing source code, data flows, and architecture
- **Continuum for penetration testing context** – Provide application understanding for penetration testing
- **Continuum for automated remediation** – Submit pull requests with fixes for vulnerabilities discovered during security assessments

AWS Security Agent authenticates to Azure DevOps as a Microsoft Entra service principal (a non-human identity managed in your Microsoft Entra tenant). You authorize the AWS Security Agent application, which appears as **AWS Continuum for Azure DevOps** in Microsoft Entra and Azure DevOps, in your Microsoft Entra tenant, add its service principal to your Azure DevOps organization, and grant it the Azure DevOps permissions the agent needs.

> **Note**
>
> AWS Security Agent supports Git repositories in Azure DevOps only. Team Foundation Version Control (TFVC) repositories are not supported.

How Azure DevOps integration works

Pull request analysis happens within Azure DevOps. After you register the integration, connect repositories, and enable code review comments in the AWS Management Console, AWS Security Agent installs a service hook (webhook) that notifies it of new pull requests. The agent scans the changes in each pull request (a differential scan of the changed code, which can also flag existing code that the changes make reachable) and posts findings as pull request comments.

You create and run **full code reviews**—which scan a repository's entire codebase—in the AWS Security Agent web application, not in Azure DevOps.

AWS Security Agent installs one set of service hooks per Azure DevOps project. An integration supports code review comments and remediation on repositories in at most 50 projects. For all quotas, see the section called "Service Quotas".

Penetration testing and **threat modeling** are initiated within the AWS Security Agent web application. Users specify target domains and select connected repositories to provide application context.

> **Note**
>
> Automated remediation is not available for public Azure DevOps repositories to avoid disclosing vulnerabilities before they are fixed.

Prerequisites

Before you begin, ensure you have:

- An Azure DevOps organization with Microsoft Entra ID enabled. A service principal can only be added to an organization that is connected to Microsoft Entra ID.
- **Project Administrator** or **Project Collection Administrator** permission in Azure DevOps, to add the service principal and grant its permissions.
- The Azure CLI with the Azure DevOps extension installed (`az extension add --name azure-devops`), signed in as an administrator. The service hook (webhook) permission in the following procedure cannot be granted from the Azure DevOps UI and must be set with the CLI.
- Permissions to configure integrations in the AWS Security Agent Management Console.
- If your Microsoft Entra policies require admin consent, a Microsoft Entra administrator who can approve the AWS Continuum for Azure DevOps application in your tenant.

Important

If your Azure DevOps organization or its network restricts access with an IP allow list, add the AWS Security Agent IP addresses for your AWS Region. Do this before you register the integration. For the IP addresses, see the section called “AWS Security Agent IP addresses”.

Register an Azure DevOps connection

The following table describes each permission the AWS Continuum for Azure DevOps application requests and why it is needed.

Permission	Purpose
Code (read and write) (`vso.code_write `)	Read repository source code and pull request diffs, post review comments with findings, and push a branch with remediation fixes to open a pull request.
Project and team (read) (`vso.project `)	Read the projects and repositories in your organization so that you can select which repositories to connect.

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **Azure DevOps**, then choose **Next**.
4. Choose **Authorize**.

   The console redirects you to Microsoft, where you grant admin consent for the AWS Continuum for Azure DevOps application in your Microsoft Entra tenant. After you approve the consent request, you return to the console.
5. In the **Organization** field, enter your Azure DevOps organization name, for example `ado-organization`.
6. In the **Registration name** field, enter a descriptive name for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
7. Choose **Connect**.

You return to the **Integrations** page, where the new connection appears with its registration name.

> **Note**
>
> Registering the connection authorizes AWS Security Agent in your Microsoft Entra tenant. Before the agent can read repositories or install pull request webhooks, you must also add its service principal to your organization and grant it permissions, as described in the following section.

Grant AWS Security Agent access in Azure DevOps

After you register the connection, complete the following steps in Azure DevOps so the service principal can access your repositories and install pull request webhooks.

Add the service principal to your organization

A service principal is added to your Azure DevOps organization in the same way as a user. You add it to the organization only once. To analyze repositories in more projects later, assign the same service principal to those projects—you do not repeat the authorization or the organization-level Service Hooks grant, which already apply across the organization.

1. In Azure DevOps, go to **Organization settings**, then **Users**, then choose **Add users**.
2. In the **Users** field, search for **AWS Continuum for Azure DevOps**, or paste its **Application (client) ID**, and select it.
3. Set the **Access level** to **Basic**, add it to each project whose repositories the agent will analyze, then choose **Add**. This grants the service principal the repository read and write access the agent needs for those projects.

> **Note**
>
> Adding a service principal to an organization requires the organization to be connected to Microsoft Entra ID. For more information, see Prerequisites.

Grant AWS Continuum for Azure DevOps Service Hooks permissions

To install the pull request webhook, the service principal needs **View and Edit Subscriptions** on Service Hooks. This permission cannot be granted from the Azure DevOps UI and must be set with the Azure CLI.

Grant it at the organization level. The Service Hooks security namespace is hierarchical, so an organization-level grant is inherited by every current and future project, and you do not need to repeat it as projects are added.

1. Get the service principal descriptor:

```
az rest --method get \
  --resource 499b84ac-1321-427f-aa17-267ca6975798 \
  --url "https://vssps.dev.azure.com/ORG_NAME/_apis/graph/serviceprincipals?api-
version=7.1-preview.1" \
  --query "value[?contains(displayName, 'AWS Continuum for Azure DevOps')].
{descriptor:descriptor, displayName:displayName}" \
  -o table
```

   `499b84ac-1321-427f-aa17-267ca6975798` is the fixed Azure DevOps resource ID. It is used only to mint the token. Copy the `descriptor` value (it looks like `aadsp.<base64>`) for the AWS Continuum for Azure DevOps service principal.

2. Grant View and Edit Subscriptions for the organization:

```
az devops security permission update \
  --org "https://dev.azure.com/ORG_NAME" \
  --namespace-id "cb594ebe-87dd-4fc9-ac2c-6a10a4c92046" \
  --subject "SP_DESCRIPTOR" \
  --token "PublisherSecurity" \
  --allow-bit 3 \
  --merge true
```

   Replace `SP_DESCRIPTOR` with the descriptor from the previous step. The namespace ID identifies the Service Hooks security namespace. `--allow-bit 3` grants View and Edit Subscriptions. `--merge true` preserves the service principal's other permissions.

Note

To scope the grant to a single project instead of the whole organization, get the project ID and append it to the token as `PublisherSecurity/PROJECT_ID`:

```
az devops project show \
  --org "https://dev.azure.com/ORG_NAME" \
  --project "PROJECT_NAME" \
  --query id \
  -o tsv
```

Repeat the grant for every project the agent analyzes.

Important

Grant **View and Edit Subscriptions** (`--allow-bit 3`), not Edit Subscriptions alone (`--allow-bit 2`). Without View, the webhook install fails a consumer availability check even though Edit is present. Without this permission, the integration registers successfully but AWS Continuum for Azure DevOps cannot install the pull request webhook, so pull requests are never reviewed.

Troubleshoot Azure DevOps integration

Pull requests are not reviewed

Symptoms

- The integration is registered and repositories are connected, but new pull requests are never analyzed and no comments are posted.

Resolution

- The pull request webhook was not installed because the service principal lacks the Service Hooks permission. Grant **View and Edit Subscriptions** (`--allow-bit 3`) on `PublisherSecurity` for the organization, as described in Grant AWS Continuum for Azure DevOps Service Hooks permissions. Edit alone (`--allow-bit 2`) is not sufficient.

To re-trigger project webhook creation, remove and add the repositories from that project.

Update rejected for exceeding the project limit

Symptoms

- Saving repository settings fails with "Code review is supported on repositories across at most 50 Azure DevOps projects per integration".

Resolution

- The update would enable code review comments or automated remediation in more than 50 projects on this integration, counted across all Agent Spaces. Disable these capabilities on repositories in projects you no longer need, then save again.

Can’t sign in with a personal account

Symptoms

- Choosing **Authorize** shows "You can’t sign in here with a personal account. Use your work or school account instead."

Resolution

- Authorizing AWS Security Agent grants tenant-wide admin consent, which requires a Microsoft Entra work or school account. A personal Microsoft account can’t grant it, and neither can a service principal. Sign in with an account in the Microsoft Entra tenant that your Azure DevOps organization is connected to, then start the registration again.

Your organization lacks a service principal for Azure DevOps

Symptoms

- Authorization fails with an error containing `AADSTS650052`, which names service `499b84ac-1321-427f-aa17-267ca6975798` (Azure DevOps).

Resolution

- Your Microsoft Entra tenant has no service principal for Azure DevOps. This happens in tenants that have never used Azure DevOps, because the service principal is created the first time the tenant uses it. Sign in to dev.azure.com once with an account in the tenant, then start the registration again.
- To confirm the service principal exists, in the Azure portal go to **Microsoft Entra ID**, then **Enterprise applications**, and filter **Application type** by **All applications**. The default filter hides Microsoft applications, so **Azure DevOps** does not appear under the **Enterprise Applications** filter even when it is present.

Consent required

Symptoms

- Authorization fails or **AWS Continuum for Azure DevOps** does not appear when you search in **Add users**.

Resolution

- Your Microsoft Entra policies require an administrator to approve the AWS Continuum for Azure DevOps application. Ask a Microsoft Entra administrator to approve the consent request, then register the connection and add the service principal again.

Next steps

After connecting Azure DevOps to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories
- Choose **Enable code review** or **Setup penetration testing** to connect specific repositories to your Agent Space (see the section called “Enable code review” and the section called “Enable penetration test”)
- Enable **Code review comments** to have AWS Security Agent analyze each pull request and post findings in Azure DevOps (see the section called “Review code security findings in pull requests”)
- Enable **Code remediation** to allow AWS Security Agent to submit pull requests with vulnerability fixes (see the section called “Enable users to start remediation of penetration test and code review findings”)

- Create threat models from connected repositories in the web application (see the section called "Enable threat modeling")

Remove an Azure DevOps integration

This procedure removes the connection between AWS Security Agent and an Azure DevOps organization. Use it when you no longer need AWS Security Agent to access that organization's repositories.

Prerequisites for removal

Before removing an Azure DevOps integration:

- Check which Agent Spaces have repositories connected from this integration.
- Note that removing the integration breaks code review, penetration testing context, threat modeling, and automated remediation for all connected repositories.

Step 1: Removing the integration from AWS Security Agent

First, remove the integration in the AWS Security Agent Management Console.

1. In the AWS Security Agent Management Console, choose **Integrations**.
2. Locate the Azure DevOps integration you want to remove.
3. Select the integration.
4. Choose **Remove**.
5. Review the confirmation dialog.
6. Choose **Confirm removal**.

When you remove the integration, AWS Security Agent deletes the pull request service hook subscriptions that it created in your Azure DevOps projects.

> **Important**
>
> Remove the integration before you remove the **AWS Continuum for Azure DevOps** service principal from your organization. AWS Security Agent uses the service principal's Service

Hooks permissions to delete its subscriptions. If you remove the service principal first, the removal might fail.

Step 2: Removing the service principal from your organization

After you remove the integration, you can remove the **AWS Continuum for Azure DevOps** service principal from your Azure DevOps organization.

1. In Azure DevOps, choose **Organization settings**.
2. Choose **Users**.
3. Locate **AWS Continuum for Azure DevOps**.
4. Open the actions menu for the service principal.
5. Choose **Remove from organization**.

Note

If you plan to register the same Azure DevOps organization again, you can keep the service principal in the organization. Registering again then doesn't require you to add the service principal or repeat the Service Hooks grant.

(Optional) Step 3: Removing the application from Microsoft Entra ID

To also revoke the admin consent that you granted when you registered the integration, delete the **AWS Continuum for Azure DevOps** enterprise application from your Microsoft Entra tenant.

1. In the Azure portal, choose **Microsoft Entra ID**.
2. Choose **Enterprise applications**.
3. Search for **AWS Continuum for Azure DevOps**.
4. Select **AWS Continuum for Azure DevOps**.
5. Choose **Properties**.
6. Choose **Delete**.

> **Important**
>
> The **AWS Continuum for Azure DevOps** enterprise application is shared by every Azure DevOps integration that uses your Microsoft Entra tenant. Delete it only if no other Azure DevOps organization in the tenant is connected to AWS Security Agent.

Find your Atlassian installation ID

The Atlassian installation ID uniquely identifies the AWS Security Agent Forge app installed on your Atlassian site, so AWS can securely link your site to your AWS account during integration setup. You need this ID when you register a Bitbucket, Confluence, or Jira integration in the AWS Console. Find the installation ID from the AWS Security Agent Forge app installed on your Atlassian site, then paste it into the registration flow.

For Bitbucket

To copy the installation ID in Bitbucket, do the following:

1. In Bitbucket, navigate to your **Workspace settings**.
2. On the sidebar, select **Forge Apps**.
3. In the list of installed apps, locate **Connect with AWS Security Agent**.
4. Choose **Copy to Clipboard** to copy the installation ID.

For Confluence

To copy the installation ID in Confluence, do the following:

1. In Confluence, navigate to **Confluence administration**.
2. On the sidebar, select **Apps**.
3. In the list of installed apps, locate **Connect with AWS Security Agent**.
4. Choose **Copy to Clipboard** to copy the installation ID.

For Jira

To copy the installation ID in Jira, do the following:

1. In Jira, navigate to **Jira administration**.
2. On the sidebar, select **Apps**.
3. In the list of installed apps, locate **Connect with AWS Security Agent**.
4. Choose **Copy to Clipboard** to copy the installation ID.

You will need this installation ID when completing the registration in the AWS Security Agent Management Console.

Connect AWS Security Agent to Bitbucket repositories

Connect your AWS Security Agent to Bitbucket Cloud repositories to enable code review, threat modeling, penetration testing, and automated remediation capabilities. Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces and shared across capabilities.

Bitbucket integration serves multiple purposes:

- **Continuum for code review** - Automatically analyze the code changes in each pull request against your organizational security requirements, and run on-demand full-repository scans
- **Continuum for threat modeling** - Provide application understanding by analyzing source code, data flows, and architecture
- **Continuum for penetration testing context** - Provide application understanding for penetration testing
- **Continuum for automated remediation** - Submit pull requests with fixes for vulnerabilities discovered during security assessments

Connecting Bitbucket to AWS Security Agent requires installing the AWS Security Agent Forge app on your Atlassian site and completing the OAuth authorization flow.

> **Note**
>
> This topic covers Bitbucket Cloud. For self-hosted Bitbucket Data Center, see the section called "Connect AWS Security Agent to Bitbucket Data Center".

How Bitbucket integration works

Pull request analysis happens within Bitbucket. After you install the Forge app, connect repositories, and enable code review comments in the AWS Management Console, AWS Security Agent automatically scans the changes in each new pull request (a differential scan of the changed code, which can also flag existing code that the changes make reachable) and posts findings as pull request comments.

You create and run **full code reviews** — which scan a repository's entire codebase — in the AWS Security Agent web application, not in Bitbucket.

Penetration testing and **threat modeling** are initiated within the AWS Security Agent web application. Users specify target domains and select connected repositories to provide application context.

> **Note**
>
> Automated remediation is not available for public Bitbucket repositories to avoid disclosing vulnerabilities before they are fixed.

Prerequisites

Before you begin, ensure you have:

- A Bitbucket Cloud workspace with admin access
- An Atlassian account with site administrator privileges
- Permissions to configure integrations in the AWS Security Agent Management Console

> **Important**
>
> If your Bitbucket workspace or the network in front of it restricts access with an IP allow list, add the AWS Security Agent IP addresses for your AWS Region before you register the integration. For the IP addresses, see the section called "AWS Security Agent IP addresses".

Important

One Atlassian site can only be associated with one AWS account per region. If you need to connect the same Bitbucket site to AWS Security Agent in a different AWS account within the same region, you must first remove the existing integration.

Note

Atlassian Forge app pricing applies to this integration. For more information, see Forge platform pricing in the Atlassian documentation.

Register a Bitbucket connection

The following table describes each permission the AWS Security Agent Forge app requests and why it is needed.

Permission	Purpose
`read:repository:bitbucket`	Read repository source code.
`write:repository:bitbucket`	Create a branch and commit remediation fixes to it. A Bitbucket Cloud pull request always merges a source branch into a destination branch, so the fix must be committed to a branch before AWS Security Agent can open a pull request.
`read:pullrequest:bitbucket`	Read pull request details and diffs to run automated code reviews.
`write:pullrequest:bitbucket`	Post review comments with findings and open remediation pull requests. Bitbucket Cloud bundles approving and merging pull requests into this same permission, but AWS Security Agent does not approve or merge pull requests.
`read:app-system-token`	Obtain the app token that AWS Security Agent uses to call the Bitbucket Cloud API.

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **Bitbucket**, then choose **Next**.
4. Install the AWS Security Agent Forge app in your Bitbucket workspace, following the on-screen instructions. After installation, copy the installation ID. See the section called “Find your Atlassian installation ID”.
5. In the **Installation ID** field, paste the installation ID you copied from Bitbucket.
6. In the **Bitbucket workspace** field, enter your Bitbucket workspace slug, for example `acme-corp`.
7. Choose **Authorize**.

   You are redirected to Atlassian to authorize AWS Security Agent to access your Bitbucket workspace. After authorization completes, you return to the console.
8. In the **Register details** section, enter a **Registration name** for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
9. Choose **Connect**.

> **Note**
>
> **Provisioning delay after connecting**
>
> After you choose **Connect**, provisioning on the Atlassian side can take a few minutes. During this time, the integration reports a pending status. If you try to select repositories or enable code review before provisioning completes, you might see a message that the installation is not yet available. Wait a few minutes and try again.

Troubleshoot Bitbucket integration

If you encounter issues during the Bitbucket integration process, use the following guidance to resolve common problems.

Unable to complete registration

If the registration process is interrupted (browser closed, session timeout), the Forge app may be installed on your Atlassian site but not registered in the AWS Console.

Resolution

- Return to the AWS Security Agent console and restart the integration process.
- The Forge app remains installed and does not need to be reinstalled.

Site already connected to another AWS account

Symptoms

- Error indicating the Atlassian site is already associated with another account

Resolution

- One Atlassian site can only be connected to one AWS account per region.
- Identify which AWS account has the existing integration and use that account, or remove the existing integration first.

Installation is not available

Symptoms

- When you select repositories or enable code review, you see a message that the Bitbucket installation is in status `PENDING_INSTALLATION`, not `AVAILABLE`.

Resolution

- The installation is still provisioning on the Atlassian side. Provisioning normally completes within a few minutes. Wait a few minutes, then try again.
- If the status does not become available after several minutes, remove the integration and register it again. Before re-registering, uninstall the Forge app from your Bitbucket workspace. For instructions, see Remove a Bitbucket integration. If you re-register without uninstalling the previous Forge app, the installation can get stuck in a pending state.

Next steps

After connecting Bitbucket to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories
- Choose **Enable code review** or **Setup penetration testing** to connect specific repositories to your Agent Space (see the section called "Enable code review" and the section called "Enable penetration test")
- Enable **Code review comments** to have AWS Security Agent analyze each pull request and post findings in Bitbucket (see the section called "Review code security findings in pull requests")
- Enable **Code remediation** to allow AWS Security Agent to submit pull requests with vulnerability fixes (see the section called "Enable users to start remediation of penetration test and code review findings")
- Create threat models from connected repositories in the web application (see the section called "Enable threat modeling")

Remove a Bitbucket integration

Remove a Bitbucket integration when you no longer need AWS Security Agent to access repositories from a specific Bitbucket workspace.

Prerequisites for removal

Before removing a Bitbucket integration:

- Check which Agent Spaces have repositories connected from this integration
- Understand the impact: Removing will break code review, penetration testing context, threat modeling, and automated remediation for all connected repositories

Step 1: Remove the integration from AWS Security Agent

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Locate the Bitbucket integration you want to remove.
3. Select the integration.
4. Choose **Remove**.
5. Review the confirmation dialog and choose **Confirm removal**.

Step 2: Uninstall the Forge app

After removing the integration from AWS Security Agent, uninstall the Forge app from your Atlassian site:

1. In Bitbucket, navigate to **Workspace settings**.
2. Select **Forge Apps**.
3. Locate **Connect with AWS Security Agent**.
4. Choose **Uninstall**.

> ⚠ **Important**
>
> **Forge app is not uninstalled automatically**
>
> Removing the integration in the AWS Security Agent console does not uninstall the Forge app from your Atlassian site. If you plan to register the same Bitbucket workspace again, you must uninstall the Forge app first. Otherwise, the new installation can get stuck in a pending state.

Connect AWS Security Agent to Bitbucket Data Center

Connect your AWS Security Agent to a Bitbucket Data Center instance to enable code review, threat modeling, penetration testing, and automated remediation capabilities for repositories hosted on your own infrastructure.

Bitbucket Data Center is a self-hosted provider. It is registered as its own integration type—separate from Bitbucket Cloud (see the section called “Connect AWS Security Agent to Bitbucket repositories”)—because the two use different authentication models. Bitbucket Data Center uses an OAuth application that you create on your own instance, with additional configuration for network connectivity to a private instance. Before you begin, review the section called “How integrations work with Agent Spaces” to understand how a registration is reused across Agent Spaces and shared across capabilities.

How Bitbucket Data Center integration works

Pull request analysis happens within Bitbucket Data Center. Unlike other providers, AWS Security Agent does not create the pull request webhook automatically for Bitbucket Data Center, because

doing so would require elevated permissions. Instead, after you register the integration and enable code review comments, you create a webhook on your instance using a payload URL and signing secret that AWS Security Agent generates for you (see the section called “Set up the pull request webhook”). The webhook notifies AWS Security Agent of new pull requests; it then scans the changes in each pull request (a differential scan of the changed code, which can also flag existing code that the changes make reachable) and posts findings as pull request comments.

You create and run **full code reviews**—which scan a repository’s entire codebase—in the AWS Security Agent web application, not in Bitbucket Data Center.

Penetration testing and **threat modeling** are initiated within the AWS Security Agent web application. Users specify target domains and select connected repositories to provide application context.

Prerequisites

Before you begin, ensure you have:

- A Bitbucket Data Center instance that is either:
  - Publicly accessible over the internet, OR
  - Accessible via a private connection (see the section called “Connect to privately hosted source control”)
- Administrator access to your Bitbucket Data Center instance to create an OAuth 2.0 external application (incoming) link with the **Repository Read** and **Repository Write** permissions (`REPO_READ` and `REPO_WRITE`). During registration, the AWS Security Agent console shows you the redirect URI and the permissions to configure on the link. Have the link’s client ID and client secret ready to enter in the console.
- Administrator access to the projects and repositories you want to connect
- Your instance must serve HTTPS traffic with a minimum TLS version of 1.2

> **Important**
>
> If your Bitbucket Data Center instance is reachable over the public internet and restricts inbound access with an IP allow list or firewall, add the AWS Security Agent IP addresses for your AWS Region before you register the integration. For the IP addresses, see the section called “AWS Security Agent IP addresses”. Instances reached through a private connection

do not need this step. That traffic arrives over the private network path, not from these public IP addresses.

Note

If your Bitbucket Data Center instance uses TLS certificates issued by a private certificate authority, you can provide the PEM-encoded public key of the certificate when creating a private connection. This allows AWS Security Agent to trust the TLS connection to your instance.

Register a Bitbucket Data Center connection

The following table describes each permission the OAuth application link requires and why it is needed.

Permission	Purpose
Repository Read (`REPO_READ`)	Read repository source code and pull request diffs.
Repository Write (`REPO_WRITE`)	Post review comments with findings, and push a branch with remediation fixes to open a pull request.

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Choose **Add integration**.
3. Select **Bitbucket Data Center**, then choose **Next**.
4. In the **Instance URL** field, enter the base URL of your instance, for example `https://bitbucket.example.com`.
5. The console displays the **redirect URI** (the AWS Security Agent Bitbucket Data Center OAuth callback) and the permissions to configure on your OAuth application link. The redirect URI is specific to your Agent Space's AWS Region and has the form:

```
https://securityagent.{region}.api.aws/oauth2/provider/register/callback/
bitbucketdatacenter
```

Copy the exact redirect URI shown in the console—it is the value your OAuth application link must redirect to.

6. In Bitbucket Data Center, create an OAuth 2.0 external application (incoming) link using the redirect URI shown in the console, and grant it the **Repository Read** and **Repository Write** permissions (`REPO_READ` and `REPO_WRITE`). In Bitbucket Data Center, **Repository Write** includes **Repository Read**, so selecting **Repository Write** grants both. Copy the generated **Client ID** and **Client secret**. See the following note about which user approves the link.
7. Back in the console, enter the **Client ID** and **Client secret** from the link.
8. If your instance is not publicly accessible, select **Connect to endpoint using a private connection**, then choose an existing private connection or create a new one. See the section called “Connect to privately hosted source control”.
9. In the **Registration name** field, enter a descriptive name for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
10. Choose **Connect**.

    AWS Security Agent runs the OAuth authorization against your instance and returns you to the **Integrations** page, where the new connection appears with its registration name.

Important

AWS Security Agent acts as the Bitbucket Data Center user who approves the OAuth authorization. It has no separate application identity on your instance. Every action it takes is performed as, and attributed to, that user. This includes reading source, fetching pull request diffs, posting comments, and opening remediation pull requests. Its access is capped at that user’s permissions.

We strongly recommend that a Bitbucket Data Center administrator create a dedicated user for this purpose (for example, `aws-security-agent`) and approve the authorization while signed in as that user. Then:

- **Scope that user’s access** to only the repositories and projects you intend AWS Security Agent to scan. Because the OAuth token inherits the user’s permissions, this is the primary control that bounds what AWS Security Agent can reach.

- **Recognize the identity in your logs.** AWS Security Agent's reads, comments, and pull requests appear in your Bitbucket Data Center audit logs and pull request history under this user. A dedicated user makes automated activity distinguishable from a real person's.

Set up the pull request webhook

Pull request scanning requires a webhook on your Bitbucket Data Center instance. Unlike other providers, AWS Security Agent does not create this webhook automatically, because doing so would require broad administrative permissions on your instance. Instead, AWS Security Agent generates the webhook's payload URL and signing secret, and you create the webhook on your instance with those values. Without the webhook, new pull requests are never analyzed and no comments are posted.

Set this up after you register the integration:

1. In the AWS Security Agent Management Console, open the **Integrations** page, find your Bitbucket Data Center connection, and choose **Manage**.
2. Choose **Create webhook**. AWS Security Agent generates:
   - A **Payload URL**—the endpoint your instance sends pull request events to.
   - A **Signing secret**—AWS Security Agent uses it to verify that inbound events genuinely originate from your instance.

   > **Important**
   >
   > The signing secret is shown only once. Copy it before you close the dialog—it cannot be retrieved afterward. If you lose it, rotate the secret (see the section called "Rotate the signing secret") to generate a new one.

3. In Bitbucket Data Center, create a webhook on each repository (or project) you want scanned:
   a. Set the **URL** to the payload URL from the console.
   b. Set the **Secret** to the signing secret from the console.
   c. Subscribe to these events:

   Repository

   - Push

Pull request

- Opened
- Source branch updated
- Modified
- Comment added

4. Save the webhook on your instance.
5. In the AWS Security Agent web application, configure the repository in an Agent Space to start receiving code review comments and automated remediation. See the section called “Enable code review”.

> **Important**
>
> Subscribe to only the events listed above. Always set the signing secret—without it, AWS Security Agent cannot verify that events came from your instance, and pull requests are not scanned.

Rotate the signing secret

If the signing secret is lost or compromised, rotate it. On the **Manage webhook** dialog, choose **Rotate**, then **Rotate secret**. Rotating generates a new signing secret and invalidates the current one; the payload URL does not change. Event delivery stops until you update the webhook in Bitbucket Data Center with the new secret, so update it promptly.

Private connectivity

If your Bitbucket Data Center instance is not publicly accessible, you must create a private connection before registering the integration. For instructions, see Create a private connection.

> **Important**
>
> Service-managed private connections require the Bitbucket Data Center instance to be running in the **same AWS account** where the Agent Space is created. For cross-account access, use a self-managed private connection where you provide your own VPC Lattice resource configuration.

Troubleshoot Bitbucket Data Center integration

Instance unreachable

Symptoms

- Connection fails with timeout or network error
- Integration was previously working but stops functioning

Resolution

- Verify your Bitbucket Data Center instance is running and accessible
- If using a private connection, verify the VPC Lattice resource gateway is healthy and the ENIs have network connectivity to your instance
- Verify security groups allow traffic on the configured port
- Verify the TLS certificate is valid and not expired

TLS certificate errors

Symptoms

- Connection fails with an SSL/TLS error

Resolution

- Verify your instance serves HTTPS with TLS 1.2 or higher
- If using a private certificate authority, ensure the PEM-encoded public key was provided during private connection setup
- Verify the certificate is not expired

Authorization fails

Symptoms

- The authorization step returns an error, or an operation reports that authentication failed.

Resolution

- A response from your instance means the request reached it and Bitbucket Data Center rejected the credentials—the problem is the OAuth application, not connectivity. Verify the client ID and client secret, and that the OAuth application grants the repository read (and, for comments or remediation, write) access AWS Security Agent needs.
- A connection timeout or a TLS error (rather than a response from your instance) indicates a network or certificate problem. See **Instance unreachable** and **TLS certificate errors**.

Pull requests are not reviewed

Symptoms

- The integration is registered and repositories are connected, but new pull requests are never analyzed and no comments are posted.

Resolution

- Verify that you created the webhook on your Bitbucket Data Center instance with the correct payload URL, signing secret, and events (see the section called “Set up the pull request webhook”). This step is manual for Bitbucket Data Center and is the most common reason pull requests are not scanned.
- Verify that code review comments are enabled for the repository in the Agent Space (see the section called “Enable code review”).
- Verify the webhook is enabled and uses the current signing secret, and that no firewall or IP allow list blocks delivery from the AWS Security Agent IP addresses for your AWS Region (see the section called “AWS Security Agent IP addresses”).

Next steps

After connecting Bitbucket Data Center to AWS Security Agent:

- Navigate to the Agent Space where you want to use these repositories
- Choose **Enable code review** or **Setup penetration testing** to connect specific repositories to your Agent Space (see the section called “Enable code review” and the section called “Enable penetration test”)

- Enable **Code review comments** to have AWS Security Agent analyze each pull request and post findings in Bitbucket Data Center (see the section called "Review code security findings in pull requests")
- Enable **Code remediation** to allow AWS Security Agent to submit pull requests with vulnerability fixes (see the section called "Enable users to start remediation of penetration test and code review findings")
- Create threat models from connected repositories in the web application (see the section called "Enable threat modeling")

Connect AWS Security Agent to Confluence

Connect your AWS Security Agent to Confluence Cloud to provide documentation context for security assessments. You can also use this connection to publish AWS Security Agent assessment results back to your Confluence site. Confluence serves as a documentation source that provides threat models, architecture documents, API specifications, and other materials that enhance the quality of security reviews. Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces.

Confluence integration serves multiple purposes:

- **Continuum for report publishing** – Create and update Confluence pages with the reports that AWS Security Agent generates from penetration tests, code reviews, and threat models
- **Continuum for penetration testing context** – Provide application documentation for deeper understanding during penetration testing
- **Continuum for threat modeling** – Provide existing threat models and system documentation for threat analysis

Connecting Confluence to AWS Security Agent requires installing the AWS Security Agent Forge app on your Atlassian site and completing the OAuth authorization flow.

> **Note**
>
> AWS Security Agent supports Confluence Cloud only. Confluence Data Center and Confluence Server are not supported.

How Confluence integration works

Confluence is a **documentation provider** rather than a source code provider. After you install the Forge app and connect spaces or pages in the AWS Management Console, AWS Security Agent can access your Confluence content to provide context during security assessments.

AWS Security Agent reads page content to understand your application architecture, security requirements, and design decisions. This context improves the quality and relevance of security findings during threat model analysis, code reviews, and penetration tests.

When you enable the create and update capabilities for a connected space, you can publish reports from AWS Security Agent to Confluence. These reports come from penetration tests, code reviews, and threat models. The agent creates a new page for each report, or updates a page it previously created. The create and update capabilities are both opt-in and disabled by default.

Prerequisites

Before you begin, ensure you have:

- A Confluence Cloud site with admin access
- An Atlassian account with site administrator privileges
- Permissions to configure integrations in the AWS Security Agent Management Console

> **Important**
>
> One Atlassian site can only be associated with one AWS account per region. If you need to connect the same Confluence site to AWS Security Agent in a different AWS account within the same region, you must first remove the existing integration.

> **Note**
>
> Atlassian Forge app pricing applies to this integration. For more information, see Forge platform pricing in the Atlassian documentation.

Register a Confluence connection

1. In the AWS Security Agent Management Console, navigate to **Integrations**.

2. Choose **Add integration**.
3. Select **Confluence**, then choose **Next**.
4. Install the AWS Security Agent Forge app in your Atlassian site, following the on-screen instructions. After installation, copy the installation ID. See the section called “Find your Atlassian installation ID”.
5. In the **Confluence site URL** field, enter your site URL, for example `https://acme.atlassian.net`.
6. In the **Installation ID** field, paste the installation ID you copied from Confluence.
7. Choose **Authorize**.

   You are redirected to Atlassian to authorize AWS Security Agent to access your Confluence site. After authorization completes, you return to the console.
8. In the **Register details** section, enter a **Registration name** for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
9. Choose **Connect**.

Select pages for an Agent Space

After you register the Confluence integration, connect specific pages to an Agent Space. Selecting a page grants AWS Security Agent read (fetch) access to that page’s content. In the review step of the connect wizard, you can remove any pages you do not want the agent to access.

For a connected space, you can also enable two optional capabilities: **Create pages** and **Update pages** (both are disabled by default). With these enabled, you can publish reports from AWS Security Agent to Confluence.

Troubleshoot Confluence integration

If you encounter issues during the Confluence integration process, use the following guidance to resolve common problems.

Unable to complete registration

If the registration process is interrupted, the Forge app may be installed on your Atlassian site but not registered in the AWS Console.

Resolution

- Return to the AWS Security Agent console and restart the integration process.
- The Forge app remains installed and does not need to be reinstalled.

Forge app uninstalled from Atlassian

If the AWS Security Agent Forge app is uninstalled from your Atlassian site while the integration still exists in AWS Security Agent:

Symptoms

- Integration appears in the AWS Console but cannot access Confluence content
- Errors when attempting to list or read pages

Resolution

- Reinstall the Forge app from the Atlassian Marketplace
- If the issue persists, remove the integration in the AWS Console and re-register

Site already connected to another AWS account

Resolution

- One Atlassian site can only be connected to one AWS account per region.
- Identify which AWS account has the existing integration and use that account, or remove the existing integration first.

Next steps

After connecting Confluence to AWS Security Agent:

- Navigate to the Agent Space where you want to use this documentation
- Select specific pages to include as context for code reviews and penetration tests
- Select pages as scope documents for threat models (see the section called “Enable threat modeling”)

- Upload additional documentation via S3 if needed (see the section called "Provide agent resources from an S3 bucket")

Remove a Confluence integration

Remove a Confluence integration when you no longer need AWS Security Agent to access documentation from a specific Confluence site.

Prerequisites for removal

Before removing a Confluence integration:

- Check which Agent Spaces have pages connected from this integration
- Understand the impact: Removing will break documentation context for design reviews, penetration testing, and threat modeling across all Agent Spaces using content from this integration

Step 1: Remove the integration from AWS Security Agent

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Locate the Confluence integration you want to remove.
3. Select the integration.
4. Choose **Remove**.
5. Review the confirmation dialog and choose **Confirm removal**.

Step 2: Uninstall the Forge app (optional)

After removing the integration from AWS Security Agent, you can optionally uninstall the Forge app from your Atlassian site:

1. In Confluence, navigate to **Confluence administration**.
2. Select **Apps**.
3. Locate **Connect with AWS Security Agent**.
4. Choose **Uninstall**.

Connect AWS Security Agent to Jira

Connect AWS Security Agent to Jira Cloud, Atlassian's issue tracking service, so you can track security findings as Jira issues. After you connect a Jira project to an Agent Space, you can create a Jira issue for a finding, and AWS Security Agent can add comments and links to that issue as the finding progresses. Before you begin, review the section called "How integrations work with Agent Spaces" to understand how a registration is reused across Agent Spaces.

When you connect Jira, you can do the following in the projects you choose:

- **Create issues** – Create a Jira issue for a security finding, so your team can track and resolve it with your usual Jira workflow.
- **Comment on issues** – Add comments to a linked issue, such as the result when you revalidate a finding.

To connect Jira to AWS Security Agent, install the AWS Security Agent Forge app on your Atlassian site and complete the authorization flow.

> **Supported Jira version**
>
> AWS Security Agent supports Jira Cloud only.

Reviewing prerequisites

Connect Jira to AWS Security Agent when you want to create and track security findings as Jira issues. Before you begin, ensure you have:

- A Jira Cloud site with admin access
- An Atlassian account with site administrator privileges
- Permissions to configure integrations in the AWS Security Agent Management Console

> **One site per AWS account**
>
> You can associate one Atlassian site with only one AWS account per Region. If you need to connect the same Jira site to AWS Security Agent in a different AWS account within the same Region, you must first remove the existing integration.

Registering a Jira connection

A Jira connection links AWS Security Agent to your Jira Cloud site so you can create and comment on Jira issues for your security findings. Register a connection before you connect projects to an Agent Space.

Part 1: Installing and authorizing the Forge app

1. On the AWS Security Agent Management Console, navigate to **Integrations**.
2. On the **Integrations** page, choose **Add integration**.
3. Choose **Jira**, then choose **Next**.
4. Install the AWS Security Agent Forge app in your Atlassian site, following the on-screen instructions.
5. After installation, copy the installation ID. For information about finding your installation ID, see Find your Atlassian installation ID.
6. In the **Jira site URL** field, enter your site URL, for example `https://example.atlassian.net`.
7. In the **Installation ID** field, paste the installation ID you copied from Jira.
8. Choose **Authorize**.

   Atlassian redirects you to authorize AWS Security Agent to access your Jira site. After authorization completes, you return to the console.

Part 2: Registering the connection

1. In the **Register details** section, enter a **Registration name** for this connection. Valid characters are letters, numbers, periods, underscores, and hyphens.
2. In the **Register details** section, choose **Connect**.

Connecting projects to an Agent Space

After you register the Jira integration, connect the Jira projects you want to use to an Agent Space. Connecting projects lets AWS Security Agent create and comment on Jira issues directly from findings, so your team can track and resolve security issues without leaving Jira. For each connected project, you choose whether AWS Security Agent can create issues, comment on issues, or both.

1. Open the Agent Space where you want to use Jira.
2. Start the Jira connection.
3. Select the projects you want to connect.
4. For each project, select the actions you want to allow:
   - **Create issues** – With this permission, AWS Security Agent can create issues in the project.
   - **Comment on issues** – With this permission, AWS Security Agent can comment on issues in the project.
5. Choose **Save** to apply your selection.

After you connect a project, you can create Jira issues for findings from the AWS Security Agent web application. For more information about creating Jira issues for findings, see Create Jira issues for findings.

Troubleshooting Jira integration

Jira integration issues can occur during registration, when the Forge app is uninstalled, or when a site is already connected to another AWS account. The following guidance helps you diagnose and resolve these common problems.

Resolving an incomplete registration

If the registration process is interrupted, the Forge app can be installed on your Atlassian site but not registered in the AWS Management Console.

Resolution

To resolve this issue, do the following:

- Return to the AWS Security Agent console and restart the integration process.
- Don’t reinstall the Forge app, because it remains installed.

Resolving an uninstalled Forge app

If you uninstall the AWS Security Agent Forge app from your Atlassian site while the integration still exists in AWS Security Agent, the integration can no longer create or comment on issues. Use the following guidance to restore it.

Symptoms

You might encounter the following symptoms:

- The integration appears in the AWS Management Console, but it cannot create or comment on issues.
- You receive errors when you create a Jira issue for a finding.

Resolution

To resolve this issue, do the following:

- Reinstall the Forge app from the Atlassian Marketplace.
- If the issue persists, remove the integration in the AWS Management Console and re-register it.

Resolving a site already connected to another AWS account

Resolution

To resolve this issue, do the following:

- You can connect one Atlassian site to only one AWS account per Region.
- Identify which AWS account has the existing integration and use that account, or remove the existing integration first.

Next steps

After you connect Jira to AWS Security Agent, you can do the following:

- Connect the Jira projects you want to use to an Agent Space.
- Create Jira issues for findings from a penetration test or code review. For more information, see Create Jira issues for findings.

This integration uses an Atlassian Forge app, and Atlassian Forge pricing applies. For more information about Forge platform pricing, see Forge platform pricing on the Atlassian website.

Remove a Jira integration

Remove a Jira integration when you no longer need AWS Security Agent to create or comment on issues in a specific Jira site.

Prerequisites for removal

Before removing a Jira integration:

- Check which Agent Spaces have projects connected from this integration
- Understand the impact: After you remove the integration, users can no longer create Jira issues for findings in the Agent Spaces that use it. Issues that AWS Security Agent already created in Jira are not affected.

Step 1: Remove the integration from AWS Security Agent

1. In the AWS Security Agent Management Console, navigate to **Integrations**.
2. Locate the Jira integration you want to remove.
3. Select the integration.
4. Choose **Remove**.
5. Review the confirmation dialog and choose **Confirm removal**.

Step 2: Uninstall the Forge app (optional)

After removing the integration from AWS Security Agent, you can optionally uninstall the Forge app from your Atlassian site:

1. In Jira, navigate to **Jira administration**.
2. Select **Apps**.
3. Locate **Connect with AWS Security Agent**.
4. Choose **Uninstall**.

Connect to privately hosted source control

AWS Security Agent can connect to source control systems running in private networks, such as GitLab Self-Managed, GitHub Enterprise Server, and Bitbucket Data Center instances. Private

connections use Amazon VPC Lattice to establish secure connectivity between AWS Security Agent and your private infrastructure without exposing your systems to the public internet.

How private connections work

A private connection creates a secure network path between AWS Security Agent and a target resource in your VPC. Under the hood, AWS Security Agent uses Amazon VPC Lattice to establish this connectivity. VPC Lattice is an AWS application networking service for connecting, securing, and monitoring traffic between services across VPCs and accounts, without you managing the underlying network infrastructure.

When you create a private connection:

1. You provide the VPC, subnets, and (optionally) security groups that have network connectivity to your target service.
2. AWS Security Agent creates a service-managed resource gateway and provisions elastic network interfaces (ENIs) in the subnets you specified.
3. The agent uses the resource gateway to route traffic to your target service's IP address or DNS name over the private network path.

Service-managed vs. self-managed private connections

AWS Security Agent supports two modes for private connections:

Service-managed (recommended for simple setups):

- AWS Security Agent creates and manages the VPC Lattice resource gateway for you.
- The target service must be running in the **same AWS account** where the Agent Space is created.
- Cannot span multiple AWS accounts.

Self-managed (for advanced or cross-account setups):

- You create and manage your own VPC Lattice resource configuration.
- You provide the Amazon Resource Name (ARN) of your existing resource configuration.
- Supports cross-account access (customer manages cross-account networking).

Prerequisites

Before creating a private connection, verify that you have:

- An active Agent Space
- A privately reachable target service (GitLab Self-Managed, GitHub Enterprise Server, or Bitbucket Data Center) at a known private IP address or DNS name
- The target service must serve HTTPS traffic with a minimum TLS version of 1.2
- Subnets in your VPC (1-20 subnets). We recommend selecting subnets in multiple Availability Zones for high availability.
- (Optional) Security groups to control traffic to the ENIs (up to 5)

> **Note**
>
> The following Availability Zones are not supported by VPC Lattice: `use1-az3`, `usw1-az2`, `apne1-az3`, `apne2-az2`, `euc1-az2`, `euw1-az4`, `cac1-az3`, `ilc1-az2`.

> **Note**
>
> Private connections are account-level resources. After you create a private connection, you can reuse it across multiple integrations and Agent Spaces that need to reach the same host.

> **Note**
>
> When you select a private connection for a provider that uses OAuth authentication, the private connection applies to both the provider endpoint and the token exchange endpoint. Ensure the private connection is configured with a host address that can route traffic to both endpoints.

Create a private connection

1. In the AWS Security Agent Management Console, navigate to **Integrations**.

2. Choose the **Private connections** tab.
3. Choose **Create private connection**.
4. Under **Connection details**, enter a **Name**. Names can contain lowercase letters, numbers, and hyphens, and must start and end with a letter or number.
5. In the **Connection details** section, choose how AWS Security Agent connects to your target service:
   - To have AWS Security Agent create and manage the connection, leave **Use existing resource configuration** cleared, then complete the **Resource location**, **Access control**, and **Service target details** sections.
   - To route through a VPC Lattice resource configuration that you already created, select **Use existing resource configuration**, then complete the **Existing resource configuration** section. The service-managed sections do not apply.
6. (Service-managed) Under **Resource location**:
   a. Select the **VPC where your resource is located**.
   b. Select one or more **Subnets**. We recommend selecting subnets in at least two Availability Zones.
   c. (Optional) Select an **IP address type** (IPv4, IPv6, or Dual stack).
7. (Service-managed) Under **Access control**:
   a. Select the **Security groups associated with your connected resources**.
   b. (Optional) Under **Advanced configuration**, enter **Port ranges** — up to 11 comma-separated TCP ports or ranges, for example `443, 8080-8090`.
8. (Service-managed) Under **Service target details**:
   a. In the **Host address** field, enter the DNS host name or IP address of your target service.
   b. For **DNS resolution**, select **Public** for a publicly resolvable host address, or **In VPC (private DNS)** for an address resolvable only inside the VPC.
   c. (Optional) In the **Certificate public key** field, enter the PEM-encoded public key if your target uses a self-signed certificate or a private certificate authority. This lets AWS Security Agent trust the TLS connection.
9. (Self-managed) Under **Existing resource configuration**, for **Resource configuration**, select a VPC Lattice resource configuration from the list, or enter a resource configuration ID (beginning with `rcfg-`) or its ARN. The list shows resource configurations in the current Region.
10. (Optional) Under **Tags**, add one or more tags to the private connection. Each tag is a key-value pair that helps you organize and identify your resources.

11. Choose **Create connection**.

    The connection status changes to **Create in progress**. This can take up to 10 minutes. When complete, the status changes to **Available**.

Use a private connection with an integration

When you register a GitLab Self-Managed, GitHub Enterprise Server, or Bitbucket Data Center integration, select **Connect to endpoint using a private connection**, then choose your connection from the **Private connection** list. AWS Security Agent routes traffic to the provider through that connection. Only connections with an **Available** status appear in the list.

You can also choose **Create a private connection** during registration. AWS Security Agent preserves your registration draft while you create the connection, then returns you to the registration flow.

Security

- **No public internet exposure** - All traffic stays on the AWS network.
- **Customer-controlled security groups** - You manage inbound and outbound traffic rules.
- **Service-linked role with least privilege** - AWS Security Agent uses a service-linked role scoped to resources tagged with `AWSSecurityAgentManaged`.

> **Note**
>
> If your organization has service control policies (SCPs) that restrict VPC Lattice API actions, the service-managed resource gateway is created through a service-linked role. Ensure your SCPs permit the necessary actions for the service-linked role.

Verify a private connection

After the private connection reaches the **Available** status, verify connectivity:

- Ensure your target service is running and accepting connections on the expected port
- Verify that security groups attached to the ENIs allow outbound traffic on the target port

- Verify that your service's security group allows inbound traffic from VPC Lattice data plane IPs within your VPC CIDR range
- Confirm that subnet route tables allow traffic between the ENI subnets and your service

Delete a private connection

1. In the AWS Security Agent console, navigate to **Integrations**.
2. Choose the **Private connections** tab.
3. Select the private connection you want to delete.
4. Choose **Delete**.

> **Important**
>
> You cannot delete a private connection that is currently in use by an integration. Remove the integration first, then delete the private connection.

Advanced setup using existing VPC Lattice resources

If your organization already uses Amazon VPC Lattice and manages your own resource configurations, you can create a private connection in self-managed mode. You provide the Amazon Resource Name (ARN) of an existing resource configuration that points to your target service. You manage the resource gateway yourself, rather than having AWS Security Agent create one.

Consider self-managed mode when you want to:

- Control the resource gateway and resource configuration lifecycle yourself.
- Share a resource configuration across multiple AWS accounts or services.
- Use VPC Lattice access logs for detailed traffic monitoring.
- Run a hub-and-spoke network architecture.

To use an existing resource configuration, select **Use existing resource configuration** when you create the private connection, then choose a configuration under **Existing resource configuration**. For the full console steps, see Create a private connection earlier on this page.

The resource configuration picker lists configurations in the current console Region. If the list is empty or access is denied, enter a resource configuration ID (starting with `rcfg-`) or its ARN directly.

> **Note**
>
> To populate the resource configuration picker, AWS Security Agent needs the `vpc-lattice:ListResourceConfigurations` permission. If this call returns an access-denied error, the picker falls back to manual entry. You can paste the resource configuration ID or ARN instead.

In self-managed mode, AWS Security Agent does not collect a certificate public key in the console. Your resource configuration defines the target service and its TLS settings.

For more information about creating VPC Lattice resource gateways and resource configurations, see the Amazon VPC Lattice User Guide.

Cross-Region connectivity

You must create a private connection in the same AWS Region as your Agent Space. If your target service runs in a different Region, use self-managed mode with inter-Region Amazon VPC peering or AWS Transit Gateway peering.

Follow this pattern:

1. Establish inter-Region connectivity between a VPC in the Agent Space Region and the target service's VPC. Use VPC peering or Transit Gateway peering. The VPC CIDR ranges must not overlap.
2. Create a VPC Lattice resource gateway in the Agent Space Region, in a VPC that has the peering connection.
3. Create a VPC Lattice resource configuration in the Agent Space Region that points to the target service's IP address. The address must be routable through the peering connection.
4. Create a self-managed private connection that uses that resource configuration ARN.

Next steps

- the section called “Connect AWS Security Agent to GitLab Self-Managed” - Connect a private GitLab instance
- the section called “Connect AWS Security Agent to GitHub Enterprise” - Connect a private GitHub Enterprise instance
- the section called “Connect AWS Security Agent to Bitbucket Data Center” - Connect a private Bitbucket Data Center instance

Connect agent to private VPC resources

If the application you want to run a penetration test on is not available on the public internet, you need to provide AWS Security Agent with a VPC configuration. AWS Security Agent will use this VPC configuration, including a VPC, subnet, and security groups, to access the application.

Note

When testing endpoints in a private VPC, only endpoints resolving to IPs in known private IP ranges are allowed (see VPC CIDR blocks for more information). The following IPv4 and IPv6 ranges are allowed:

```
10.0.0.0/8
172.16.0.0/12
192.168.0.0/16
fd00::/8
```

Note

When connecting to a subnet, AWS Security Agent will create an ENI (Elastic Network Interface) in the subnet configured for the penetration test. This ENI does not have an associated public IP address, meaning that it cannot communicate with VPC Internet Gateways in public subnets. If your penetration test requires open internet access, please use a private subnet with an associated VPC NAT Gateway instead

> **Tip**
>
> When testing against an endpoint that has an IP allowlist, you can add a private VPC configuration with an associated VPC NAT Gateway to your penetration test. You can then use the NAT Gateway IP address to allowlist outbound traffic from the penetration test.

You grant AWS Security Agent general access to a VPC from the AWS Management Console. In the Security Agent web application, users select the specific configuration for a penetration test.

To add a VPC in the Agent Space

1. Navigate to the Agent Space overview page
2. Select **Actions** and then **Edit penetration test configuration**
3. Under the **VPC** heading, specify the **VPC**, **Subnets**, and **Security groups**

You can add up to 5 VPCs.

Required Agent Space service role permissions

To run a penetration test with a VPC, your Agent Space service role must include the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeNetworkInterfaceAttribute",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:DeleteNetworkInterface"
      ],
      "Resource": "*"
```

```
    },
    {
      "Effect": "Allow",
      "Action": "ec2:CreateNetworkInterfacePermission",
      "Resource": "arn:aws:ec2:{{region}}:{{accountId}}:network-interface/*",
      "Condition": {
        "ArnEquals": {
          "ec2:Subnet": [
            "arn:aws:ec2:{{region}}:{{accountId}}:subnet/[[subnetIds]]"
          ]
        }
      }
    }
  ]
}
```

To select a specific VPC configuration for a penetration test in the Security Agent web application

1. Navigate to the Penetration Tests overview page
2. Select the penetration test that you need to add VPC configuration for, and then choose **Modify pentest details**
3. Choose **Save & Next** to reach the **VPC Resources** step
4. Select the **VPC**, **Subnet**, and **Security groups**
5. Choose **Save & Next** on each remaining step
6. On the last step, choose **Save** to save the penetration test

> **Note**
>
> Cross-account penetration testing is currently supported for VPC resources (subnets and security groups) shared using AWS Resource Access Manager. Secrets Manager secrets and Lambda functions used for authentication credentials must be configured in the same AWS account as your AWS Security Agent setup.

Running a penetration test against VPC resources in another AWS account

You can run penetration tests against VPC resources shared with your account using AWS Resource Access Manager. Both accounts must be part of the same AWS Organization.

1. (Optional) Enable automatic resource sharing for your AWS organization

```
aws ram enable-sharing-with-aws-organization
```

2. Using credentials from the AWS account that owns the VPC resources, share subnet and security group resources with the penetration test owner account

```
aws ram create-resource-share \
    --name SharePentestResources \
    --resource-arns <subnet ARN> <security group ARN> \
    --principals <penetration test owner account ID>
```

3. Navigate to the Agent Space overview page
4. Select **Penetration test** and locate **Service role name**
5. Verify that the IAM role grants access to the shared VPC resources
6. Select **Actions** and then **Edit penetration test configuration**
7. Under the **VPC** heading, specify the shared **VPC**, **Subnets**, and **Security groups** and save the updated configuration.
8. Navigate to the Penetration Tests overview page on the AWS Security Agent web application
9. Select the penetration test that you need to add VPC configuration for, and then choose **Modify pentest details**
10. Update the penetration test to use the shared VPC resources

Configure capabilities

Enable and configure penetration testing, code review, and threat modeling for your Agent Spaces, including remediation, S3 sources, and target domains.

Enable penetration test

Configure AWS Security Agent to run autonomous penetration tests on your applications. This setup enables AWS Security Agent to access your AWS resources, verify domain ownership, and perform comprehensive security testing that identifies exploitable vulnerabilities in your web applications and APIs.

Enabling penetration testing allows AWS Security Agent to continuously validate your application security without the delays of manual testing. By configuring the necessary AWS integrations, you ensure AWS Security Agent can test both public and private applications, log findings to CloudWatch, and access credentials for authenticated testing.

In this procedure, you'll configure target domains, optionally set up VPC, CloudWatch logging, credentials storage, and Lambda functions for testing, configure S3 integrations for providing additional context, and set up service access through IAM roles.

Prerequisites

Before you begin, ensure you have:

- AWS account with administrative permissions to create IAM roles
- Domain ownership verification capability (DNS or HTTP record modification)
- Target domains or applications to test
- (Optional) VPC configuration details if testing private applications
- (Optional) S3 bucket if providing additional artifacts to AWS Security Agent

Step 1: Configure domain

In the first step of the wizard, specify the target domains you want to test and how AWS Security Agent should verify ownership.

1. In the **Target domains** section, enter your domain in the **Domain** field.
2. Select a **Verification method**:
   - **DNS_TXT** – Prove domain ownership by adding a TXT record to your domain's DNS configuration.
   - **HTTP_ROUTE** – Prove domain ownership by hosting a verification file at a specific URL on your domain.

- **PRIVATE_VPC** - Only usable for private VPC penetration testing. Verifies that the domain resolves to an IP in a private CIDR range
- For more information, see the section called "Enable an application domain for penetration testing".

3. Choose **Add another domain** to add additional domains (up to 20 total).
4. Choose **Next** to proceed to domain verification.

Step 2: Verify domains

In the second step of the wizard, verify ownership of each domain you configured. AWS Security Agent requires verified ownership before it can perform penetration testing.

1. Review the domains listed in the **Target domains** table.
2. For each domain, select it and trigger verification based on your chosen method:
   - **Route 53 domains (same AWS account)**: Choose **One-click verification**. AWS Security Agent automatically creates the DNS record and completes verification.
   - **DNS TXT (other DNS providers)**: Copy the verification token, add the TXT record with your DNS registrar, then select the domain and choose **Verify**.
   - **HTTP route**: Place the verification token at the required route path on your web server, then select the domain and choose **Verify**. For details, see the section called "Enable an application domain for penetration testing".
3. Choose **Next** to proceed to optional configuration.

> **Note**
>
> Sub-domains of a verified domain do not require individual verification when using DNS TXT or HTTP route verification. For Private VPC verification, the target endpoint domain name must match the full domain name configured for verification.

> **Note**
>
> For private domains inside a VPC, you can proceed even if the domain verification status is UNREACHABLE. AWS Security Agent will attempt domain verification for private endpoints at the start of each penetration test run.

Step 3: (Optional) Configure additional capabilities

The third step of the wizard lets you configure optional AWS resources to expand your penetration testing scope. All sections in this step are optional and collapsed by default.

(Optional) Configure VPC settings

If you plan to test private target domains hosted within a VPC, configure VPC settings for AWS Security Agent. This section is optional and collapsed by default.

1. Expand the **VPCs** section.
2. In the **VPC** dropdown, select the VPC that hosts your private target domains.
3. In the **Subnet** dropdown, select the VPC subnets that AWS Security Agent should use:

   > **Tip**
   >
   > For high availability, select multiple subnets from multiple Availability Zones. Ensure your subnets include a NAT gateway for outbound connectivity.

4. In the **Security group** dropdown, select the VPC security groups that AWS Security Agent should use:

   > **Important**
   >
   > Ensure your security groups allow outbound connections for AWS Security Agent to perform penetration testing.

(Optional) Configure CloudWatch logging

Configure CloudWatch to store logs from your penetration test runs. This section is optional and collapsed by default.

1. Expand the **CloudWatch logs** section.
2. In the **Log Groups** dropdown, select existing CloudWatch log groups
3. If not selecting any log group, AWS Security Agent will create a log group named `/aws/securityagent/<agent name>/<pentest id>` with appropriate permissions.

> **Note**
>
> Ensure your IAM role has permissions to write to the selected CloudWatch log group.

(Optional) Configure Secrets for test credentials

If your application requires authentication credentials for testing, AWS Security Agent can securely retrieve them from AWS Secrets Manager. This section is optional and collapsed by default.

1. Expand the **Secrets** section.
2. Select the AWS Secrets Manager secrets that contain credentials for your application.
3. When configuring a penetration test in the web application, you can reference these secrets for authenticated testing.

> **Important**
>
> Credentials are encrypted and stored in AWS Secrets Manager. Ensure your IAM role has permissions to access Secrets Manager for AWS Security Agent to use these credentials during testing.

(Optional) Configure Lambda functions for test credentials

Configure Lambda functions that can provide credentials for your application during testing. This section is optional and collapsed by default.

1. Expand the **Lambda functions** section.
2. Select the Lambda functions that can provide authentication credentials for your application.
3. AWS Security Agent will invoke these functions during penetration tests to obtain credentials dynamically.

> **Note**
>
> Ensure your IAM role has permissions to invoke the specified Lambda functions. Lambda functions should return credentials in the expected format for AWS Security Agent to use during testing.

(Optional) Configure S3 bucket

Provide S3 bucket details if you plan to upload documents or artifacts to provide as input to AWS Security Agent. This section is optional and collapsed by default.

1. Expand the **S3 buckets** section.
2. In the **Bucket** field, enter or search for your S3 bucket name.

> **Note**
>
> You can also connect GitHub repositories later or upload files directly in the web application. Information provided can ensure thorough coverage, reduce false positives, and deliver actionable results.

(Optional) Configure service access

AWS Security Agent requires an IAM role to access your AWS resources (VPC, CloudWatch log groups, Secrets Manager, Lambda functions, etc.) for penetration testing. This section is optional and collapsed by default.

1. Expand the **Service access** section.
2. By default, AWS Security Agent uses a default IAM role with the required permissions for penetration testing.
3. To customize the IAM role, select one of the following options:
   a. **Create default role** – AWS Security Agent automatically creates a new IAM role with the necessary permissions
   b. **Use an existing service role** – Select an existing IAM role from the dropdown menu
4. If using an existing role:

a. Choose the dropdown menu under **Choose an existing role**
b. Select your IAM role from the list
c. Choose the refresh icon to update the list if needed

> **Note**
>
> The default IAM role includes permissions for accessing VPC resources, CloudWatch log groups, Secrets Manager, Lambda functions, and other services required for penetration testing. It is recommended to use the default IAM role unless you have specific security requirements.

(Optional) Enable penetration test scheduling

Enable scheduling if you want penetration tests in this Agent Space to run automatically, either once or on a repeating schedule. For how to create a schedule in the web application, see the section called “Schedule a penetration test”.

1. Expand the **Penetration test scheduling** section.
2. Select **Enable scheduling**.

When you save the configuration, the console creates an IAM role named `continuum-schedule-`*`agentSpaceId`* in your account. Amazon EventBridge Scheduler assumes this role to start the scheduled runs. Each Agent Space has its own role. For the policies that the console attaches to it, see the section called “Scheduler execution role”.

You can also enable scheduling for an existing Agent Space without editing its configuration. On the Agent Space details page, choose the **Penetration test** tab, and then choose **Enable scheduling**. In the **Enable penetration test scheduling** dialog box, choose **Enable**. The **Penetration test scheduling** section on the tab shows the status and the name of the role.

Step 4: Save and enable penetration testing

After configuring all required settings, enable penetration testing for your AWS Security Agent agent.

1. Review all configuration sections to ensure accuracy.

2. Choose **Save**.
3. AWS Security Agent validates your configuration and creates the necessary AWS resources.

Next steps

After enabling penetration testing:

- Configure penetration test scopes in the AWS Security Agent web application
- Set up notification preferences for test findings
- Review and respond to penetration test findings as they are discovered
- (Optional) Configure additional repositories for findings remediation

For more information about running and managing penetration tests, see the AWS Security Agent web application documentation.

Enable pull request code review for GitHub repositories

Pull request code review is now configured as part of the code review setup wizard. When you connect GitHub repositories to your Agent Space, you can enable pull request comments for individual repositories to have AWS Security Agent automatically analyze pull requests and post security findings.

For complete setup instructions, see the section called “Enable code review”.

For information about how pull request findings appear in GitHub and how to respond to them, see the section called “Review code security findings in pull requests”.

Enable code review

Configure your Agent Space to enable code review by connecting source code from GitHub repositories or S3 buckets. Code review analyzes your source code for security vulnerabilities and compliance with your organization’s custom security requirements.

Setting up code review configurations is an Agent Space-wide operation. The integrations and S3 buckets you connect are shared across capabilities, including code review and penetration testing.

After completing setup, users can create and run code reviews in the AWS Security Agent web application to scan repositories and S3 sources for security issues.

> **Note**
>
> If you already have GitHub repositories connected to your Agent Space (for example, through penetration testing setup), code review is already enabled. You can skip this setup and go directly to the web application to create code reviews. See the section called "Create a code review".

Prerequisites

Before you begin, ensure you have:

- An Agent Space created in the AWS Management Console (see the section called "Create an Agent Space")
- At least one of the following source code inputs:
  - A GitHub organization or user account with the AWS Security Agent GitHub App installed (see the section called "Connect AWS Security Agent to GitHub repositories")
  - An S3 bucket containing source code you want to review
- Permissions to configure integrations for your Agent Space
- (Optional) At least one security requirement enabled in a security requirement pack if you plan to use security requirement validation (see the section called "Manage security requirements")

Access the code review setup wizard

Navigate to the code review setup for your Agent Space.

1. In the AWS Security Agent console, select your Agent Space.
2. Choose **Enable code review** from one of the following locations:
   - The **Code review** card
   - The **Code review** tab, then choose **Enable code review**

> **Tip**
>
> If you would like AWS Security Agent to address code review feedback automatically, enable **Code remediation** as well. See the section called "Enable users to start remediation of penetration test and code review findings" for details.

You are directed to the **Setup code review configurations** wizard.

Step 1: Connect integrations, repos, and buckets

In the first step of the wizard, connect your source code inputs and configure code review settings. You must add at least one GitHub repository or S3 bucket to proceed.

> **Important**
>
> Integrations and S3 buckets configured here are shared across your Agent Space. Changes apply to both code review and penetration testing capabilities.

Connect GitHub repositories

Add GitHub repositories from your authorized GitHub organizations or user accounts. Choosing **Add** opens the two-step **Connect GitHub** wizard, where you first select repositories and then configure what actions AWS Security Agent can take on each one.

1. In the **Connected integrations** section, choose **Add**.
2. Select the GitHub registration that contains the repositories you want to review.
3. On the **Connect GitHub repositories** step, select the checkbox for each repository you want to connect.
4. Choose **Next** to go to **Manage capabilities**.

> **Note**
>
> Connected repositories are accessed read-only for code analysis during code review and penetration testing. You configure write actions such as posting review comments and opening remediation pull requests on the next step.

> **Note**
>
> If you haven't registered a GitHub integration yet, choose **Settings** to navigate to the Integrations page where you can authorize the AWS Security Agent GitHub App. For more information, see the section called "Connect AWS Security Agent to GitHub repositories".

Configure GitHub repository capabilities

On the **Manage capabilities** step of the **Connect GitHub** wizard, choose what AWS Security Agent can do in each repository. **Code review comments** and **Code remediation** are set independently per repository.

1. For each repository, toggle **Code review comments** on to have AWS Security Agent post security findings as comments on pull requests.
2. For each repository, toggle **Code remediation** on to let AWS Security Agent fix findings. When enabled, web application users can request pull requests that fix findings, and pull request authors can comment `@AWS-Security-Agent fix all findings.` to have the agent address its review comments.
3. Choose **Save** to apply your selections and return to the code review setup wizard.

When **Code review comments** is enabled for a repository:

- By default, AWS Security Agent automatically analyzes pull requests when they are marked ready for review. Draft pull requests are not analyzed.
- You can limit reviews by target branch or label, review draft pull requests, or start a review when a label is added. For more information, see Configure pull request code review triggers.
- To start a review on demand, comment `@AWS-Continuum-Pentesting review` on the pull request. For more information, see the section called "Manually trigger a review".
- Security findings are posted as review comments directly on the pull request with specific remediation guidance.
- The analysis uses your configured code review settings (security vulnerabilities, custom requirements, or both).

> **Note**
>
> Pull request comments are only available for private GitHub repositories.

When **Code remediation** is enabled for a repository, web application users can start remediation for both code review and penetration test findings on that repository, and AWS Security Agent delivers each fix as a pull request. For more information, see the section called “Enable users to start remediation of penetration test and code review findings”.

For more information about how pull request findings appear in GitHub and how to respond to them, see the section called “Review code security findings in pull requests”.

Add S3 buckets

Add S3 buckets containing source code or contextual resources for code review.

1. In the **S3 buckets** section, choose **Add S3 resource**.
2. Enter the S3 URI for the bucket or prefix containing your source code.
3. Choose **Add**.

> **Note**
>
> You can add up to 10 S3 resources. S3 buckets are shared across capabilities including code review and penetration testing.

> **Tip**
>
> You can add S3 buckets that contain source code, configuration files, infrastructure-as-code templates, or other artifacts you want AWS Security Agent to analyze for security issues.

Configure code review settings

Configure the types of security issues AWS Security Agent analyzes during code reviews. This setting applies to all repositories and sources with code review enabled in this Agent Space.

1. In the **Code review settings** section, select one of the following options:
   - **Security requirement validation** – Validate whether code complies with the security requirements you've enabled in your security requirement packs.
   - **Security vulnerability findings** – Identify common security vulnerabilities in code.
   - **Security requirements and vulnerability findings** – Analyze code for both compliance with the security requirements you've enabled and common security vulnerabilities. This is the default setting.

> **Note**
>
> When security requirement validation is enabled, AWS Security Agent checks code against the security requirements you've enabled in your security requirement packs. If you select security requirement validation but do not have at least one security requirement enabled, AWS Security Agent will not identify requirement-based findings. For more information about security requirements, see the section called "Manage security requirements".

1. Choose **Next** to proceed to optional configurations.

Step 2: Optional configurations

In the second step of the wizard, configure optional CloudWatch logging and service access settings for your code review environment.

(Optional) Configure CloudWatch logs

Configure CloudWatch log groups to capture and analyze application behavior during code review.

1. Expand the **CloudWatch logs** section.
2. In the **Log Groups** dropdown, select one or more existing CloudWatch log groups from your AWS account.

> **Note**
>
> If you don't select a log group, AWS Security Agent automatically creates a default log group to store code review execution logs.

(Optional) Configure service access

Configure the IAM service role that AWS Security Agent uses to access your AWS resources such as S3 buckets and CloudWatch logs for code review.

1. Expand the **Service access** section.
2. Select one of the following options:
   - **Create default role** – AWS Security Agent automatically creates a new IAM role with the necessary permissions for code review.
   - **Use an existing service role** – Select an existing IAM role from the dropdown menu.
3. If using the default role, enter a **Service role name**. The name must be unique across all roles in the account, use alphanumeric characters and `+=,.@-_` characters, and cannot include spaces.

> **Note**
>
> The service role name has a maximum length of 64 characters. A service role is automatically created if you don't select an existing role.

1. Choose **Save** to complete the setup.

After setup

After completing the code review setup wizard:

- The **Code review** card on your Agent Space page shows a **Ready** status.
- Users can launch the web application and create code reviews to scan connected repositories and S3 sources.
- You can modify your configuration at any time by choosing **Edit configuration** on the **Code review** tab.

Edit code review configuration

To modify your code review configuration after initial setup:

1. Navigate to your Agent Space in the AWS Security Agent console.

2. Select the **Code review** tab.
3. Choose **Edit configuration**.
4. Update your integrations, S3 buckets, code review settings, CloudWatch logs, or service access as needed.
5. Choose **Save**.

Next steps

After setting up code review configurations:

- Launch the web application to create and run code reviews (see the section called "Create a code review")
- Connect additional GitHub repositories or S3 buckets as your codebase grows
- Configure security requirement packs for organization-specific validation (see the section called "Manage security requirements")
- Review how pull request findings appear in GitHub (see the section called "Review code security findings in pull requests")

Configure pull request code review triggers

Configure when AWS Security Agent reviews pull requests and merge requests in connected repositories. By default, AWS Security Agent reviews a pull request when it's ready for review. It doesn't review draft pull requests.

To focus reviews on the pull requests that matter to you, use trigger filters. Trigger filters can limit reviews by target branch or label. They can also review draft pull requests or start a review when a label is added. A review that you start with a pull request comment doesn't use trigger filters.

Trigger filters apply only to repositories where you turn on **Code review comments**. AWS Security Agent checks each filter group on its own. A pull request starts a review if it meets every condition in at least one group.

Prerequisites

Before you begin, make sure that you have:

- An Agent Space. For more information, see Create an Agent Space.

- A connected repository. For more information, see Connect integrations.
- Code review enabled for the repository. For more information, see Enable code review.
- Permission to edit the repository's connected integration settings in AWS Security Agent.

Configure trigger filters

1. In the AWS Security Agent console, select your Agent Space. Then select the **Code review** tab, and choose **Edit configuration**.
2. In **Connected integrations**, edit the repositories that you want to configure.
3. On the **Manage capabilities** step, turn on **Code review comments** for each repository that you want AWS Security Agent to review.
4. Expand **Code review comments trigger filters**. In the section for the repository, choose **Add filter group**.
5. Configure the group. Each group needs trigger events, conditions, or both.
   - For **Trigger events (optional)**, choose the events that start a review.
   - For **Conditions (optional)**, choose **Add condition**, and then choose **Target branches** or **Labels**. Choose **Include matches** or **Exclude matches**. Then enter a name or RE2 regular expression, and choose **Add**. Repeat to add more patterns.
6. (Optional) To also review pull requests that meet other criteria, add another filter group. You can add up to five groups for each repository.
7. Choose **Save**.

A group without trigger events applies when a pull request is opened, updated, or marked ready for review, including draft pull requests. A group without conditions applies to all branches and labels. A pull request must meet every condition in a group. **Include matches** is met when any pattern matches, and **Exclude matches** is met when no pattern matches.

> **Filter groups replace the default trigger**
>
> When a repository has filter groups, AWS Security Agent reviews a pull request only if it matches one of them. A group without trigger events also reviews draft pull requests. To limit reviews by branch without reviewing drafts, choose **Pull request ready for review** for **Trigger events (optional)**.

You can also configure trigger filters with the `triggerFilterGroups` capability in the UpdateIntegratedResources API operation.

Filter patterns

Patterns are RE2 regular expressions, not wildcards, and each pattern must match the complete branch or label name. You can add up to 20 patterns to a condition, and the conditions in all of a repository's groups can contain up to 100 patterns in total. Each pattern can contain up to 256 characters. RE2 doesn't support lookarounds or backreferences.

The following table shows example patterns.

Pattern	Matches
`main`	Only `main`
`release/.*`	Branches that start with `release/`, such as `release/1.0`
`main|develop`	`main` and `develop`
`release/*`	Only `release` and `release/`, because `*` repeats the preceding `/`
`security-.*`	Labels that start with `security-` , such as `security-review`

Review pull requests based on labels

For GitHub and GitLab, a **Labels** condition matches the labels currently on the pull request. A group that reviews new pull requests also reviews one that is opened with a matching label. A **Labels** condition with **Exclude matches**, such as one for `skip-review`, prevents reviews while that label is on the pull request.

To start a review when someone adds a label, create a filter group that has only the **Pull request label added** event, or **Merge request label added** for GitLab. Then add a **Labels** condition with **Include matches**. A review starts only when the label that was added matches that condition. Adding an unrelated label to a pull request that already has a matching label doesn't start another review.

Labels conditions aren't available for Bitbucket Cloud or Azure DevOps.

Manually trigger a review

To start a review at any time, add a comment that contains only `@AWS-Continuum-Pentesting review` to an open pull request. In GitHub, add the comment to the main conversation, not to a line of code or a review thread. AWS Security Agent reviews all of the changes in the pull request, even if the pull request doesn't match a filter group. The repository must have **Code review comments** turned on.

If a pull request isn't reviewed

AWS Security Agent doesn't comment on pull requests that don't match a filter group. If a pull request isn't reviewed, check the following:

- **Code review comments** is turned on for the repository.
- The pull request is open. Draft pull requests are reviewed only by a group without trigger events or a group that includes the **Pull request drafted** event.
- The pull request meets every condition in at least one filter group.
- Each pattern matches the complete branch or label name.

To review a pull request that doesn't match any filter group, see the section called "Manually trigger a review".

Related topics

- Review code security findings in pull requests – what AWS Security Agent posts when a review runs.
- Service Quotas – each review counts toward the monthly quota for pull request code reviews. Reviewing draft pull requests or every update uses more of the quota.
- Enable code review – remediation commands, such as `@AWS-Security-Agent fix all findings`, follow the **Code remediation** setting, not trigger filters.

Enable threat modeling

Configure your Agent Space to enable threat modeling by connecting source code repositories and configuring AWS resources. Threat modeling analyzes your application's architecture and identifies security threats from source code, design documents, or both.

Setting up threat modeling configurations is an Agent Space-wide operation. The integrations and S3 buckets you connect are shared across capabilities, including threat modeling, code review, and penetration testing.

After completing setup, users can create and run threat models in the AWS Security Agent web application.

> **Note**
>
> If you already have repositories or S3 buckets connected to your Agent Space (for example, through code review or penetration testing setup), threat modeling may already be enabled. You can go directly to the web application to create a threat model. See the section called “Create a threat model”.

Prerequisites

Before you begin, ensure you have:

- An Agent Space created in the AWS Management Console (see the section called “Create an Agent Space”)
- Permissions to configure integrations for your Agent Space
- (Optional) A GitHub, GitLab, or Bitbucket organization with the AWS Security Agent app installed (see Connect AWS Security Agent to GitHub repositories, Connect AWS Security Agent to GitLab repositories, or Connect AWS Security Agent to Bitbucket repositories)

Access the threat modeling setup wizard

Navigate to the threat modeling configuration for your Agent Space.

1. In the AWS Security Agent console, select your Agent Space.
2. Choose **Configure threat model** from the **Threat model** card, or from the **Threat model** tab.

You are directed to the **Configure threat model** wizard, which has two optional steps.

Step 1: Connect source code repositories (optional)

Connect the GitHub, GitLab, or Bitbucket repositories you want to enable threat modeling for. Threat models themselves are created and viewed in the web application.

> **Important**
>
> Integrations configured here are shared across your Agent Space. Changes apply to threat modeling, code review, and penetration testing capabilities.

Connect repositories

1. In the **Connected integrations** section, choose **Add**.
2. Select the registration that contains the repositories you want to use.
3. Select the checkbox for each repository you want to connect.
4. Choose **Save** to apply your selections.

> **Note**
>
> If you haven't registered an integration yet, choose **Settings** to navigate to the Integrations page where you can authorize the AWS Security Agent app. For more information, see Connect AWS Security Agent to GitHub repositories, Connect AWS Security Agent to GitLab repositories, or Connect AWS Security Agent to Bitbucket repositories.

1. Choose **Next** to proceed to optional configurations.

Step 2: Optional configurations

Configure S3 buckets, CloudWatch logging, and service access settings for your threat modeling environment. These settings are shared with other capabilities on your Agent Space.

S3 buckets (optional)

Add S3 buckets containing source code you want the agent to use as context during threat modeling.

1. In the **S3 buckets** section, choose **Add S3 resource**.
2. Enter the S3 URI for the bucket or prefix.
3. Choose **Add**.

> **Note**
>
> You can add up to 10 S3 resources.

CloudWatch logs (optional)

Configure CloudWatch log groups to capture and analyze application behavior during threat model runs.

1. In the **CloudWatch logs** section, select one or more existing CloudWatch log groups from your AWS account.

Service access

Configure the IAM service role that AWS Security Agent uses to access your AWS resources such as S3 buckets and CloudWatch logs for threat modeling. A service role is required to enable threat modeling.

1. In the **Service access** section, select an existing IAM service role from the dropdown, or leave it empty to have a service role automatically created.

> **Note**
>
> A service role will be automatically created if you don't select an existing role.

1. Choose **Save** to complete the configuration.

After setup

After completing the threat modeling configuration:

- The **Threat model** card on your Agent Space page shows a **Ready** status.
- Users can launch the web application and create threat models from connected source code, uploaded scope docs, Confluence pages, or a combination of these inputs.

Next steps

After enabling threat modeling:

- Launch the web application to create and run threat models (see Create a threat model)
- Connect additional repositories or S3 buckets as your codebase grows
- Connect Confluence to enable selecting pages as scope documents (see Connect AWS Security Agent to Confluence)

Enable users to start remediation of penetration test and code review findings

In the AWS Management Console, you can enable automatic remediation so users of the AWS Security Agent web application can request fixes for a specific finding. AWS Security Agent delivers each fix as a GitHub pull request on the affected repository.

You enable remediation per repository from within an Agent Space. The **Penetration test** and **Code review** tabs open the same repository configuration wizard, so you can use either tab to manage the **Automatic remediation enabled** setting. Remediation applies to findings from both capabilities, so you only need to enable it once per repository.

Prerequisites

Before you begin, ensure you have:

1. Enabled penetration testing (see the section called “Enable penetration test”) or code review (see the section called “Enable code review”)
2. Installed and authorized the AWS Security Agent GitHub App for your GitHub organization (see the section called “Connect AWS Security Agent to GitHub repositories”)

Open the repository configuration wizard

Choose the entry point that matches how your repositories are set up.

From the penetration test tab

Use this path to add GitHub repositories for penetration testing and configure remediation in the same pass.

1. Navigate to the Agent Space overview page.
2. Choose the **Penetration test** tab.
3. Select a GitHub registration that owns your repositories:
   a. If you haven't associated any GitHub registration with the Agent Space, choose **Add** in the **Connect GitHub to AWS Security Agent** information box to select a registration.
   b. If one or more GitHub registrations are already associated, choose **Add** in the **Connected integrations** section to select another.
4. Choose **Next** to choose GitHub repositories.
5. Choose **Next** to configure repository capabilities.

From the code review tab

Use this path when your repositories are already connected for code review.

1. Navigate to the Agent Space overview page.
2. Choose the **Code review** tab.
3. In the GitHub repositories table, choose **Edit** to open the repository configuration wizard.

Enable automatic remediation and save

After you reach the repository capabilities step through either path above:

1. In the **Automatic remediation enabled** column, mark each repository you want remediated as **Enabled**.
2. Choose **Connect** to save the configuration.

Users of the web application can now start remediation for findings on these repositories, and AWS Security Agent opens pull requests with the proposed fixes.

Provide agent resources from an S3 bucket

Connect an Amazon S3 bucket to your Agent Space to give AWS Security Agent the source code and documents it works from, such as source code archives, API documents, and threat model or design documents.

S3 buckets are an Agent Space-wide integration. The buckets you connect are shared across capabilities. The same bucket can supply source code for code review and threat modeling, and supporting context — such as API specifications and design documents — for penetration testing. You connect a bucket once. When you create a code review, penetration test, or threat model in the web application, you select the specific resources you need.

Providing relevant resources helps the agent understand your application, which can improve coverage, reduce false positives, and produce more actionable findings.

Prerequisites

Before you begin, ensure you have:

- An Agent Space created in the AWS Management Console (see the section called “Create an Agent Space”).
- An Amazon S3 bucket in the same AWS account as your Agent Space that contains the resources you want to provide.
- Permissions to configure integrations for your Agent Space.

Add an S3 bucket

You add S3 buckets while configuring a capability for your Agent Space. The **S3 buckets** section appears in the code review, penetration test, and threat modeling configuration, and the buckets you add there are shared across all three.

1. In the AWS Security Agent console, select your Agent Space.
2. Open the configuration for any capability that uses shared integrations, such as the **Code review** or **Penetration test** tab, and go to the **S3 buckets** section.
3. Choose **Add S3 resource**.
4. Enter the **S3 URI** for the bucket or prefix that contains your resources.
5. Choose **Add**.

> **Note**
>
> You can add up to 10 S3 resources. S3 buckets are shared across capabilities, including code review, penetration testing, and threat modeling.

Requirements for code review sources

For code review, S3 sources must be ZIP files stored in a connected bucket. Any ZIP file in a connected bucket can be selected as a source when a user creates a code review. Resources that you provide as context for penetration testing or threat modeling, such as API documents or design documents, do not need to be ZIP files.

Next steps

- Enable code review and select S3 sources (see the section called “Enable code review”).
- Configure penetration testing for your Agent Space (see the section called “Enable penetration test”).
- Run a threat model on connected sources (see the section called “Enable threat modeling”).
- Run a differential code scan from a diff stored in S3 (see the section called “Run a differential code scan with S3”).

Enable an application domain for penetration testing

Before you can run a penetration test on an application, you need to add a target domain and verify ownership. AWS Security Agent will only perform penetration tests against verified domains.

> **Note**
>
> You do not need to validate ancillary domains that your application may use. You only need to validate the domain you will actively run penetration tests against.

Step 1: Add a target domain

To add a target domain, navigate to the **Penetration test** tab on the Agent Space overview page. Depending on whether you have already configured penetration testing, use one of the following methods:

- **First-time setup**: Choose **Set up penetration test** to open the penetration test wizard. In the first step, enter the domain and choose a verification method.
- **Adding a domain to an existing configuration**: In the **Target Domains** table, choose **Add domain**. Enter the domain and choose a verification method in the modal.

You can add a base domain or a sub-domain, such as `example.com` or `billing.example.com`. AWS suggests using a sub-domain where you have permission to create TXT records.

> **Note**
>
> When you verify a domain using DNS TXT or HTTP route verification, sub-domains of that domain are automatically covered. For example, verifying `example.com` allows you to test `api.example.com` or `billing.example.com` without additional verification. For Private VPC verification, the target endpoint domain name must match the full domain name configured for verification.

Choose one of the following verification methods:

- **DNS TXT record**: Prove domain ownership by creating a DNS TXT record with your DNS provider.
- **HTTP route**: Prove domain ownership by creating a route on your web server that contains a unique token provided by AWS Security Agent.
- **Private VPC**: Only usable for private VPC penetration testing. Verifies that the domain resolves to an IP in a private CIDR range. The domain name configured must match the full domain name of any associated target endpoints

Step 2: Verify domain ownership

After adding the domain, verify ownership using the method you selected. You can trigger verification at any time from the **Target Domains** table by selecting the domain and choosing **Verify**.

Verify using a DNS TXT record

AWS Security Agent generates a TXT DNS record value. You must add this record with your DNS provider to prove ownership.

If the domain is registered in Route 53 (same AWS account):

AWS Security Agent can create the DNS record automatically. Select the domain from the Target Domains table and choose **One-click verification**. AWS Security Agent creates the DNS validation record and completes verification automatically.

If the domain is registered with another DNS provider:

1. Copy the TXT record value provided by AWS Security Agent.
2. Add the TXT record with your DNS registrar.
3. Return to the Target Domains table, select the domain, and choose **Verify**.

Verify using HTTP route validation

This method proves domain ownership by placing a unique token on your web server. Only domain owners or authorized web administrators can create routes on a web server, which proves ownership.

1. Create a file at the following path on your web server:

```
.well-known/aws/securityagent-domain-verification.json
```

2. Place the token provided by AWS Security Agent in the file using this format:

```
{
  "tokens": ["<insert-token>"]
}
```

3. Return to the Target Domains table, select the domain, and choose **Verify**. AWS Security Agent sends an HTTPS GET request to the verification URL and validates the token.
4. If the domain is accessible on the public internet, make sure that your domain has a valid SSL certificate before running verification.

> **Note**
>
> If your domain is registered in multiple Agent Spaces and you are using HTTP route validation, you can place the tokens for both Agent Spaces in the same `tokens` array.

Bypass domain ownership verification

Customers who have authorization to perform penetration testing on an endpoint but cannot complete ownership verification can request manual verification from us. Please open a customer support case to make this request. Your request must include your business use case and your justification for manual ownership verification (i.e., why you are authorized to perform penetration testing on the endpoint).

Managed target domains used for penetration testing

In the AWS Management Console, you can add and manage target domains consumed by Agent Spaces for penetration testing. You must verify these target domains before you can use them in a penetration test. For more information about verifying target domains, see the section called "Enable penetration test".

Prerequisites

Before you begin, ensure you have:

1. Enabled penetration test (see the section called "Enable penetration test").

Manage target domain resources

- Navigate to the Target Domains overview page.
- You should see all target domain resources associated with your account.
  - If you don't see any target domains associated with your account, follow the steps in the section called "Enable penetration test" to create and verify a target domain.
- Target domains can be reused between Agent Spaces and share verification status.
  - To add an existing target domain to an Agent Space, navigate to the **Penetration test** tab of the Agent Space. Select **Add domain** and choose the desired domain under **Select from available previously registered domains** in the domain name field.
  - Target domains must be associated with an Agent Space before they can be used in a penetration test.
- Removing a domain from an Agent Space does not delete the domain. The associated target domain can be permanently deleted from the Target Domains overview page.

Verify target domains

In order for a target domain to be used in a penetration test, it must first be verified using one of the below methods:

- **Route 53 domains (same AWS account)**: Choose **One-click verification**. AWS Security Agent automatically creates the DNS record and completes verification.
- **DNS TXT (other DNS providers)**: Copy the verification token, add the TXT record with your DNS registrar, then select the domain and choose **Verify**.
- **HTTP route**: Place the verification token at the required route path on your web server, then select the domain and choose **Verify**. For details, see the section called “Enable an application domain for penetration testing”.
- **Private VPC**: Verifies the target domain's IP falls within a private CIDR range (see the section called “Connect agent to private VPC resources” for a list of private CIDR ranges). The penetration test target endpoint domain name must match the full domain name configured for verification. Only usable for private VPC penetration testing.

> **Note**
>
> For DNS TXT, HTTP route, and Route 53 verification, sub-domains of a verified domain are automatically covered and do not require separate verification. For Private VPC verification, each domain must be verified individually.

Manage security requirements

Define the security requirements AWS Security Agent evaluates during design and code reviews, using AWS-managed packs, custom packs, or requirements imported from your own documentation.

Manage security requirements

Organize the security requirements that AWS Security Agent uses to analyze your applications into security requirement packs. A pack is a collection of security requirements. You can enable AWS-managed packs, create your own custom packs, and generate requirements for a custom pack by uploading your security documentation.

Overview

A security requirement defines a security standard or policy that AWS Security Agent evaluates your application against during design reviews and code reviews. Code reviews include full scans of your entire codebase and differential (diff) scans of only the changed code. When a design or code does not meet a requirement, AWS Security Agent reports a finding. Every security requirement belongs to a security requirement pack and is shared across all Agent Spaces.

AWS Security Agent provides two types of packs, shown on separate tabs:

- **Managed security requirements packs** – AWS-provided packs of security requirements based on industry standards and best practices. You can enable these packs instantly to evaluate against common security policies without custom configuration. The requirements in a managed pack are read-only. You can use an AWS-managed requirement as a template for a custom requirement. For the list of available managed packs, see AWS managed security requirement packs.
- **Custom security requirements packs** – Packs that you create to enforce your organization's specific policies and standards. You build the requirements in a custom pack from scratch, customize AWS-managed requirements as a starting point, or generate them by uploading your security documentation.

> **Note**
>
> The compliance framework packs are designed to help identify security requirements that may be relevant for compliance with certain frameworks. They do not assess your compliance or guarantee that you will pass an audit.

You enable and disable packs as a unit. When a pack is enabled, AWS Security Agent evaluates your applications against the requirements in that pack during design and code reviews.

> **Note**
>
> Enabling or disabling a pack applies to new design reviews and code reviews. Existing reviews are not affected.

Enable or disable a managed pack

Enable an AWS-managed pack to evaluate your applications against a set of AWS-maintained security requirements. Disable it when you no longer need it.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Managed security requirements packs** tab.
4. Select the pack you want to enable or disable.
5. Do one of the following:
   a. To enable the pack, choose **Enable pack**.
   b. To disable the pack, choose **Disable pack**.

> **Tip**
>
> Choose a pack to view the security requirements it contains. Choose a requirement to view its full definition, including its applicability, compliance criteria, and remediation guidance.

Create a custom pack

Create a custom pack to group security requirements that are specific to your organization. After you create the pack, add requirements to it manually, customize an AWS-managed requirement, or upload documents to generate requirements.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Custom security requirements packs** tab.
4. Choose **Create security requirements pack**.
5. Enter a **Security requirement pack name** and an optional description.
6. Choose **Create security requirement pack**.

> **Note**
>
> After you create a pack, you can add or upload source documents to generate the security requirements for your pack.

Add a security requirement manually

Add a security requirement to a custom pack to define a security standard that AWS Security Agent enforces.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Custom security requirements packs** tab, and then choose the pack you want to add a requirement to.
4. Choose **Add requirement manually**.
5. Configure the following fields:
   a. **Security requirement name** – Enter a descriptive name that identifies the security control, for example `enforce-encryption-at-rest` (maximum 80 characters).
   b. **Description** – Provide a brief description of what this security requirement checks (maximum 500 characters).
   c. **Applicability** – Describe the scenarios, system types, or conditions where this security requirement should be evaluated. Include the scenarios where the requirement should be marked NOT_APPLICABLE (maximum 10,000 characters).
   d. **Compliance criteria** – Define what constitutes compliance versus non-compliance for this security requirement. Provide the indicators, examples, and technical details that AWS Security Agent looks for when it evaluates compliance (maximum 10,000 characters).
   e. **Remediation guidance** (Optional) – Provide guidance on how to fix violations, including links to your organization's internal documentation or standards (maximum 10,000 characters).
6. Do one of the following:
   a. To create the requirement without enabling it, choose **Create security requirement**.
   b. To create the requirement and enable it, choose **Create and enable security requirement**.

> **Note**
>
> A requirement is evaluated during security reviews only when the pack that contains it is enabled. To control whether a pack is evaluated, enable or disable the pack.

Customize an AWS-managed security requirement

Create a custom security requirement that uses an AWS-managed requirement as its starting point. AWS Security Agent pre-populates the requirement details from the managed requirement, and you edit them to fit your organization.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Custom security requirements packs** tab, and then choose the custom pack to add the requirement to.
4. Choose **Create custom security requirement**.
5. To pre-populate the form, in the **Customize an AWS-managed security requirement** section, search for and select an AWS-managed requirement to use as a template.
6. Edit any of the fields to fit your organization's needs.
7. Do one of the following:
   a. To create the requirement without enabling it, choose **Create security requirement**.
   b. To create and immediately enable the requirement, choose **Create and enable security requirement**.

> **Note**
>
> Customizing an AWS-managed requirement creates an independent custom security requirement. Later changes to the AWS-managed requirement do not affect your custom version.

Generate requirements by uploading documents

Generate the security requirements for a custom pack from your existing security documentation instead of writing each requirement by hand. AWS Security Agent reads the documents you upload,

identifies the security-relevant content, and generates structured requirements that you can review and edit. For the full procedure, see Generate security requirements from documents. For guidance on what to include in the documents you upload, see Prepare documents for requirement generation.

> **Important**
>
> Uploading new source documents regenerates all requirements for the pack. Any existing requirements, including ones you added manually, are replaced.

Edit a custom security requirement

Modify a custom security requirement to update its definition, criteria, or remediation guidance. You cannot edit the requirements in an AWS-managed pack.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Custom security requirements packs** tab, and then choose the pack that contains the requirement.
4. Select the requirement you want to edit, and then choose **Edit custom security requirement**.
5. Update the requirement fields as needed. The security requirement name cannot be changed after creation.
6. Choose **Update security requirement**.

> **Note**
>
> Changes to a security requirement apply to new design reviews and code reviews. Existing reviews are not affected.

Enable or disable a pack

Enable or disable a security requirement pack to control whether AWS Security Agent evaluates the requirements it contains. You enable and disable packs as a unit.

1. In the AWS console, navigate to AWS Security Agent.

2. In the navigation pane, choose **Security requirements**.
3. Choose the tab for the type of pack, and then select the pack.
4. Do one of the following:
   a. To enable the pack, choose **Enable pack**.
   b. To disable the pack, choose **Disable pack**.

> **Note**
>
> When a pack is enabled, the requirements in the pack are evaluated during security reviews. When a pack is disabled, its requirements are not evaluated.

Delete a custom security requirement

Delete a custom security requirement when you no longer want AWS Security Agent to evaluate it.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Custom security requirements packs** tab, and then choose the pack that contains the requirement.
4. Select one or more security requirements, and then choose **Delete security requirement**.
5. Confirm the deletion.

> **Note**
>
> When you delete security requirements, they are no longer evaluated in future reviews. The requirements remain visible in past reviews as read-only results.

Best practices for defining security requirements

When you create a security requirement, follow these guidelines to help AWS Security Agent evaluate your applications accurately and provide actionable findings.

Security requirement name – Use a clear, specific name that identifies the security control. Avoid generic terms that do not convey the requirement's purpose.

Description – Explain what the control checks and the risk it mitigates. This helps users understand why the requirement matters.

Applicability – Describe the workloads, system types, or conditions where the requirement should be evaluated. State when the requirement should be marked NOT_APPLICABLE, with specific scenarios, to avoid false positives. Phrases such as "This control applies to ALL workloads that…" and "Mark as NOT_APPLICABLE if…" set clear scope boundaries.

Compliance criteria – Structure this in two parts: what demonstrates compliance and what indicates non-compliance. Be specific with technical indicators and include edge cases. Start with "A design is compliant if it demonstrates…" followed by technical details, then "A design is non-compliant if it…" with the patterns that indicate a violation.

Remediation guidance – Provide guidance with technical details and configuration examples. Include links to your organization's internal documentation so that developers have the resources they need to fix violations.

Next steps

After you configure your security requirement packs:

- Create a design review or code review to evaluate your application against the packs you enabled.
- Enable penetration testing to complement your design and code reviews.
- Grant users access to the AWS Security Agent web application.

AWS managed security requirement packs

An AWS managed security requirement pack is a collection of security requirements that AWS creates and maintains, based on industry standards and best practices. You enable a managed pack to evaluate your applications against its requirements during design reviews and code reviews, without writing requirements yourself.

The security requirements in a managed pack are read-only. You cannot change them. You can use an AWS-managed requirement as a template to create a custom requirement, and then edit the copy to fit your organization. For more information, see Manage security requirements.

AWS updates managed packs over time. When AWS adds or revises a requirement in a managed pack, the change applies to every account that has the pack enabled.

> **Note**
>
> The compliance framework packs are designed to help identify security requirements that may be relevant for compliance with certain frameworks. They do not assess your compliance or guarantee that you will pass an audit.

AWS managed pack: ASA Base Pack

A baseline set of security requirements that apply to most workloads. This pack covers common application security concerns such as authentication and authorization, data protection, logging, and input validation. Enable this pack to establish a security baseline for your design and code reviews.

AWS managed pack: AWS Well-Architected Pack

Security requirements derived from the security pillar of the AWS Well-Architected Framework. This pack evaluates your application against AWS guidance for protecting data, managing identities and permissions, and detecting and responding to security events. For more information about the framework, see Security Pillar - AWS Well-Architected Framework.

AWS managed pack: NIST CSF Pack

Security requirements derived from the NIST Cybersecurity Framework (CSF). This pack evaluates your application against control areas drawn from the framework, such as identity and access management, data security, and protective technology. Enable this pack to align your design and code reviews with NIST CSF.

AWS managed pack: PCI DSS Pack

Security requirements derived from the Payment Card Industry Data Security Standard (PCI DSS). This pack evaluates your application against control areas relevant to handling cardholder data, such as access control, encryption of data in transit and at rest, and logging. Enable this pack to align your design and code reviews with PCI DSS.

Generate security requirements from documents

Generate the security requirements for a custom pack by uploading your existing security documentation. AWS Security Agent reads the documents you upload, identifies the security-relevant content, and generates structured security requirements with applicability, compliance

criteria, and remediation guidance. You can review and edit the generated requirements before they are used in reviews.

Use this instead of writing each requirement by hand when you already have security policies, standards, or guidelines documented. For guidance on what to include in the documents you upload, see Prepare documents for requirement generation.

Supported file formats

You can upload up to 3 files for each pack, with a combined total of 6 MB. The maximum size of a file depends on its format:

DOC

2 MB

DOCX

2 MB

MD

0.5 MB

PDF

2 MB

TXT

0.5 MB

> **Note**
>
> Markdown and text files have a lower size limit because AWS Security Agent analyzes them as plain text. A DOC, DOCX, or PDF file of the same size contains less text, because its size also includes formatting and embedded content.

Generate requirements

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.

3. Choose the **Custom security requirements packs** tab.
4. Create a pack, or choose an existing custom pack to generate requirements for.
5. Choose **Choose files**, or drag and drop your documents into the upload area.
6. Choose **Generate requirements**.
7. Wait for AWS Security Agent to process the documents. Generation runs in the background and can take a couple of minutes for large files. You cannot start another upload for the pack while generation is in progress.
8. Review the generated security requirements. Edit, delete, or add requirements as needed. Enable the pack when you want AWS Security Agent to evaluate its requirements during security reviews.

> **Note**
>
> AWS Security Agent generates requirements at the workload level and focuses on the security-relevant content in your documents. The number of requirements it generates depends on the content you provide. Review the generated requirements to confirm they reflect your intent.

Replace requirements with a new upload

Uploading new source documents regenerates the requirements for a pack.

> **Important**
>
> When you upload new source documents to a pack, AWS Security Agent regenerates all requirements for that pack. Any existing requirements, including ones you added manually, are replaced. To keep your current requirements, do not upload new documents.

1. In the AWS console, navigate to AWS Security Agent.
2. In the navigation pane, choose **Security requirements**.
3. Choose the **Custom security requirements packs** tab, and then choose the pack.
4. Start a new upload, and acknowledge that uploading new source documents replaces the existing requirements.

5. Choose your files, and then choose **Generate requirements**.

Next steps

- Review and enable the generated requirements. See Manage security requirements.
- Create a design review or code review to evaluate your application against the pack.

Prepare documents for requirement generation

When you generate security requirements from documents, AWS Security Agent reads your documents and produces structured requirements from the security-relevant content it finds. You do not need to format your documents in any particular way or pre-write the applicability, compliance criteria, and remediation guidance. AWS Security Agent derives those from the content you provide. Upload the security documentation you already have, written in your own words.

This page describes what to include so that AWS Security Agent generates accurate, useful requirements. For the upload procedure and file limits, see Generate security requirements from documents.

What to include

AWS Security Agent looks for content that describes your security expectations. Documents that cover the following topics produce the strongest requirements:

- Access control and authorization
- Authentication
- Data protection and encryption
- Network security
- Logging and audit
- Incident response
- Vulnerability and patch management
- Compliance obligations that apply to your workloads

Good source documents include security policies, engineering standards, control catalogs, architecture guidelines, and secure coding standards.

Write at the workload level

AWS Security Agent generates requirements that apply to an individual workload or application, the same scope it evaluates during design and code reviews. Content that describes how to build and operate a secure workload generates requirements. Organization-wide governance topics, such as multi-account strategy, root user management, and account-level logging setup, are outside this scope and do not generate workload requirements.

Describe the outcome, not a single implementation

State the security outcome you expect rather than a single prescribed product or configuration. AWS Security Agent generates requirements that focus on the security capability needed and allow more than one valid implementation. For example, "data at rest is encrypted" generates a clearer, more broadly applicable requirement than a statement that names one specific service or setting.

Be specific about what good looks like

The more concretely your documents describe expected behavior, the better AWS Security Agent can define compliance criteria. Where you can, describe what a compliant design demonstrates and what indicates a violation. Vague or purely aspirational statements generate fewer and weaker requirements than concrete, workload-relevant ones.

Review what you get back

Each generated requirement includes a name, applicability, compliance criteria, and remediation guidance that AWS Security Agent derived from your documents. Review the generated requirements, edit any that need refinement, and enable the ones you want AWS Security Agent to evaluate. For more information, see Manage security requirements.

Manage users and access

Control who can access each Agent Space, and configure the IAM roles and encryption keys AWS Security Agent uses.

Grant users access to the AWS Security Agent web application

AWS Security Agent provides two methods for users to access the web application, depending on how you configured your Agent Space during setup.

Access methods overview

IAM Identity Center (SSO) - If you enabled IAM Identity Center when creating your Agent Space, assign individual users or IAM Identity Center groups to the Agent Space in the AWS Security Agent console. Group assignments let you manage access for teams without assigning each user individually. Users sign in to the web application with their Identity Center credentials.

Admin Access - Users with AWS Console access can launch the web application through an admin access link on the Agent Space overview page in the AWS Management Console.

Grant access with IAM Identity Center (SSO)

If you configured your Agent Space with IAM Identity Center, you can assign individual users, IAM Identity Center groups, or both to the Agent Space using the AWS Security Agent console.

Assign users through the AWS Security Agent console

1. In the AWS Security Agent Management Console, navigate to your Agent Space.
2. Choose the **Web app** tab.
3. In the **Users** table, choose **Add users**.
4. Select existing users from IAM Identity Center or create new users.
5. Confirm the user assignments.

> **Tip**
>
> Users assigned to the Agent Space can access the web application by logging in through IAM Identity Center with their SSO credentials.

Assign groups through the AWS Security Agent console

Assign a group to give its direct members access to the Agent Space without assigning each user individually. Create groups and manage their membership in IAM Identity Center. In AWS Security Agent, you assign existing groups to Agent Spaces. Only direct group members receive access because IAM Identity Center doesn't support nested groups.

To assign groups to an Agent Space:

1. In the AWS Security Agent Management Console, navigate to your Agent Space.
2. Choose the **Web app** tab.
3. In the **Groups** table, choose **Add groups**.
4. In **IAM Identity Center groups**, search for groups by name and select the groups to add.
5. Confirm the group assignments.

The selected groups appear in the **Groups** table. Their direct members can access the Agent Space.

Keep the following in mind when you assign groups:

- You can assign up to 100 groups to an Agent Space. IAM Identity Center also limits the AWS Security Agent application to 100 assigned groups across all of its Agent Spaces.
- You can add groups only from the AWS Security Agent console in the primary Region of your IAM Identity Center instance. Group assignment isn't supported in additional (replica) Regions.
- Removing a user from an assigned group, or deleting the group in IAM Identity Center, removes that user's group-based access. Deleting the group doesn't remove its assignment from the Agent Space. To remove the leftover assignment, delete the row from the **Groups** table.
- For users who are already signed in, changes to group membership can take up to two minutes to apply. Signing in again immediately picks up changes that are already available in IAM Identity Center. If you synchronize groups from an external identity provider, synchronization can add another delay before the changes are available in IAM Identity Center.

Remove a group from an Agent Space

To remove a group from an Agent Space:

1. In the AWS Security Agent Management Console, navigate to your Agent Space.
2. Choose the **Web app** tab.
3. In the **Groups** table, select the groups to remove, and then choose **Delete groups**.
4. Confirm the deletion.

Removing a group takes effect on the next request, including for users who are already signed in. Members of a removed group lose access to the Agent Space unless you assign them individually or through another group.

Access the web application with SSO

After users or groups are assigned to the Agent Space:

1. Users navigate to the web application URL for the Agent Space.

> **Tip**
>
> Find the web application URL on the Agent Space detail page in the AWS Security Agent console. Bookmark this URL for easy access.

2. Users log in using their SSO credentials.
3. After authentication, users can select the Agent Space and begin conducting security assessments.

Grant access with IAM-only access

If you configured your Agent Space with IAM-only access, users with AWS Console access can launch the web application through an admin access link.

Use the admin access link

1. Log into the AWS Security Agent console.
2. Navigate to the Agent Space you want to access.
3. On the Web app tab of the Agent Space landing page choose the **Admin access** button to launch the web application.
4. The web application opens in a new tab with the user automatically authenticated.

> **Note**
>
> The admin access link is only available to users who are already authenticated to the AWS Console with appropriate AWS Security Agent permissions. This method does not require IAM Identity Center configuration.

Next steps

After granting users access to the web application:

- Users can create and manage penetration test configurations and runs
- Users can create and manage design reviews
- Users can view security findings and remediation guidance
- Configure notification preferences for security findings

Create the IAM roles for AWS Security Agent

AWS Security Agent assumes IAM service roles in your account. It uses these roles to call its own APIs on behalf of web application users and to reach the AWS resources that you want it to assess. If you schedule penetration tests, Amazon EventBridge Scheduler also assumes a role in your account to start the scheduled runs. There are three roles:

- **Application role** – One per application. AWS Security Agent assumes this role to give web application users permission to call AWS Security Agent APIs.
- **Agent Space service role** – One per Agent Space. AWS Security Agent assumes this role to reach the AWS resources you attach to that Agent Space, such as VPCs, secrets, S3 buckets, Lambda functions, and CloudWatch log groups.
- **Scheduler execution role** – One per Agent Space. EventBridge Scheduler assumes this role to start scheduled penetration test runs in that Agent Space. This role exists only if you enable penetration test scheduling for the Agent Space.

The console creates these roles for you, and generates their permissions from the resources you select. You do not need to write these policies by hand. This topic describes what the console creates, so that you can review and audit it. In the policies that follow, replace `111122223333` with your account ID, `us-east-1` with your Region, and the resource names with your own.

A penetration test can also assume a role in your account to authenticate to your target application. That role is separate from the application role and the Agent Space service role. You configure it per penetration test, as a credential, and you can use an IAM role, an AWS Secrets Manager secret, or a Lambda function. For more information, see the section called “Provide authentication credentials for penetration testing”.

Application role

The application role is created once for your application, when you enable AWS Security Agent. AWS Security Agent assumes it to grant web application users the permissions they need to call AWS Security Agent APIs, including for IAM Identity Center and admin access link sign-in.

Trust policy

The trust policy allows the AWS Security Agent service principal to assume the role. It also lets the service set the IAM Identity Center identity context on the resulting session. Both statements are required. Without `sts:SetContext`, IAM Identity Center sign-in to the web application fails. The `aws:SourceAccount` and `aws:SourceArn` conditions restrict the role to calls that AWS Security Agent makes on behalf of resources in your own account. For more information, see the section called “Cross-service confused deputy prevention”.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowServiceAssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:securityagent:us-east-1:111122223333:application/*",
            "arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
          ]
        }
      }
    },
    {
      "Sid": "AllowServiceSetContext",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
```

```
      },
      "Action": "sts:SetContext",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ForAllValues:ArnEquals": {
          "sts:RequestContextProviders": "arn:aws:iam::aws:contextProvider/
IdentityCenter"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:securityagent:us-east-1:111122223333:application/*",
            "arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
          ]
        }
      }
    }
  ]
}
```

Permissions policy

The application role carries two sets of permissions, and it needs both. The first is the AWS managed policy `AWSSecurityAgentWebAppPolicy`, which grants the AWS Security Agent API permissions that web application users need. Use the managed policy rather than writing an equivalent policy yourself, so that your role picks up new permissions as AWS Security Agent adds capabilities. For its contents, see the section called “AWS managed policies”. The second is a customer managed policy with the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogsRead",
      "Effect": "Allow",
      "Action": "logs:GetLogEvents",
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:*:log-stream:*"
    },
    {
      "Sid": "SecretsManagerCreate",
      "Effect": "Allow",
      "Action": "secretsmanager:CreateSecret",
```

```
      "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:*"
    }
  ]
}
```

If you encrypt your Agent Space with a customer managed key, the application role needs additional AWS KMS permissions. For those statements, see the section called “Customer managed keys”.

Permissions for penetration test scheduling

To schedule penetration tests in the web application, you need permissions in Amazon EventBridge Scheduler and IAM. `AWSSecurityAgentWebAppPolicy` grants these permissions, so a role that has the managed policy attached receives them automatically. To add these permissions to your own policy instead, see the section called “AWS managed policy: AWSSecurityAgentWebAppPolicy”.

Agent Space service role

Each Agent Space has one service role. AWS Security Agent assumes it to reach the AWS resources you attached to that Agent Space while it runs an assessment. The role is shared by every capability in the Agent Space, including penetration testing, code review, and threat modeling, so changing its permissions affects all of them.

Trust policy

This role does not use an external ID. The `aws:SourceAccount` and `aws:SourceArn` conditions serve the same purpose for an AWS service principal. For more information, see the section called “Cross-service confused deputy prevention”.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSecurityTestingAgentSpaceAssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
```

```
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
        }
      }
    }
  ]
}
```

Permissions policies

The console attaches a base policy to this role, plus one additional policy for each type of resource you attach to the Agent Space. If you don't attach a resource type, the role receives no permissions for it. The following table lists the policies and when each one is created.

Policy name	Created when	Grants
roleName-SecurityAgentBasePolicy	Always	Read its own role, read the secrets AWS Security Agent creates for the Agent Space, and write assessment logs
roleName-VpcPolicy	You attach a VPC with subnets or security groups	Create and manage elastic network interfaces, and describe VPC configuration
roleName-StaticCloudWatchPolicy	You attach CloudWatch log groups	Write to the log groups you selected
roleName-SecretsPolicy	You attach Secrets Manager secrets	Read the secrets you selected
roleName-LambdaPolicy	You attach Lambda functions	Invoke the functions you selected
roleName-S3Policy	You attach S3 buckets	Read the buckets you selected

If a resource you attach is encrypted with a customer managed key, the corresponding policy also grants AWS KMS permissions scoped to that key. For more information, see the section called "Customer managed keys".

In the policies that follow, `my-agent-space` is the name of your Agent Space and `my-service-role` is the name of the role.

The base policy, which the console always creates:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:SimulatePrincipalPolicy"
      ],
      "Resource": "arn:aws:iam::111122223333:role/service-role/my-service-role"
    },
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-agent-space*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "logs:CreateLogGroup",
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:/aws/securityagent/my-
agent-space*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
```

```
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:/aws/securityagent/my-agent-
space*:log-stream:my-agent-space*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

The VPC policy, which lets AWS Security Agent attach an elastic network interface to your VPC so that it can reach targets inside it:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeNetworkInterfaceAttribute",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:DeleteNetworkInterface"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ec2:CreateNetworkInterfacePermission",
      "Resource": "arn:aws:ec2:us-east-1:111122223333:network-interface/*",
      "Condition": {
        "ArnEquals": {
          "ec2:Subnet": [
            "arn:aws:ec2:us-east-1:111122223333:subnet/subnet-1234567890abcdef0"
```

```
          ]
        }
      }
    }
  ]
}
```

The CloudWatch Logs policy, scoped to the log groups you selected, with the Agent Space name as the log stream prefix:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:CreateLogGroup",
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group:log-stream:my-
agent-space/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

The Secrets Manager policy, scoped to the secrets you selected:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret-AbCdEf"
      ]
    }
  ]
}
```

The Lambda policy, scoped to the functions you selected, such as a function that resolves penetration test credentials:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": [
        "arn:aws:lambda:us-east-1:111122223333:function:my-function"
      ]
    }
  ]
}
```

The Amazon S3 policy, scoped to the buckets you selected, such as a bucket that holds source code for a code review:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
```

```
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

Scheduler execution role

Amazon EventBridge Scheduler assumes this role to start scheduled penetration test runs, so the runs start on time without anyone signing in. The console creates the role when you enable penetration test scheduling for an Agent Space. For information about how to enable it, see the section called “(Optional) Enable penetration test scheduling”. Each Agent Space has its own role, named `continuum-schedule-`*`agentSpaceId`* , under the `/service-role/` path. The web application finds the role by this name, so don’t rename it.

In the policies that follow, `as-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111` is the ID of your Agent Space.

Trust policy

With the following JSON trust policy, EventBridge Scheduler can assume the role only for schedules in the schedule group for this Agent Space. The schedule group has the same name as the Agent Space ID. For more information, see the section called “Cross-service confused deputy prevention”.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEventBridgeSchedulerAssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": "scheduler.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333",
          "aws:SourceArn": "arn:aws:scheduler:us-east-1:111122223333:schedule-group/as-
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        }
      }
    }
  ]
}
```

Permissions policy

The console attaches the following JSON customer managed policy, named `continuum-schedule-`*`agentSpaceId`*`-SchedulerPolicy-`*`suffix`* . With this policy, the role can start jobs only in this Agent Space. The policy also includes the actions to start code review and threat model jobs and to create design reviews, but scheduled penetration test runs use only `securityagent:StartPentestJob`. For more information about these actions, see the AWS Security Agent API Reference.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "TriggerSecurityAgentJobsForAgentSpace",
      "Effect": "Allow",
      "Action": [
        "securityagent:StartPentestJob",
        "securityagent:StartCodeReviewJob",
        "securityagent:StartThreatModelJob",
        "securityagent:CreateDesignReview"
      ],
      "Resource": "arn:aws:securityagent:us-east-1:111122223333:agent-space/as-
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
  ]
}
```

If you encrypt the Agent Space with a customer managed key, the console adds the following JSON statement to the policy. With this statement, the role can use the key only through AWS Security Agent, and only for data in this Agent Space.

```
{
  "Sid": "AllowKmsKeyDecryptionForAgentSpaceData",
  "Effect": "Allow",
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "Resource": "arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-
EXAMPLE11111",
  "Condition": {
    "StringEquals": {
      "aws:ResourceAccount": "111122223333"
    },
    "StringLike": {
      "kms:ViaService": "securityagent.us-east-1.amazonaws.com",
      "kms:EncryptionContext:aws:securityagent:agent-space": "arn:aws:securityagent:us-
east-1:111122223333:agent-space/as-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
  }
}
```

Customer managed keys for AWS Security Agent

By default, AWS Security Agent encrypts all data at rest using AWS-managed encryption keys. You can optionally use a customer managed key from AWS Key Management Service (AWS KMS) to encrypt your data, giving you full control over the encryption keys that protect your resources.

AWS Security Agent supports resource-level customer managed keys. When you create a top-level resource such as an Agent Space or integration, you can specify a KMS key to encrypt all data belonging to that resource and its subresources. For example, specifying a customer managed key when creating an Agent Space encrypts all data associated with that Agent Space, including Agent Space configurations, penetration test configurations, jobs, and execution details, security findings, discovered endpoints, and screenshots. You can also provide AWS resources that are already encrypted with your own customer managed key, such as S3 buckets, CloudWatch Logs log groups, or Secrets Manager secrets. For the required KMS permissions, see the section called “Required KMS permissions”.

How customer managed keys work

AWS Security Agent uses the AWS KMS Hierarchical keyring to encrypt your data with customer managed keys. When you specify a KMS key during resource creation, the service creates a branch

key protected by your KMS key and stores it in an Amazon DynamoDB-based key store. For each encryption operation, the Hierarchical keyring derives a unique wrapping key from the active branch key, which encrypts a unique data encryption key for each request.

The Hierarchical keyring provides the following benefits:

- **Per-resource encryption scope** – Each top-level resource has its own branch key, so data for different resources is encrypted under separate keys.
- **Automatic key rotation** – AWS Security Agent periodically rotates branch keys. After rotation, new data is encrypted with the new branch key version, while older versions are retained to decrypt previously encrypted data.

Encryption context

AWS Security Agent uses encryption context in all cryptographic operations with your KMS key. The encryption context is a set of non-secret key-value pairs that provides additional authenticated data for the encryption operation.

The encryption context key follows the format `aws:securityagent:_<resource-type>_`, where *`<resource-type>`* is the type of resource being encrypted (for example, `agent-space` or `integration`). The value is the Amazon Resource Name (ARN) of the resource.

> **Note**
>
> For integrations, the encryption context key includes the `aws-crypto-ec:` prefix, resulting in the format `aws-crypto-ec:aws:securityagent:integration`.

You can use the encryption context to scope down your KMS key policy to specific resource types. For more information, see the section called “Required KMS permissions”.

Default encryption

When you create a resource without specifying a customer managed key, AWS Security Agent encrypts the resource using AWS-managed encryption keys provided by the underlying storage services (Amazon DynamoDB and Amazon S3). No additional configuration is required for default encryption.

Default KMS key for applications

You can set a default KMS key at the application level. When a default KMS key is configured, AWS Security Agent uses it as the fallback for new Agent Spaces and integrations when you don't explicitly specify a KMS key during resource creation.

To set a default KMS key, specify the `defaultKmsKeyId` parameter when creating or updating your application using the AWS CLI or SDK. The console prompts you to specify a default KMS key during application setup.

When you explicitly specify a KMS key during resource creation, it takes precedence over the application-level default.

Prerequisites

Before you configure a customer managed key, complete the following prerequisites:

- Create a symmetric encryption KMS key in AWS KMS. The key must meet the following requirements:
  - Key type: Symmetric
  - Key usage: Encrypt and decrypt
  - Key spec: `SYMMETRIC_DEFAULT`
  - Key state: Enabled

  For instructions, see Creating keys in the *AWS Key Management Service Developer Guide*.
- Configure the KMS key policy and IAM policies to grant AWS Security Agent the required permissions. For details, see the section called "Required KMS permissions".

Required KMS permissions

AWS Security Agent requires KMS permissions in two scenarios:

- **Encrypting data within AWS Security Agent** – When you specify a customer managed key for an Agent Space or integration, the service needs permissions to use that key for cryptographic operations on your data.
- **Using CMK-encrypted AWS resources** – When you provide AWS resources encrypted with a customer managed key (such as S3 buckets, CloudWatch Logs log groups, or Secrets Manager secrets), the service needs permissions to use those keys to access the encrypted resources.

AWS Security Agent accesses your KMS key using different identities depending on the operation:

- **AWS Management Console operations** – When administrators create or manage resources in the AWS Management Console (for example, creating an Agent Space or updating an application), the service uses the administrator role to call AWS KMS.
- **Web application operations** – When IAM Identity Center users access data through the AWS Security Agent web application (for example, viewing penetration test results or starting a penetration test), the service uses the application role created during AWS Security Agent setup to call AWS KMS.
- **Accessing customer-provided resources** – When the service accesses customer-provided AWS resources during penetration testing (such as S3 buckets, CloudWatch Logs log groups, and Secrets Manager secrets), it uses the penetration test service role to call AWS KMS.
- **Asynchronous workflows** – For background operations that run outside of any user session (for example, penetration testing execution, code review processing, and branch key rotation), the service uses its own service principal (`securityagent.amazonaws.com`) to call AWS KMS on your behalf.

The following sections describe the required permissions for each identity.

Key policy

Your KMS key policy must grant AWS Security Agent permission to use the key for cryptographic operations. The required key policy statements depend on which resource types you plan to encrypt and which CMK-encrypted AWS resources you provide to the service.

Key policy for Agent Spaces

The following key policy grants AWS Security Agent the permissions required to encrypt and decrypt Agent Space data, including penetration test results and screenshots.

Replace the following placeholder values in the policy:

- *111122223333* – Your AWS account ID
- *MyRole* – The IAM role you use to manage AWS Security Agent in the console
- *MyApplicationRole* – The application role created during AWS Security Agent setup
- *us-east-1* – The AWS Region where you use AWS Security Agent

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowKeyMetadataValidationForApplicationAndAgentSpaces",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyRole"
      },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowUseOfHierarchicalKeyringForAgentSpaces",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyRole"
      },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws:securityagent:agent-space":
 "arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
        },
        "StringEquals": {
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        }
      }
    },
    {
```

```
      "Sid": "AllowSynchronousDataAccessForAgentSpaces",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyApplicationRole"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws:securityagent:agent-space":
"arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
        },
        "StringEquals": {
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowAsynchronousDataAccessForAgentSpaces",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws:securityagent:agent-space":
"arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
        }
      }
    },
    {
```

```
      "Sid": "AllowAsynchronousDataAccessForCodeRemediation",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "*"
    }
  ]
}
```

Important

To rotate branch keys, your KMS key policy must grant `kms:GenerateDataKeyWithoutPlaintext`, `kms:ReEncryptTo`, and `kms:ReEncryptFrom` permissions to the AWS Security Agent service principal (`securityagent.amazonaws.com`), or branch key rotation will fail. For more information about the required permissions, see Rotate a branch key.

Key policy for integrations

The following key policy grants AWS Security Agent the permissions required to encrypt and decrypt integration data, including code review findings.

Replace the following placeholder values in the policy:

- *111122223333* – Your AWS account ID
- *MyRole* – The IAM role you use to manage AWS Security Agent in the console
- *us-east-1* – The AWS Region where you use AWS Security Agent

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowKeyAccessValidationForIntegrations",
      "Effect": "Allow",
```

```
      "Principal": {
          "AWS": "arn:aws:iam::111122223333:role/MyRole"
      },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws-crypto-ec:aws:securityagent:integration":
"arn:aws:securityagent:us-east-1:111122223333:integration/*"
        },
        "StringEquals": {
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowKeyMetadataValidationForIntegrations",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": "kms:DescribeKey",
      "Resource": "*"
    },
    {
      "Sid": "AllowAsynchronousDataAccessForIntegrations",
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom"
      ],
      "Resource": "*",
```

```
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:securityagent:us-east-1:111122223333:integration/*"
        },
        "StringLike": {
          "kms:EncryptionContext:aws-crypto-ec:aws:securityagent:integration":
 "arn:aws:securityagent:us-east-1:111122223333:integration/*"
        }
      }
    }
  ]
}
```

> **Note**
>
> You can combine the Agent Space, integration, and security requirement pack key policy statements into a single key policy if you want to use the same KMS key for multiple resource types.

Key policy for security requirement packs

The following key policy grants AWS Security Agent the permissions required to encrypt and decrypt security requirement pack data. Security requirement packs do not use a web application role. The policy uses two access paths:

- **Synchronous path** – When you call the API, the service uses your forwarded credentials to call AWS KMS on your behalf (via the `kms:ViaService` condition).
- **Asynchronous path** – For asynchronous operations where the packs can be used, the service uses its own service principal to call AWS KMS directly.

Replace the following values in the policy:

- *111122223333* – Your AWS account ID
- *MyRole* – The IAM role you use to invoke security requirement pack operations
- *us-east-1* – The AWS Region where you use AWS Security Agent

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowKeyMetadataValidation",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyRole"
      },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowUseOfHierarchicalKeyringForSecurityPacks",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyRole"
      },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        },
        "StringLike": {
          "kms:EncryptionContext:aws:securityagent:security-requirement-pack":
  "arn:aws:securityagent:us-east-1:111122223333:security-requirement-pack/*"
        }
      }
    },
    {
      "Sid": "AllowAsynchronousDataAccessForSecurityPacks",
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws:securityagent:security-requirement-pack":
 "arn:aws:securityagent:us-east-1:111122223333:security-requirement-pack/*"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:securityagent:us-east-1:111122223333:security-
requirement-pack/*"
        }
      }
    }
  ]
}
```

The policy contains three statements:

- **AllowKeyMetadataValidation** – Grants `kms:DescribeKey` so that AWS Security Agent can validate that your customer managed key exists, is enabled, and uses symmetric encryption when you specify a CMK during resource creation. Uses the `kms:ViaService` condition to ensure the call originates through the service.
- **AllowUseOfHierarchicalKeyringForSecurityPacks** – Grants `kms:GenerateDataKeyWithoutPlaintext` (create new branch keys), `kms:ReEncryptTo` and `kms:ReEncryptFrom` (rotate existing branch keys), and `kms:Decrypt` (retrieve existing branch keys) for hierarchical keyring operations. These operations use your forwarded credentials through the synchronous path and are scoped to security requirement pack resources by the encryption context condition.
- **AllowAsynchronousDataAccessForSecurityPacks** – Grants `kms:GenerateDataKeyWithoutPlaintext`, `kms:Decrypt`, `kms:ReEncryptTo`, and `kms:ReEncryptFrom` to the AWS Security Agent service principal for asynchronous operations when no caller credentials are available.

Unlike the Agent Spaces key policy, the security requirement pack policy does not include a web application role principal. Security requirement packs are accessed through the AWS Management Console using your administrator role, not through the AWS Security Agent web application. All synchronous operations use the single caller role (via `kms:ViaService`).

> **Note**
>
> If you use the same KMS key for both Agent Spaces and security requirement packs, you can combine the key policy statements from both sections into a single key policy.

Key policy for CMK-encrypted AWS resources

If you provide AWS resources encrypted with a customer managed key to AWS Security Agent, you must update the key policy on each resource's KMS key to grant the necessary permissions. This applies to the following resources:

- **S3 buckets** – If you provide learning resources from an S3 bucket encrypted with a customer managed key (SSE-KMS), the key policy must allow the penetration test service role to decrypt objects.
- **Secrets Manager secrets** – If your penetration test credentials are stored in Secrets Manager secrets encrypted with a customer managed key, the key policy must allow the penetration test service role to decrypt those secrets. If the web application creates secrets on your behalf, the key policy must also allow the application role to encrypt those secrets.
- **CloudWatch Logs log groups** – If the CloudWatch Logs log group used for storing penetration test execution logs is encrypted with a customer managed key, the key policy must allow CloudWatch Logs to validate the KMS key through the service role and allow the CloudWatch Logs service principal to perform cryptographic operations.

> **Important**
>
> AWS Security Agent may also create CloudWatch Logs log groups and Secrets Manager secrets on your behalf. When configuring your KMS key policy, include the corresponding statements for these service-created resources as well.

The following key policy statements grant the required permissions for CMK-encrypted resources. Include only the statements that apply to your configuration. If a resource uses the same KMS key you specified for your Agent Space, add these statements to that key's policy. If a resource uses a different KMS key, add these statements to that key's policy instead.

Replace the following placeholder values in the policy:

- *111122223333* – Your AWS account ID
- *MyApplicationRole* – The application role created during AWS Security Agent setup
- *MyPenTestServiceRole* – The penetration test service role
- *us-east-1* – The AWS Region where you use AWS Security Agent

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowKmsKeyAccessForCreatingSecrets",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyApplicationRole"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "secretsmanager.*.amazonaws.com",
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:*"
        }
      }
    },
    {
      "Sid": "AllowKmsKeyDecryptionForS3Objects",
      "Effect": "Allow",
      "Principal": {
```

```
        "AWS": "arn:aws:iam::111122223333:role/MyPenTestServiceRole"
      },
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "s3.*.amazonaws.com",
          "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::YOUR-BUCKET-NAME*"
        }
      }
    },
    {
      "Sid": "AllowKmsKeyDecryptionForSecretsManagerSecrets",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyPenTestServiceRole"
      },
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "secretsmanager.*.amazonaws.com",
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:YOUR-SECRET-NAME*"
        }
      }
    },
    {
      "Sid": "AllowKmsKeyValidationForCloudWatchLogsLogGroups",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/MyPenTestServiceRole"
      },
      "Action": [
```

```
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "logs.*.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowKmsKeyAccessForCloudWatchLogsLogGroups",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-
east-1:111122223333:log-group:YOUR-LOG-GROUP-NAME*"
        }
      }
    }
  ]
}
```

Note

- The `AllowKmsKeyDecryptionForS3Objects` statement is required only if you provide learning resources from S3 buckets encrypted with a customer managed key. For more information about configuring SSE-KMS for S3, see Protecting data with SSE-KMS.

- The `AllowKmsKeyDecryptionForSecretsManagerSecrets` statement is required only if your penetration test credentials are stored in Secrets Manager secrets encrypted with a customer managed key. The service role needs access to decrypt secrets during penetration testing. For more information about secret encryption, see Secret encryption and decryption in Secrets Manager.
- The `AllowKmsKeyValidationForCloudWatchLogsLogGroups` statement grants the service role `kms:DescribeKey` so that CloudWatch Logs can validate the KMS key through the service role when associating it with a log group.
- The `AllowKmsKeyAccessForCreatingSecrets` statement is required if the web application creates Secrets Manager secrets on your behalf using a customer managed key. The application role needs `kms:GenerateDataKey` and `kms:Decrypt` to encrypt the secret with your KMS key.
- The `AllowKmsKeyAccessForCloudWatchLogsLogGroups` statement grants the CloudWatch Logs service principal (`logs.`*`us-east-1`*`.amazonaws.com`) the permissions it requires to encrypt and decrypt log data. This statement is also required if AWS Security Agent creates a log group on your behalf when you don't provide an existing one. For more information, see Encrypt log data in CloudWatch Logs using AWS KMS.
- If multiple resources share the same KMS key, you can combine the statements into a single key policy.

Administrator role

No additional IAM policy is required for the administrator role (the IAM role you use to manage AWS Security Agent in the console).

Application role

In addition to the KMS key policy, attach the following IAM policy to the application role specified during AWS Security Agent setup. This policy grants the role permissions to use your customer managed keys for encrypting and decrypting Agent Space data and creating secrets in AWS Secrets Manager.

Replace the following placeholder values in the policy:

- *111122223333* – Your AWS account ID

- *us-east-1* – The AWS Region where you use AWS Security Agent
- The KMS key ARNs in Resource – The ARNs of your KMS keys

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSynchronousDataAccessForAgentSpaces",
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890cd"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333",
          "kms:ViaService": "securityagent.us-east-1.amazonaws.com"
        },
        "StringLike": {
          "kms:EncryptionContext:aws:securityagent:agent-space":
 "arn:aws:securityagent:us-east-1:111122223333:agent-space/*"
        }
      }
    },
    {
      "Sid": "AllowKmsKeyAccessForCreatingSecrets",
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890cd"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
```

```
        },
        "StringLike": {
          "kms:ViaService": "secretsmanager.*.amazonaws.com",
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:*"
        }
      }
    }
  ]
}
```

Note

The application role is an IAM role that you specify or that the console creates during AWS Security Agent setup. The web application assumes this role to retrieve penetration test execution logs and create Secrets Manager secrets on your behalf.

- The `AllowKmsKeyAccessForCreatingSecrets` statement is required if you configure authentication resources for penetration tests and choose to enter credentials directly instead of specifying an existing secret. The web application creates a secret on your behalf, and the application role needs the permissions in this statement to encrypt the secret with the customer managed key specified for your Agent Space. Update the `Resource` ARNs in this statement to match the KMS key used for your Agent Space.

Service role

During penetration testing, AWS Security Agent assumes the penetration test service role to access your AWS resources. If any of these resources are encrypted with a customer managed key, you must grant the service role permissions to use the corresponding KMS keys. This applies to the following resources:

- **S3 buckets** – If you provide learning resources (such as API documents, threat models, or source code) from an S3 bucket encrypted with a customer managed key, the service role needs permissions to decrypt objects in that bucket. For more information about configuring SSE-KMS for S3, see Protecting data with SSE-KMS.
- **Secrets Manager secrets** – If your penetration test credentials are stored in Secrets Manager secrets encrypted with a customer managed key, the service role needs permissions to decrypt

those secrets. For more information about secret encryption, see Secret encryption and decryption in Secrets Manager.

- **CloudWatch Logs log groups** – If the CloudWatch Logs log group used for storing penetration test execution logs is encrypted with a customer managed key (including log groups created by AWS Security Agent on your behalf), the service role needs `kms:DescribeKey` permission to validate the key. The CloudWatch Logs service principal handles the actual encryption and decryption of log data, and those permissions are granted through the key policy (see the section called "Key policy"). For more information about encrypting log data, see Encrypt log data in CloudWatch Logs using AWS KMS.

> **Important**
>
> If you use a different KMS key for encrypting these resources than the one you specified for your Agent Space, you must grant the service role permissions for each KMS key that protects a resource the service role accesses during penetration testing.

Attach the following IAM policy to the penetration test service role. Include only the statements that apply to your configuration.

Replace the following placeholder values in the policy:

- *111122223333* – Your AWS account ID
- *us-east-1* – The AWS Region where you use AWS Security Agent
- The KMS key ARNs in `Resource` – The ARNs of the customer managed keys used to encrypt each resource

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowKmsAccessForEncryptedS3Buckets",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
```

```
        "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE-S3-KEY-ID"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "s3.*.amazonaws.com",
          "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::YOUR-BUCKET-NAME*"
        }
      }
    },
    {
      "Sid": "AllowKmsAccessForEncryptedSecrets",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE-SECRETS-KEY-ID"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "secretsmanager.*.amazonaws.com",
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:YOUR-SECRET-NAME*"
        }
      }
    },
    {
      "Sid": "AllowKmsKeyValidationForCloudWatchLogsLogGroups",
      "Effect": "Allow",
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:111122223333:key/EXAMPLE-CLOUDWATCH-LOGS-KEY-ID"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
```

```
        },
        "StringLike": {
          "kms:ViaService": "logs.*.amazonaws.com"
        }
      }
    }
  ]
}
```

Note

- The `AllowKmsAccessForEncryptedS3Buckets` statement is required only if you provide learning resources from S3 buckets encrypted with a customer managed key. Update the `Resource` ARN to match the KMS key used to encrypt your S3 bucket, and update the `kms:EncryptionContext:aws:s3:arn` value to match your bucket name.
- The `AllowKmsAccessForEncryptedSecrets` statement is required only if your penetration test credentials are stored in Secrets Manager secrets encrypted with a customer managed key. Update the `Resource` ARN to match the KMS key used to encrypt your secrets, and update the `kms:EncryptionContext:SecretARN` value to match your secret name.
- The `AllowKmsKeyValidationForCloudWatchLogsLogGroups` statement is required only if your CloudWatch Logs log group is encrypted with a customer managed key, including log groups created by AWS Security Agent on your behalf. Update the `Resource` ARN to match the KMS key used to encrypt your log group.
- If multiple resources share the same KMS key, you can combine the statements and list the shared key ARN once in the `Resource` field.

Create a resource with a customer managed key

You can specify a customer managed key when setting up AWS Security Agent or when creating individual resources. The KMS key encrypts all data belonging to that resource and its subresources.

Set a default KMS key during setup (console)

You can configure a default KMS key during the initial AWS Security Agent setup. This key is used as the default for new Agent Spaces and integrations unless you specify a different key.

1. On the **Set up AWS Security Agent** page, expand the **Encryption - optional** section.
2. Select **Customize encryption settings (advanced)**.
3. For **Choose an AWS KMS key**, select a KMS key from your account, or enter a KMS key ARN.
4. Complete the remaining setup steps and choose **Set up AWS Security Agent**.

AWS Security Agent validates the KMS key to confirm that it exists, is enabled, and uses symmetric encryption. If validation fails, setup does not proceed and you receive an error message.

Create an Agent Space with a customer managed key (console)

1. In the AWS Security Agent console, navigate to the **Agent Spaces** page.
2. Choose **Create Agent Space**.
3. Enter a name and optional description for your Agent Space.
4. Expand the **Advanced** section.
5. Under **Data encryption**, choose one of the following:
   - **Use application default key** – Uses the default KMS key configured during AWS Security Agent setup. This option is selected by default if a default key is configured.
   - **Use a different key** – Specify a different KMS key for this Agent Space. Select **Customize encryption settings (advanced)**, then for **Choose an AWS KMS key**, select a KMS key from your account or enter an ARN.
6. Choose **Create**.

Create an integration with a customer managed key (console)

1. In the AWS Security Agent console, navigate to the **Integrations** page.
2. Choose **Add integration** and select your provider (for example, GitHub).
3. Complete **Step 1: Install and authorize** the provider application.
4. In **Step 2: Register details**, enter a registration name and configure the provider settings.
5. In the **Data encryption** section, select **Customize encryption settings (advanced)**.
6. For **Choose an AWS KMS key**, select a KMS key from your account, or enter a KMS key ARN.
7. Choose **Connect**.

Create a resource with a customer managed key (AWS CLI or SDK)

To specify a customer managed key using the AWS CLI or SDK:

- Include the `defaultKmsKeyId` parameter when calling `CreateApplication` to set a default KMS key for your application. This key is used as the fallback when creating Agent Spaces or integrations without an explicit KMS key.
- Include the `kmsKeyId` parameter when calling `CreateAgentSpace` or `CreateIntegration` to encrypt a specific resource. This takes precedence over the application-level default.

For parameter details, see the AWS Security Agent API Reference.

If you don't specify a KMS key, AWS Security Agent uses the application-level default KMS key if one is configured. Otherwise, the resource is encrypted with AWS-managed keys.

Key rotation

AWS Security Agent automatically rotates branch keys on a periodic basis. Branch key rotation creates a new active version of the branch key while retaining all previous versions. After rotation:

- New data is encrypted with the new branch key version.
- Previously encrypted data remains decryptable using the older branch key version.
- No data re-encryption is required.

Branch key rotation is separate from KMS key rotation. You can enable automatic KMS key rotation for your customer managed key independently. For more information, see Rotating KMS keys in the *AWS Key Management Service Developer Guide*.

After a branch key rotation, there is a brief period during which cached copies of the previous branch key may still be used for new encryption operations. This window is determined by the internal cache time-to-live (TTL) and does not affect the security of your data.

Considerations

Keep the following in mind when using customer managed keys with AWS Security Agent:

- **Customer managed keys apply to new resources only.** Existing resources created before you configure a customer managed key continue to use AWS-managed encryption. There is no migration of existing data.

- **Customer managed keys are specified at creation time.** You specify the KMS key when creating a resource. You cannot add or change the KMS key for an existing resource.
- **Multiple KMS keys are supported.** You can use different KMS keys for different resources. For example, you can use one key for production Agent Spaces and a different key for development Agent Spaces.
- **Disabling or deleting a KMS key makes data inaccessible.** If you disable or schedule deletion of a KMS key, AWS Security Agent cannot decrypt data encrypted under that key. Affected resources become inaccessible until the key is re-enabled. Deleting a KMS key permanently prevents access to all data encrypted under that key.
- **Key policy permissions are required.** If you remove the required permissions from your KMS key policy, AWS Security Agent cannot access encrypted data. Ensure that the key policy grants permissions for the administrator role (used to manage the service in the AWS Console), the application role (used by the web application), the penetration test service role (assumed by agents to run penetration tests), and the AWS Security Agent service principal as described in the section called “Required KMS permissions”.
- **AWS KMS quotas apply.** Cryptographic operations against your KMS key count toward your AWS KMS request quotas. Under normal usage, the hierarchical keyring architecture minimizes KMS calls through branch key caching. For more information, see Request quotas in the *AWS Key Management Service Developer Guide*.

Troubleshooting

Find solutions to commonly seen errors when using AWS Security Agent.

Access Denied: Incorrect GitHub account type selected or incorrect organization name specified

- You installed the AWS Security Agent application into your desired GitHub organization but incorrectly set the `GitHub Account Type` to `User` instead of `Organization`
- You installed the AWS Security Agent application into your desired GitHub organization and correctly set the `GitHub Account Type` to `Organization` but left the `Organization Name` field blank or entered an incorrect organization name that does not match the organization you installed the application into

Solution:

1. Go to GitHub and uninstall the app from the organization. For more information, see the section called “Step 1: Uninstall the AWS Security Agent GitHub App from GitHub”.
2. Go back to the integrations page and restart the integration process by clicking on `Add Integrations`, install and authorize the app into your desired GitHub organization once again.
3. Select `Organization` from the dropdown of `GitHub account type`
4. Make sure the `Organization Name` you input is the EXACT same as the one you installed the application into.
5. Choose **Connect** to create your GitHub organization integration.

Access Denied: Insufficient permissions to install GitHub App into organization

When you attempt to install the AWS Security Agent application into your desired GitHub organization, you will see two different messages on the button in the installation page.

- An organization `Member` will see `Authorize & Request`
- An organization `Owner` will see `Install & Authorize`

You can verify whether you are a `Member` or an `Owner` of the GitHub organization by following the below steps.

1. Go to github.com
2. Choose your profile on the website
3. Navigate to `Organizations` on the dropdown menu and choose it
4. Find the organization you wish to install AWS Security Agent into from the list of organizations, it will specify whether you are a `Member` or an `Owner` next to the organization name.

To verify the installation, AWS Security Agent lists the apps installed on the organization. It uses the GitHub account that authorized the app. GitHub allows only organization owners to list installed apps. Registration fails for a member even after an owner approves the member’s installation request.

Possible solutions:

- Have an organization owner complete the whole integration process: install the app, authorize it, and register it in the AWS Console
- Have an owner update your role in the GitHub organization from a `Member` to an `Owner` and restart the integration process again

Agent cannot connect to endpoint during a penetration test

If the penetration test agent is unable to make calls to the configured target URL or fails to successfully navigate the target endpoint:

- If your endpoint makes calls to domains outside the configured target URL, verify the additional domains are added as **Accessible URLs** in your penetration test configuration
- If your application serves traffic on a non-standard port, include the port in the target URL (for example, `https://example.com:8443`)
- If you have a WAF configured, check that you WAF is not blocking penetration test traffic. You can allowlist penetration test traffic by `User-Agent` header, which will be set to `securityagent` by default

Endpoint validation failed during a penetration test

A penetration test failing during endpoint validation indicates that your endpoint is not accessible from the penetration testing environment. The steps for debugging this issue will depend on if you have configured a VPC for your penetration test:

- For a non-VPC penetration test:
  - Ensure the target endpoint has a public DNS record that resolves to a publicly accessible IP address
  - Verify that your target endpoint can be reached without special networking requirements. Some common blockers are a specific VPN required to reach the endpoint or a WAF that blocks public internet traffic
  - Attempt to reach the target endpoint using `curl` or a browser and verify you receive a valid response
- For a VPC penetration test:

- Ensure the selected VPC/subnet has proper routing to the target endpoint (e.g., via VPN tunnel or VPC peering), DNS resolution is correctly configured for the private domain (e.g., Route 53 Resolver or custom DNS), and security group rules allow outbound traffic to the target
- To debug further, launch an EC2 instance in your account with the same VPC, subnet, and security group as your penetration test. Connect to the EC2 instance and verify that networking requests to your target endpoint succeed

Penetration test failed due to an IP allowlist

If the endpoint that a penetration test targets has an IP allowlist enabled, you can add a private VPC configuration with an associated VPC NAT Gateway to your penetration test. You can then use the NAT Gateway IP address to allowlist outbound traffic from the penetration test.

Penetration test login blocked by an identity provider

A penetration test might report a login failure for an application that delegates sign-in to a third-party identity provider. The provider might have blocked the agent's browser instead of rejecting the credentials. Providers that apply anti-automation controls can refuse an automated browser before they evaluate the credentials. They usually report that the browser or app is not secure.

Solution:

Provide a credential that the agent can replay, rather than one it has to earn by completing the provider's sign-in flow. For longer tests, prefer an AWS Lambda function, because it can return a fresh session for each run.

- Sign in yourself and capture the resulting session cookie or bearer token.
- Store it in AWS Secrets Manager, or return it from an AWS Lambda function. For more information, see the section called "Select static credential from connected AWS Secrets Manager".
- Select it as the credential for the penetration test.
- Use the **Agent Space login prompt** to state how to apply it, for example which header to set on each request.
- Keep the session valid for the duration of the test.
- Point the agent at a sign-in path that isn't delegated to the provider, such as a local test account or an API key, if the application offers one.

Getting additional help

If you continue to experience issues after trying these troubleshooting steps:

- Check the AWS Security Agent service status page
- Contact AWS Support for assistance with complex configuration issues

For users and developers (web application and IDE)

Run penetration tests, code reviews, threat models, and design reviews in the web application, an IDE, or a connected source provider, then review and remediate findings.

Log in to the AWS Security Agent web application

You should log in to the AWS Security Agent web application to complete tasks such as:

- Perform a design review
- Perform a penetration test

Access methods

AWS Security Agent provides different methods to access the web application, depending on how your organization configured access during initial setup and whether you have AWS Console access.

IAM Identity Center (SSO) - If your organization enabled IAM Identity Center, you can log in using your SSO credentials. To use an Agent Space in the web application, you must be assigned to it individually or belong to an IAM Identity Center group that is assigned to it.

Admin access (IAM) - If you have AWS Console access with appropriate permissions, you can launch the web application through the admin access link on any Agent Space page. This method provides authentication through your existing Console session.

> **Note**
>
> The primary access method your organization uses was determined during initial AWS Security Agent setup. For information on configuring user access and assignments, see the section called “Grant users access to the AWS Security Agent web application”.

Log in with IAM Identity Center (SSO)

If your organization configured IAM Identity Center access, you can log in to the web application using your SSO credentials through multiple entry points.

Prerequisites

- You are assigned individually to at least one Agent Space, or you belong to an IAM Identity Center group that is assigned to one
- You have your IAM Identity Center SSO credentials

Access the web application

Choose one of the following methods based on your needs:

To view all Agent Spaces you're assigned to:

1. In the AWS Security Agent console, navigate to **Settings**.
2. Locate the **Web application domain** and copy the URL.

> **Tip**
>
> Bookmark this URL for easy access. This is the universal entry point to view all Agent Spaces you're assigned to.

3. Navigate to the web application URL.
4. Enter your IAM Identity Center credentials when prompted.
5. After authentication, you'll see a list of all Agent Spaces you're assigned to.
6. Select the Agent Space you want to work in.

To access a specific Agent Space directly (requires AWS Console access):

1. Log into the AWS Management Console.
2. Navigate to the AWS Security Agent console.
3. Navigate to the Agent Space you want to access.
4. Choose one of the following:
   - **Launch web application** button on the Agent Space overview page
   - **Launch web app** button in the Code review section
   - **Launch web app** button in the Penetration testing section
5. Enter your IAM Identity Center credentials when prompted.

6. After authentication, you'll be taken directly to that Agent Space in the web application.

> **Note**
>
> If you don't have AWS Console access, use the web application domain from the Settings page to access all your assigned Agent Spaces.

Log in with admin access (IAM)

If you have AWS Console access with appropriate AWS Security Agent permissions, you can launch the web application through the admin access link with automatic authentication.

Prerequisites

- You are logged into the AWS Management Console
- You have appropriate permissions to access AWS Security Agent resources

Access the web application

Choose one of the following methods:

To access a specific Agent Space:

1. Log into the AWS Management Console.
2. Navigate to the AWS Security Agent console.
3. Navigate to the Agent Space you want to access.
4. Choose the **Admin access** button on the Agent Space overview page.
5. The web application opens in a new tab with automatic authentication for that Agent Space.

> **Note**
>
> The admin access button is always available to users with AWS Console access, regardless of whether your organization uses IAM Identity Center as the primary access method.

Create a penetration test

Set up automated penetration testing for your web applications by configuring test scope, target domains, and AWS resource access. Penetration tests help identify security vulnerabilities in running applications by simulating real-world attack scenarios against your verified domains.

AWS Security Agent performs comprehensive security testing against your web applications based on configured scope and permissions, providing detailed findings about exploitable vulnerabilities before attackers can discover them.

In this procedure, you'll create a penetration test by configuring test details, defining the test scope, and setting up required permissions.

Prerequisites

Before you begin, ensure you have:

- Access to the AWS Security Agent web application
- At least one verified domain for testing
- IAM role with appropriate permissions for AWS Security Agent
- Understanding of your application's architecture and critical paths

Start creating a penetration test

Navigate to the penetration test creation page in the web application.

1. Log in to the AWS Security Agent web application.
2. Navigate to the **Penetration tests** section.
3. Choose **Create a penetration test**.

> **Tip**
>
> Only verified domains can be included in penetration tests. Ask your admin to verify the domain in AWS management console. See the section called "Enable an application domain for penetration testing".

When you complete the **Credentials** and **Network configuration** steps, AWS Security Agent tests more of your application and stays away from anything you do not want reached. The wizard has five steps:

1. **Penetration test details** – Name the test, choose the test scope, set the target URLs, and choose the service role and log group.
2. (Optional) **VPC Resources** – Configure network access for a target that is not publicly reachable.
3. (Optional) **Credentials** – Add the credentials the agent uses to sign in, and test that each one works.
4. **Network configuration** – Review every domain the test may reach, including the domains discovered while your credentials were signed in.
5. (Optional) **Additional configuration** – Add application context and set run options.

Choose **Save & Next** to move to the next step. Each **Save & Next** saves your progress, so you can leave the wizard and come back later.

A test you started but did not finish appears in the **Penetration tests** list with a **Draft** badge. Choose **Continue setup** on that test to return to the wizard. You cannot start a draft until you finish the wizard and create the test.

Name your penetration test

Provide a descriptive name that helps identify the purpose and scope of this penetration test.

1. In the **Penetration test name** field, enter a descriptive name for your penetration test.

   Example

   The name should clearly identify the application, environment, or component being tested. Maximum 100 characters.

Choose a test scope

Choose the kind of application this penetration test targets. The test scope determines which risks AWS Security Agent tests for.

Choose one of the following in the **Test scope** field:

- **Web application (default)** – A full application penetration test covering web and API risks. AI features are in scope but are not the focus. For more information about the risk types in scope, see web application vulnerability coverage.
- **Gen AI application** – A focused test of large language model (LLM) risks in chatbots, agents, and other generative AI features. AWS Security Agent discovers the application's LLM-backed surfaces and tests them. For more information about the risk types in scope, see Gen AI application vulnerability coverage.

For how the test scope affects risk exclusion, see the section called "Exclude risk types (optional)".

Configure penetration test scope

Define which domains the penetration test is allowed to attack.

In the **Penetration testing scope** section, you enter one or more target URLs. In its **Advanced network access** subsection, you can add custom HTTP headers and certificates.

You classify the other domains the test may reach on the **Network configuration** step, after the agent has tested your credentials. For more information, see the section called "Review network scope".

Add target domains

Specify the verified domains that will be actively tested for security vulnerabilities.

1. In the **Penetration testing scope** section, locate **Target URLs**.
2. Expand the **Verified domains** section to view available domains.
3. In the **Target URL** field, enter a target domain URL.

> **Important**
>
> Only verified domains can be tested. The URL must be under a domain you've previously verified in AWS Security Agent. Sub-domains of a verified domain do not require separate verification.

> **Non-standard ports**
>
> If your application serves traffic on a non-standard port, include the port in the URL. For example, `https://example.com:8443`. Domain verification applies to the host. You don't need to verify the domain again to test a different port.

4. To add multiple target domains:
   a. Choose **Add domain**.
   b. Enter each additional domain URL.
5. To remove a target domain, choose **Remove** next to the domain URL.

> **Tip**
>
> For best results, include all domains that are part of your application's user flow, including subdomains for APIs, authentication services, and content delivery. Sub-domains of a verified parent domain do not require separate verification.

Add custom HTTP headers (optional)

Specify custom HTTP headers that AWS Security Agent adds to every request it makes during penetration tests and credential tests. Credential tests can reach domains outside your target URLs. Do not put a value in a custom header that you would not send to those domains. For more information about credential tests, see Test credentials.

1. Expand **Advanced network access**, then locate the **Custom HTTP headers** section.
2. In the **Custom HTTP headers** input field, enter a header name and value that will be associated with outbound requests.

> **Note**
>
> By default, AWS Security Agent adds a custom header for **User-Agent** set to **securityagent** unless a different custom **User-Agent** header value is specified.

3. To add multiple custom headers:

a. Choose **Add header**.

b. Enter each additional custom header.

4. To remove a custom header, choose **Remove** next to the header.

Add certificates (optional)

Add certificates so that AWS Security Agent can connect to target endpoints that use certificates that aren't publicly trusted, or that require a client certificate. Both certificate types are in the **Advanced network access** subsection.

> **Note**
>
> You must add one or more target URLs before you can attach a client certificate. You choose the certificate's hosts from the target URLs, so until you add them, there are no hosts to choose.

- If a target endpoint's Transport Layer Security (TLS) certificate isn't publicly trusted, add its trust anchor under **Trusted CA certificates**. This applies to self-signed certificates and to certificates issued by a private or internal certificate authority (CA). For more information about trusted CA certificates, see the section called "Provide trusted CA certificates for a penetration test".
- To test a target endpoint that requires mutual TLS (mTLS), attach a client certificate under **Client certificates for mTLS**. For more information about client certificates, see the section called "Provide client certificates for mutual TLS".

Configure IAM Role

Select the pre-configured service role for this penetration test. AWS Security Agent uses an Agent Space-based permission model where administrators configure IAM roles when setting up your Agent Space. You select from roles that are already configured and ready to use.

1. In the **Permissions** section, locate the **Service roles** dropdown.
2. Select the IAM role that grants AWS Security Agent access to required AWS resources.

> **Important**
>
> The selected IAM role must have permissions to access VPC resources, CloudWatch Logs, and any other AWS services needed for the penetration test. Verify that the role has the correct trust relationship with AWS Security Agent.

3. Locate the **CloudWatch log group** dropdown.
4. Select the log group where penetration test logs will be stored. (optional)

> **Note**
>
> The selected CloudWatch log group will store detailed logs of the penetration test execution, including requests made, responses received, and vulnerabilities discovered. If you don't select a log group, a new CloudWatch log group will be automatically created with the `/aws/securityagent` prefix to store the penetration test logs.

Automatic code remediation

Select the **Enable automatic remediation** checkbox.

> **Important**
>
> To remediate security findings in your source code repositories, AWS Security Agent may submit pull requests to your repositories. The pull requests may be visible to all users who have read access to the repositories.

Configure VPC resources (optional)

If your target domains are private and hosted within a VPC, configure the VPC settings where AWS Security Agent should run penetration tests. This step is only necessary for applications that are not publicly accessible.

> **Note**
>
> Skip this step if your target domains are publicly accessible. VPC configuration is only required for testing private applications hosted within an Amazon Virtual Private Cloud.

Choose the VPC, subnets, and security groups for the penetration test environment.

1. In the **VPC** section, locate the **VPC ID** dropdown.
2. Select the VPC where your target domains are hosted.

> **Important**
>
> The selected VPC must contain the target domains you specified in Step 3. Ensure the VPC has appropriate routing and network configuration to allow AWS Security Agent to access your applications.

3. Locate the **Subnets** dropdown.
4. Select one or more subnets where the penetration test should run.

> **Note**
>
> Choose subnets that have network access to your target applications. The penetration test will execute from resources deployed in these subnets.

5. Locate the **Security group** dropdown.
6. Select the security group that controls network access for the penetration test.

> **Important**
>
> The selected security group must allow outbound traffic to your target domains and any accessible domains. Ensure the security group rules permit the necessary network access for comprehensive testing.

Configure authentication credentials (optional)

If your target domains require authentication, provide credentials to allow AWS Security Agent to access protected areas of your application during penetration testing. This step is only necessary for applications that require user authentication.

> **Note**
>
> Skip this step if your target domains do not require authentication or if all areas you want tested are publicly accessible. Configure credentials only when you need AWS Security Agent to test authenticated sections of your application.

Credentials appear in a table on the **Credentials** step. The table lists each credential's name, test status, and authentication type.

> **Important**
>
> When AWS Security Agent signs in with your credentials, it tests everything your application exposes to that signed-in user. Authenticated paths can reach shared services and third-party domains that you are not authorized to test. Only provide credentials for applications you are authorized to test.
> Test against a non-production environment, and do not provide credentials that have access to production systems. For more information, see Use non-production environments for penetration testing.

Add credentials

Provide authentication credentials that AWS Security Agent will use to access your application.

1. On the **Credentials** step, choose **Add credential**.
2. In the **Credential name** field, enter a name that is unique within this penetration test. AWS Security Agent uses this name to report the credential's test results and findings.
3. Select a credential input method:
   - **Input credentials** - Enter your credentials directly into AWS Security Agent.

- **Advanced setting** - For sensitive credential information, use advanced options such as AWS Secrets Manager or AWS Lambda functions. See the section called “Provide authentication credentials for penetration testing” for details.

> **Tip**
>
> For production environments or sensitive credentials, we recommend using the advanced setting option to securely reference credentials stored in AWS Secrets Manager or Systems Manager Parameter Store.

4. For **Input credentials**, enter the **Username** and **Password** for the authenticated account. Use an account whose access matches a typical user rather than an administrator.
5. In the **Access URL** dropdown, select the URL where these credentials will be used. You can select only a URL that is already in **Target URLs**. To use a different URL, add it as a target URL first.
6. Under **Agent Space login prompt**, enter instructions that describe how to sign in with these credentials. A login prompt is required for each credential, because it tells AWS Security Agent how to reach your login, complete it, and confirm that it worked. To start from a worked example, choose one of the buttons below the field, such as **Website login** or **API request**, then edit the inserted text. Choosing an example replaces whatever is in the field.
7. Choose **Save**, or choose **Save and test** to save the credential and immediately test it.

To change a credential later, choose the edit icon in its row. To delete one, choose the remove icon.

Test credentials

Test a credential to confirm that AWS Security Agent can sign in with it before the penetration test runs. Testing also discovers the domains your application reaches while signed in. AWS Security Agent then suggests those domains on the **Network configuration** step.

> **Important**
>
> A credential test runs without the network scope so that it can discover these domains. It can reach any domain, including domains that you later mark **Out of scope**. AWS Security Agent adds your custom HTTP headers to every request.
> If you configured VPC resources, the credential test runs from your subnets. Controls that you apply in your VPC also apply to the credential test, such as security groups and Amazon Route 53 Resolver DNS Firewall.

Testing credentials is optional. To keep all traffic within the network scope, skip credential testing and classify the domains on the **Network configuration** step yourself.

1. In the credential's row, choose **Test**. To test every credential at once, choose **Test all credentials**.
2. Watch the **Status** column. A credential moves through **Starting** and **In progress** to one of these results:
   - **Success** – AWS Security Agent signed in with the credential.
   - **Failed** – Sign-in did not succeed. Choose the status to see the reason.
   - **Timed out** – AWS Security Agent could not complete the sign-in in time. A test that cannot sign in stops after 15 minutes.
   - **Unverified** – The credential has not been tested, or you changed it after its last test.
3. To see what the agent did during a test, choose **View logs** in the credential's row.

Most sign-in tests finish in 2 to 3 minutes. A test keeps running if you leave the **Credentials** step, so you can continue through the wizard while it finishes.

If a credential fails, check that the username and password are correct and that the login prompt describes the steps to reach and complete your login. Then edit the credential and test it again.

A timeout here does not stop you from starting the penetration test. During the penetration test, the agent gets an hour to sign in, four times what it gets for a credential test. A sign-in that was only slow often succeeds with that extra time.

When you leave the **Credentials** step with credentials that have not been tested, AWS Security Agent asks whether to test them. Choose **Test credentials** to test the remaining ones, or **Skip testing** to continue without testing. An untested credential discovers nothing, so any domain reachable only behind it is missing from the suggestions on the **Network configuration** step and you must add it yourself.

Add multiple credentials (optional)

If your application requires multiple sets of credentials or different domains need separate authentication, add additional credential sets.

1. Choose **Add credential** again for each additional credential.
2. Repeat the credential configuration steps for each one.

3. To remove a credential, choose the remove icon in its row.

> **Tip**
>
> Configure multiple credentials when testing different user roles, accessing multiple authenticated domains, or verifying role-based access controls in your application.

Review network scope

On the **Network configuration** step, classify every domain the penetration test may reach. If you tested credentials on the previous step, the domains your application reached while signed in are already listed for you and marked **Suggested**. Review each one, then add anything the tests did not reach.

Every domain has one of three classifications:

- **Target** – A URL the penetration test is allowed to attack. Target URLs come from the **Penetration test details** step and are marked with a **Target** badge. To change one, go back to that step.
- **Accessible** – A URL the penetration test can access but must not attack. Use this for a hosted login page, a content delivery network (CDN), or another supporting service your application depends on.
- **Out of scope** – A URL the penetration test must not access at all.

By default, only your target URLs are reachable during the penetration test. This network scope does not apply to credential tests. Add an accessible entry for anything else your application needs. An entry can be a whole domain, such as `auth.example.com`, or a specific path, such as `https://example.com/callback`. An out-of-scope entry always takes precedence over an accessible one.

Accessible domains do not require ownership verification, even if they belong to a different domain than your target. Only target domains require verified ownership. AWS Security Agent does not penetration test accessible domains; it uses them only for login and navigation.

Non-standard ports for accessible domains

To make a non-standard port accessible, enter a full URL that includes the scheme and the port. For example, `https://auth.example.com:8443`. If you enter a domain without a scheme, such as `auth.example.com:8443`, AWS Security Agent allows the domain only on standard ports.

Classify a domain

1. On the **Network configuration** step, review the **Network scope** table.
2. For each domain, select **Accessible** or **Out of scope**.
3. To add a domain the tests did not discover, choose **Add domain**, then enter the URL and select its classification.
4. To remove a domain you added, choose the remove icon in its row. You cannot remove a target URL here.

Warning

Mark as out of scope any path that should not be accessed during testing, such as a destructive operation or a sensitive administrative function. AWS Security Agent excludes the specified path and all paths nested beneath it. For example, if you mark `https://example.com/admin` as out of scope, `https://example.com/admin/tools` is also out of scope.

To review the same domains grouped by classification instead of in a table, choose **Grouped** in **Domain view**. In grouped view, choose **Edit** to make changes, then choose **Save**.

The **Derived examples** list at the bottom of the section shows endpoints that cannot receive traffic from the penetration test, based on how you classified each domain. Use it to confirm your classifications before you continue.

Attach additional resources (optional)

Provide supplementary resources to help AWS Security Agent conduct more thorough and accurate penetration testing. Additional resources can include architecture diagrams, API documentation,

configuration files, GitHub repositories, or S3-hosted materials that give context about your application.

> **Note**
>
> Additional resources are optional but recommended. Providing comprehensive information about your application helps ensure thorough test coverage, reduces false positives, and delivers more actionable results.

Add resources to the penetration test

Select existing resources or upload new files that will help guide the penetration test.

1. In the **Connected resources** section, you can:
   - Choose **Select from available** to choose from resources already connected to AWS Security Agent (such as GitHub repositories or S3 buckets).
   - Choose **Upload** to add new files directly from your local system.

> **Tip**
>
> Useful resources include API documentation, architecture diagrams, OpenAPI/Swagger specifications, configuration files, authentication flow diagrams, and any other materials that describe your application's structure and behavior.

Select from available resources

Choose from resources that are already integrated with AWS Security Agent.

1. Choose **Select from existing resources**.
2. Browse the list of available resources from connected sources such as:
   - GitHub repositories, under the **GitHub repositories tab**
   - S3 buckets
   - Previously uploaded files
   - Documentation repositories

3. Select the resources you want to include in the penetration test.
4. Choose **Add to penetration test** to attach the selected resources.

Example

We recommend selecting and adding relevant GitHub repositories to your penetration test, so AWS Security Agent can develop an understanding of your application context, and generate ready-to-implement code fixes through pull requests (when enabled)

> **Tip**
>
> When you select a repository, you can specify a branch. By default, the service uses the primary branch. To use a different branch, enter the branch name in the field next to the repository.

> **Note**
>
> Resources selected from available sources remain synchronized with their original location. If you update a GitHub repository or S3 file, the penetration test will use the updated version.

> **Note**
>
> If you have a private VPC associated with your penetration test and a GitHub repository configured, ensure that GitHub is accessible through your private VPC for pulling GitHub resources. In most cases, you need to either ensure that outbound traffic through VPC NAT Gateway is allowed by default or configure specific rules to allow outbound traffic for GitHub IPs (see GitHub Meta API Endpoint)

Upload new resources

Upload files directly from your local system or provide plain text content to AWS Security Agent.

1. Choose **Upload**.

2. Choose one of the following input methods:
   - **Upload local files** - Select one or more files from your local system.
   - **Paste plain text** - Type or paste text content directly into the input field. Choose **Upload**.
3. Then choose **Add** to complete uploading.
4. The uploaded resources appear in the **Connected resources** table.

> **Tip**
>
> Use the plain text option when you want to quickly provide API endpoint lists, URL patterns, test instructions, or other text-based information without creating a separate file.

> **Important**
>
> Ensure uploaded files and pasted content do not contain sensitive information such as production credentials, private keys, or personally identifiable information (PII). Use sanitized versions of configuration files and documentation.

Connect existing resources

Existing resources can be from what you've previously uploaded to AWS Security Agent, from your S3 bucket, and your integrated GitHub repositories. Choose **Select from existing resources** to select them.

Manage connected resources

Review, organize, and remove resources attached to the penetration test.

The **Connected resources** table displays all resources included in the penetration test with the following information:

- **Name** - The filename or resource identifier
- **Type** - The resource category (Uploaded files, S3 resources, GitHub repositories, etc.)

To manage resources:

1. Select one or more resources using the checkboxes.
2. Choose **Remove from penetration test** to detach selected resources.

> **Note**
>
> You can sort the table by Name or Type by clicking the column headers. This helps organize resources when working with many files.

Set a maximum task-hours limit

Limit how much work AWS Security Agent performs for a penetration test by setting a maximum number of task hours. Task hours measure the cumulative time the agent spends actively working on the test, including time across parallel testing. Task hours are not the same as elapsed wall-clock time, because AWS Security Agent runs multiple tasks at once. Total task hours is also the unit that AWS Security Agent bills for a run, so setting a limit caps how much a test can cost. The smallest limit you can set is 20 hours.

1. (Optional) In the **Max task hours** section, choose a limit:
   - Choose a preset value, such as **20** or **30** hours.
   - Choose **No limit** to run the test to completion without a task-hours limit.
   - Choose **Custom** to enter your own value. A custom limit must be at least 20 hours.

> **Tip**
>
> For larger applications, or to get the most complete results, set a higher limit such as **30** hours or more. AWS Security Agent bills only for the task hours a run actually uses, so a higher limit does not increase cost unless the test needs the extra time.

> **Note**
>
> When a test reaches its maximum task hours, AWS Security Agent stops working on the test and keeps the findings already discovered. The run finishes with a status of **Completed**. You can review these findings or create and run a new test.

Exclude risk types (optional)

Choose specific risk categories to exclude from testing if they're not applicable to your application.

> **Note**
>
> **Exclude risk types** applies only to a **Web application** test scope. See the section called "Choose a test scope".

1. On the **Additional configuration** step, locate the **Exclude risk types** field.
2. Choose the dropdown to view available risk types.
3. Select one or more risk types to exclude from the penetration test.

> **Note**
>
> Excluding risk types limits the scope of testing. Only exclude risk types that are not relevant to your application or that you want to test separately.

Configure best-effort resource cleanup (optional)

While testing, AWS Security Agent can create resources in your application, such as user accounts or records it submits through forms and APIs. When the run ends, AWS Security Agent tries to delete the resources it created.

Cleanup runs as a final phase of the run. If you stop the run, or if the run ends before this phase, AWS Security Agent does not delete the resources it created.

Cleanup is best effort. If a deletion fails, AWS Security Agent continues with the remaining resources, and cleanup does not cause the run to fail. Some resources can therefore remain after the run. Review your application after a run and remove anything left behind.

1. On the **Additional configuration** step, locate the **Best-effort resource cleanup** section.
2. Choose whether to clean up created resources:
   - **Enabled** (default) – AWS Security Agent tries to delete the resources it created when the run ends.
   - **Disabled** – AWS Security Agent does not delete the resources it created.

Create the penetration test

Finalize and launch your penetration test configuration.

After configuring all settings, you're ready to create the penetration test.

1. Review all configuration sections to ensure accuracy.
2. Choose one of the following options:
   - Choose **Create penetration** to save the configuration without running it immediately.
   - Choose **Create and execute** to save the configuration and immediately start the penetration test.
   - Choose **Cancel** to discard the penetration test configuration.

Important

Before running a penetration test, verify that:

- All target domains are correctly verified and accessible
- IAM roles have appropriate permissions
- Out-of-scope paths are properly configured to prevent testing destructive operations
- You have authorization to perform security testing on all target domains

Note

After the penetration test starts, you can monitor its progress from the **Penetration test runs** section. The test can take several hours to complete, depending on the scope and complexity of your application. If you set a maximum task-hours limit, AWS Security Agent stops the test when it reaches that limit. AWS Security Agent preserves the findings discovered so far.

Review findings from a penetration test

Monitor penetration test execution in real time on the Penetration Test Logs page after AWS Security Agent starts a penetration test. AWS Security Agent logs every action during the

penetration test. After completion, review the penetration test summary, which includes application overview, coverage with identified endpoints, and risk assessment of security findings.

Evaluate security findings to address application vulnerabilities. Each finding contains impact assessment, severity rating, supporting evidence and remediation pull request details (when automatic code remediation is enabled).

Prerequisites

Before you begin, ensure you have:

- A completed or in-progress penetration test run
- Access to the AWS Security Agent web application

Step 1: Access the penetration test run

Navigate to your penetration test run to view overview, logs and findings pages.

1. Log in to the AWS Security Agent web application.
2. Navigate to the **Penetration tests** section.
3. Select the penetration test run you want to examine from the list.

> **Tip**
>
> The penetration test details page displays a summary of test status, completion date, and the number of findings identified.

Step 2: Monitor test progress

Track the progress of your penetration test run using the step indicator.

1. Locate the horizontal step indicator below the page header.
2. Review the status of each testing phase:
   - **Preflight** – Initial setup and connectivity checks
   - **Static analysis** – Code and configuration analysis
   - **Pentests** – Runtime testing and vulnerability scanning

- **Finalizing** – Final validation and report generation

> **Note**
>
> Each step displays a status indicator (Complete, In progress, or Pending). Findings are discovered and validated throughout the testing process, with new vulnerabilities appearing as each phase completes.

Step 3: Navigate to the penetration test run overview tab

1. Run Summary section provides test status, duration and other high level details. It also provides a dashboard of security findings categorized by severity level and risk-types
2. Application overview by AWS Security Agent provides a summary of the penetration test run
3. Discovered endpoints by AWS Security Agent provides a list of all endpoints discovered and tested by the AWS Security agent during the penetration test run

Step 4: Navigate to the penetration test logs tab

Access detailed logs of all actions AWS Security Agent executed during the penetration test.

1. The actions are categorized by action type and risk-types.
2. Select a specific action to view detailed logs:
   - **Testing Summary** – High-level summary of the agent actions and results
   - **Penetration test logs** – Detailed logs of all testing activities

Each task reports its own duration and task hours. For more information, see the section called “Pricing and billing”.

> **Note**
>
> Validator actions provide logs that validate findings in each category

> **Note**
>
> The Findings tab displays a split view with the findings list on the left and selected finding details on the right.

Step 5: Navigate to the findings

Each finding in the list displays key information to help you quickly assess its importance.

> **Note**
>
> **Default confidence filter**
>
> By default, you see only findings with **High** agent confidence. To also show findings with **Medium** or **Low** agent confidence and false positives, turn off the **Hiding unverified findings** toggle.

Review the information displayed on each finding card:

- **Finding name** – The title and identifier for the vulnerability
- **Confidence badge** – Indicates the agent's confidence level in the finding (High, Medium, or Low)
- **Severity badge** – Shows the color-coded risk level (Critical, High, Medium, Low, or Informational). For what each level means, see the section called "Severity ratings and CVSS metrics".
- **Last update timestamp** – Shows when the finding was last modified or validated
- **Description preview** – Brief summary of the vulnerability

> **Important**
>
> Prioritize findings with **Critical** or **High** severity badges and **High** confidence levels, as these represent validated vulnerabilities requiring immediate remediation.

Step 6: Review finding details

Select individual findings to view comprehensive information about each vulnerability.

1. Select a finding name in the left panel to display its details in the right panel.
2. Review the validation status:

> **Note**
>
> If a finding displays the Unknown "This finding is not validated by AWS Security Agent yet," it means the vulnerability detection is still being confirmed. These findings may require manual verification.

3. Review the key attributes displayed at the top:
   - **Agent confidence** – The confidence level AWS Security Agent has in this finding
   - **Severity** – The risk level with a color-coded badge
   - **Finding logs** – Choose "Trace actions & logs" to view detailed execution logs and evidence
   - **Risk type** – The category or type of security risk (e.g., Authentication Bypass, SQL Injection)
4. Expand the **Description** section to read:
   - A detailed explanation of the vulnerability
   - How the vulnerability works
   - Why it represents a security risk
   - The potential impact on your application
5. Expand the **Risk Reasoning** section to see the CVSS metrics breakdown that shows how the severity was calculated. For what each metric means, see the section called “Severity ratings and CVSS metrics”.
6. Expand the **Steps to reproduce** section to view:
   - Detailed technical steps to recreate the vulnerability
   - Request and response examples
   - Specific parameters or conditions that trigger the issue
7. The Verification Script section provides an executable way to reproduce the finding. Expand this section (when available) to view:
   - **Instructions** – How to set up and run the verification script

- **Environment variables** – Lists the required variables you must configure before running the script. Sensitive values are redacted for security.
- **Download Script** – Choose to download the executable verification script

> **Note**
>
> Verification scripts are available only for confirmed vulnerabilities. The agent must have successfully generated and validated an executable reproduction script.

> **Note**
>
> Verification scripts are generated using generative AI. Review the script before execution and run it only against systems you are authorized to test. For guidance on testing AI-generated scripts responsibly, see the AWS Responsible AI Policy.

> **Tip**
>
> The verification script provides an executable way to reproduce the finding independently. Set the required environment variables with your own credentials, then run the script against your target system to confirm the vulnerability exists.

> **Tip**
>
> Use the "Trace actions & logs" link to access the complete evidence package, including HTTP requests, responses, and exploitation attempts that demonstrate the vulnerability.

Edit findings

You can edit any finding to correct its details or refine the agent's assessment. The following fields are editable:

- **name** — The title of the finding
- **description** — Detailed description of the security vulnerability

- **status** — Current status of the finding
- **riskType** — Type of security risk identified
- **riskLevel** — Severity level (CRITICAL, HIGH, MEDIUM, LOW, INFORMATIONAL)
- **riskScore** — Numeric risk score
- **reasoning** — Justification for the assigned risk score
- **attackScript** — Proof-of-concept code demonstrating the vulnerability
- **customerNote** — Optional note explaining your rationale for the edit

To edit a finding:

1. Navigate to the finding detail page.
2. Choose the edit icon to modify any of the fields in the preceding list.
3. Save your changes.

> **Note**
>
> Your edits are saved immediately and reflected in the finding. If Findings Personalization is enabled, all editable fields you change will be used to refine the agent's preferences for future runs.

View the original agent version

When you edit a finding, a **Version selector** appears in the finding details header. This allows you to view the original agent assessment:

1. Locate the **Version selector** in the finding details header.
2. Select **Original** to view the finding as it was originally produced by the agent.
3. Select **Latest** to return to the current version with your edits applied.

Retest of previous findings

When you run a penetration test again, AWS Security Agent retests the findings from the most recent completed full run of that penetration test, before it tests for new findings. The **Logs** tab

shows these actions in the **Retest previous findings** section. A finding that the retest reproduces appears in the new run's findings, linked to the finding it was retested from.

AWS Security Agent doesn't retest findings that are false positives, findings that you accepted, or findings that the agent couldn't confirm. If you revalidated a finding after that run, AWS Security Agent uses the most recent conclusive revalidation result instead of the original finding.

To retest specific findings without running a full penetration test, see the section called "Revalidate penetration test findings".

Review personalized findings

When Findings Personalization is enabled, AWS Security Agent learns from your edits to findings. The agent applies those preferences to similar findings in future penetration test runs. When a finding has been adjusted based on your previous edits, the details panel displays a **Personalization changes** section. For information about enabling this feature, see Enable or disable Findings Personalization.

1. Locate the **Personalization changes** section in the finding details panel.

> **Note**
>
> The **Personalization changes** section appears only on findings that AWS Security Agent aligned to your previous edits. Findings that match no learned preference are shown exactly as the agent produced them, without this section.

2. Review the summary of what changed and why. The summary lists each adjusted attribute with its original agent-produced value, the personalized value, and the reasoning. For example:

Based on your previous edits the following changes have been made to the finding originally produced by the system: Risk level changed from MEDIUM to CRITICAL; Risk score changed from 5.3 to 9.5; Reasoning: Upgraded severity to reflect full exposure of application source code through publicly accessible source maps.

1. Review the summary to understand how the finding was adjusted before deciding how to act on it.
2. To override a personalized finding, edit it again as you would any other finding (for example, change the severity or risk score). Your most recent edit always takes precedence: AWS Security

Agent learns from it and applies the updated preference on the next run, replacing the earlier rule.

> **Note**
>
> Personalization changes adjust finding attributes such as riskLevel, riskScore, name, description, reasoning, and attackScript to reflect the standards AWS Security Agent learned from your edits. Any field you have previously edited may be adjusted on similar findings in future runs.

How personalization learns from your edits

When Findings Personalization is enabled, AWS Security Agent learns from all edits you make to findings and applies similar adjustments to future runs. Any field you edit (as listed in the preceding Edit findings section) will be used to refine the agent's learned preferences.

> **Note**
>
> For the **status** field, only marking a finding as **FALSE_POSITIVE** is treated as a learnable preference. Changes to other status values do not trigger learning.

On the next penetration test run, the agent evaluates new findings against your learned preferences and applies adjustments where applicable. A **Personalization changes** section appears on any finding that was adjusted, explaining what was changed.

> **Note**
>
> Personalization only learns from edits made in the most recent completed penetration test run. The feature must be enabled at the time the penetration test starts for learning to occur. If you edit the same finding in a later run, the most recent edit takes precedence — the agent updates its preferences to reflect your latest decision.

Enable or disable Findings Personalization

Findings Personalization is enabled by default. To disable or re-enable it:

1. Navigate to your penetration test configuration.
2. Locate the **Findings Personalization** toggle.
3. To enable Findings Personalization, turn on the toggle. To disable, turn it off.

When disabled, findings appear in their raw agent-produced state with no personalization applied. Previously learned preferences are preserved and will be applied again if the feature is re-enabled.

Step 7: Prioritize and address findings

Take action on findings to remediate vulnerabilities and improve your application's security posture.

For **Critical** and **High** severity findings with **High** confidence:

1. Review the Description and Steps to reproduce sections thoroughly.
2. Access the detailed logs via the "Trace actions & logs" link to gather complete evidence.
3. Access ready-to-implement code fixes through one of these methods:
   - For automatic remediation: Use the pull request link in the remediation section
   - For manual requests: Choose 'Remediation Code' on the findings page to request a pull request **Prerequisites:**
   - Admin must enable code remediation for GitHub repositories in the AWS Security Agent console
   - Repositories must be included in your penetration test configuration
4. Plan for a follow-up penetration test to verify the fix is effective.

For **Medium** and **Low** severity findings:

1. Prioritize based on your risk tolerance and business context.
2. Include remediation tasks in your regular development sprint planning.
3. Consider whether multiple low-severity findings together create higher risk.
4. Document any accepted risks with appropriate justification.

Important

Do not ignore low-severity findings. Multiple low-severity vulnerabilities can often be chained together to create more serious exploits, especially when combined with social engineering or physical access.

Step 8: Track remediation progress

Use the findings interface to track which vulnerabilities have been addressed and which require further action.

1. As you work on remediation, refer back to the Steps to reproduce section to verify your fixes.
2. Document your remediation approach for each finding for future reference and compliance audits.

Tip

Maintain a remediation log that maps each finding to its resolution, including the code changes, configuration updates, or architectural decisions made to address the vulnerability.

Next steps

After reviewing your penetration test findings:

- Prioritize critical and high-severity findings with high confidence for immediate remediation
- Download, review, and run verification scripts in your test environment to test the scripts and identify vulnerabilities
- Create a Jira issue for a finding so your team can track it in Jira. See the section called “Create Jira issues for findings”.
- Implement fixes and security controls to address identified vulnerabilities
- Monitor the penetration test run progress indicator for newly discovered vulnerabilities
- Revalidate a finding after you deploy a fix to confirm whether it is still exploitable. See the section called “Revalidate penetration test findings”.

- Schedule a follow-up penetration test to verify that broader changes have not introduced new vulnerabilities
- Update your application security testing process and threat model based on findings
- Review CVSS metrics to understand your application's overall security posture

For more information about performing penetration tests, see the section called "Create a penetration test".

For more information about understanding the Security Agent lifecycle, see the section called "Understand the resource hierarchy and lifecycle".

Exporting a penetration test report

Exporting a penetration test report generates a downloadable PDF record of your test results. Use the report to preserve test evidence and share findings with stakeholders. For example, you can share it with an auditor or attach it to an internal risk assessment record.

The report does not establish compliance or certification. Your auditor determines whether the report and your supporting records satisfy your organization's audit requirements.

Understanding compliance-ready reports

Use a compliance-ready report when you need to give an auditor structured evidence of your penetration testing. The report records when and how you tested, what you tested, what you found, and how your organization responded. You can use it as supporting evidence for audits under these frameworks:

- The American Institute of Certified Public Accountants (AICPA) Trust Services Criteria for System and Organization Controls (SOC) 2
- International Organization for Standardization and International Electrotechnical Commission (ISO/IEC) 27001:2022
- The Health Insurance Portability and Accountability Act (HIPAA) Security Rule

These frameworks do not require a specific penetration test report. They define control outcomes rather than a required report format. Use the report as one source of evidence within your organization's broader control program.

For framework requirements, see the AICPA Trust Services Criteria, ISO/IEC 27001, and 45 CFR 164.308.

Mapping report evidence to control areas

The following table explains how auditors can use report evidence. It does not replace the framework requirements or your auditor's judgment.

Framework and control areas	What an auditor evaluates	Relevant report evidence
AICPA Trust Services Criteria for SOC 2: CC4.1, CC7.1, and CC7.4–CC7.5	Whether your organization evaluates controls, identifies vulnerabilities, and tracks corrective action. For a Type II examination, the auditor also evaluates whether evidence falls within the examination period.	The test window, scope, methodology, findings, severity ratings, proof-of-concept, and reproduction steps. Reviewer annotations can document your response when you include them. Pair the report with remediation and revalidation records.
ISO/IEC 27001:2022: A.8.8, A.8.25, and A.8.29	Whether your organization uses planned and documented processes to identify and manage technical vulnerabilities and perform security testing during development and acceptance.	The test scope, four-phase methodology, tester-independence statement, vulnerability coverage, findings, and reproduction evidence. Pair the report with your risk-treatment and remediation records.
Health Insurance Portability and Accountability Act (HIPAA) Security Rule: 45 CFR	Whether your organization performs an accurate and thorough risk analysis, manages identified risks, and periodically evaluates its safeguards.	The assessment boundary, test window, findings, severity ratings, and evidence. Pair the report with records of mitigation decisions and follow-up evaluation.

Framework and control areas	What an auditor evaluates	Relevant report evidence
164.308(a)(1)(ii)(A), 164.308(a)(1)(ii)(B), and 164.308(a)(8)		

Neither SOC 2, ISO/IEC 27001:2022, nor the HIPAA Security Rule explicitly requires a penetration test. A penetration test report can provide evidence for the control areas in the table when its scope and timing match the audited system and period.

Reviewing the compliance-ready report contents

The following list describes the content added when you turn on **Generate a compliance-ready report**:

- **About this report** — Explains that the report provides supporting evidence for security-testing and vulnerability-management controls. It also states the report's compliance limits.
- **Methodology** — Describes the four-phase testing methodology: preflight, static analysis, penetration testing, and finalizing.
- **Tester independence** — Identifies AWS Security Agent as an automated third-party testing service that operates independently from the tested system's builders and operators.
- **Test window** — Shows when the engagement occurred so you can align the evidence with an audit or examination period.
- **Vulnerability coverage** — Lists the vulnerability classes in scope and groups them by category. For a web application test scope, coverage aligns with the Open Worldwide Application Security Project (OWASP) Top 10 for web applications.
- **Confidentiality** — Marks the report as confidential. You can share it with your auditors and assessors under confidentiality.

The **Tasks Executed** section records the tests performed during the engagement. The vulnerability coverage appendix describes classes in scope; it does not state that AWS Security Agent tested every class against every endpoint.

AWS Security Agent is not a professional penetration testing service, a Payment Card Industry Qualified Security Assessor (QSA), or an Approved Scanning Vendor (ASV). The report is not a Payment Card Industry Data Security Standard (PCI DSS) 11.4 penetration test deliverable. It is also not a Federal Risk and Authorization Management Program (FedRAMP) penetration test deliverable.

Preparing evidence for an auditor

Before you share a report, confirm that the evidence matches the audit request:

- Make sure that the target URLs, connected resources, and exclusions match the audited system boundary.
- Make sure that the test window falls within the period your auditor requested.
- Include the severities, confidence levels, statuses, and finding types your auditor expects.
- Document who authorized the test and why the tester was appropriate.
- Include records that show how you handled each applicable finding.
- Include revalidation results or other follow-up evidence when your auditor requests them.
- Follow your organization's confidentiality requirements when sharing and storing the report.

The report provides point-in-time test evidence. Your vulnerability-management process supplies the management response, remediation ownership, exceptions, and ongoing control evidence that an auditor may also request.

Including annotation notes

Turn on **Include annotation notes** to add each reviewer's annotation to its finding. Use annotations to capture context such as risk justification, triage decisions, accepted-risk rationale, or remediation status.

The report separates annotation notes added by the tested organization from the agent's assessment. For information about adding an annotation, see Edit findings.

Preparing to export a report

Confirm that you have access to the AWS Security Agent web application and permission to view the penetration test run. A completed run provides the final test window and findings. You can also export an in-progress run when you need an interim record.

For information about running a test, see Create a penetration test. For information about interpreting findings, see Review findings from a penetration test.

To open the Generate report dialog

Open the dialog when you need to export a completed or in-progress penetration test run:

1. Log in to the AWS Security Agent web application.
2. Navigate to **Penetration tests**.
3. Choose the penetration test run that you want to export.
4. In the toolbar, choose **Generate report**.

Choosing findings to include

Filter findings to focus on the issues relevant to your review and reduce generation time for large reports. The following table lists the available filters and the findings they include.

Filter	Findings included
Risk levels	Findings with the selected severity levels: Critical, High, Medium, Low, or Informational.
Confidence levels	Findings with the selected agent-confidence levels: High, Medium, or Low.
Finding status	Findings with the selected workflow statuses.
Finding types	Security findings, requirement violations, or both.

A report with many findings might take several minutes to generate. If the report is large, narrow the risk level or finding type filters first. For information about severity levels, see Understanding severity and CVSS scores.

To generate and download the report

Generate the report when you need to share findings or preserve evidence for remediation and audit review. This procedure downloads the findings that match your filters. After you confirm the filters, complete the following steps:

1. Review your filter selections.
2. (Optional) Turn on **Generate a compliance-ready report**.
3. (Optional) Turn on **Include annotation notes**.
4. Choose **Generate and download**.

For details about these options, see Understanding compliance-ready reports and Including annotation notes.

Your report downloads as a PDF file. A large report might take several minutes to generate.

After you download the report

After downloading the report, you can take these actions:

- Share the report with stakeholders or your auditor according to your organization's confidentiality requirements.
- Track a finding in Jira. See Export findings to an issue tracker.
- Revalidate a finding after you deploy a fix. See Revalidate findings.

Revalidate penetration test findings

After you remediate a vulnerability, use AWS Security Agent to revalidate the finding and confirm whether it is still exploitable. Revalidation re-tests one or more existing findings without running a full penetration test. You get a fast answer on whether your fix worked.

During revalidation, AWS Security Agent authenticates to your application and re-runs the validation steps for each selected finding. It does not rediscover new vulnerabilities or test other parts of your application. This makes revalidation faster and more focused than a full penetration test.

Prerequisites

Before you begin, ensure you have:

- A completed penetration test with at least one finding
- Access to the Agent Space that contains the finding

- A target application that is reachable with the current authentication configuration of the penetration test

Select findings to revalidate

Choose one or more findings from a completed penetration test run:

1. Log in to the AWS Security Agent web application.
2. Open the penetration test run that contains the findings you want to revalidate.
3. In the findings panel, choose **Multiselect findings for revalidation**.
4. Select the check box next to each finding you want to revalidate. The findings you select are listed as you add them.
5. Choose **Revalidate findings**.

> **Before you revalidate**
>
> Revalidate a finding after you deploy a fix to the environment the original test ran against. Revalidation re-tests the same target, so the fix must be live for the result to be accurate.

Interpret revalidation results

Revalidation runs as its own job and does not change the original finding. For each finding, revalidation reports whether AWS Security Agent could reproduce the vulnerability.

Result	What it means
Active	AWS Security Agent reproduced the vulnerability. The finding is still exploitable and needs further remediation.
Resolved	AWS Security Agent could not reproduce the vulnerability during revalidation.

If a finding is linked to a Jira issue, AWS Security Agent adds a comment to that issue with the revalidation result.

Trace revalidation history

Revalidation keeps the link between a finding and the runs that re-tested it, so you can trace remediation over time.

- From a revalidation job, choose **Original finding** to open the finding it re-tested.
- From a finding, view its **Revalidation jobs** to see each time it was revalidated and the result.

A revalidation run appears in your run history with a type of **Revalidation**, which distinguishes it from a full penetration test.

Next steps

After revalidation, consider these next steps:

- Revalidate a finding again after each remediation attempt until its status is **Resolved**.
- Run a full penetration test for broad changes to your application instead of revalidating individual findings. For details, see the section called “Create a penetration test”.

Provide authentication credentials for penetration testing

Provide credentials to enable AWS Security Agent to test authenticated areas of your web applications. Without credentials, the agent can only test publicly accessible pages and APIs.

> **Important**
>
> Some identity providers block sign-in from automated browsers under their own anti-automation policies. A provider that does this can refuse the agent’s browser before it evaluates your credentials. Sign-in then fails even though the credentials are valid. Each provider controls whether it applies these checks and when.
> If your application delegates sign-in to such a provider, provide a credential that the agent can replay rather than earn by signing in. For more information, see the section called “Select static credential from connected AWS Secrets Manager”.

Configure authentication credentials

1. In the penetration test creation workflow, go to the **Credentials** step.

2. Choose **Add credential**, then enter a **Credential name** that is unique within this penetration test.
3. Choose your credential input method:
   - **Input credentials** - Enter credentials directly. Best for development and testing environments.
   - **Advanced setting** - Use AWS-native credential management. Recommended for production environments and sensitive credentials.

Advanced options

If you select **Advanced setting**, you can choose from three credential strategies:

- **IAM role assumption** - For applications using AWS Cognito or IAM authentication
- **AWS Secrets Manager** - For secure credential storage with encryption and rotation
- **Lambda function** - For dynamic credential generation or complex authentication flows

Input credentials directly

1. Select **Input credentials**.
2. Enter the **User name** and **Password**.
3. In the **Access URL** dropdown, select the URL where these credentials will be used. This must be selected from the list of target endpoints.
4. (Optional) In the **2FA - optional** field, provide a TOTP secret for applications that require two-factor authentication. You can either:
   - Enter the TOTP secret directly (for example, `JBSWY3DPEHPK3PXP`), or enter the full `otpauth://totp/` URI (for example, `otpauth://totp/Example:user@example.com?secret=JBSWY3DPEHPK3PXP&issuer=Example`).
   - Choose the upload icon to upload a QR code image from your authenticator app setup page. The QR code is scanned locally and the TOTP URI is extracted automatically.

     When a TOTP secret is provided, the agent automatically generates fresh one-time codes and enters them when a 2FA prompt is detected during login.
5. Under **Agent Space login prompt**, enter instructions that describe how to sign in with these credentials. A login prompt is required for each credential. To start from a worked example, choose **Website login**, **API request**, **Amazon Cognito**, or **IAM Identity Center** below the field, then edit the inserted text.

6. Choose **Save**, or choose **Save and test** to save the credential and test the sign-in right away. For more information about testing, see the section called “Create a penetration test”.

> **Important**
>
> Use test accounts with representative access rather than personal or administrative accounts. Test against a non-production environment, and do not use an account that has access to production systems. For more information, see Use non-production environments for penetration testing.

Use advanced setting

When you select an advanced credential strategy (Secrets Manager, Lambda, or IAM role), AWS Security Agent retrieves your credentials directly from the configured AWS resource using the service role. Use the **Agent Space login prompt** to provide the agent with instructions on how to apply those credentials to your application - for example, which login URL to navigate to and which form fields to fill in.

> **Note**
>
> Secrets Manager secrets and Lambda functions must be in the same AWS account as your AWS Security Agent setup. Cross-account credentials are not currently supported.

1. Select **Advanced setting**.
2. In the **User access strategy** dropdown, choose one of the following:

Select available IAM role for agent to assume

Use this option for applications using AWS Cognito, API Gateway with IAM authentication, or other AWS-native authentication systems. AWS Security Agent assumes the role at the start of each penetration test run. It then uses the returned temporary credentials to sign requests to your application.

Configure the IAM role in two steps:

1. **Register the role with your Agent Space.** In the AWS Security Agent console, open your Agent Space and choose **Edit configuration**. Under **Service access**, choose **Use an existing service role**, and then select your role from the dropdown list.

   > **Note**
   >
   > You can select only roles registered here as an actor credential in the penetration test configuration.

2. **Configure the role's trust policy** to allow AWS Security Agent to assume it. Use the following trust policy. Replace `<region>` and `<your-account-id>` with your own values:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "securityagent.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": { "aws:SourceAccount": "<your-account-id>" },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:securityagent:<region>:<your-account-id>:agent-
space/*"
      }
    }
  }]
}
```

The role also needs permissions to access your application's authentication system (for example, API Gateway or Cognito).

Select static credential from connected AWS Secrets Manager

Use this option to retrieve credentials securely from AWS Secrets Manager with encryption, rotation, and access auditing.

The IAM role must have `secretsmanager:GetSecretValue` and `secretsmanager:DescribeSecret` permissions.

The agent retrieves the secret value directly from Secrets Manager. Use the **Agent Space login prompt** to tell the agent how to apply those credentials to your application's login flow - for example, which URL to navigate to and which form fields to fill in. You may use any format to store your secret, as the agent will interpret the format using the instructions you provide in the login prompt.

For example, if the agent is to submit a username/password login form at https://example.com/login, you may format your secret as JSON with `username` and `password` fields. If the application requires TOTP-based 2FA, include a `totpSecret` field with either the TOTP secret directly or a full `otpauth://totp/` URI:

```
{
  "username": "test-user",
  "password": "secure-password-here",
  "totpSecret": "JBSWY3DPEHPK3PXP"
}
```

Then, configure the authentication instructions: . Set **Access URL** to `https://example.com` (or any other URL selected from the list of target endpoints). . Enter the following into **Agent Space login prompt**: "Navigate to https://example.com/login and enter the provided username and password into the form."

As another example, if you instead have an API key to be provided in an HTTP header, you may store it as plaintext:

```
"api-key-here"
```

Then, configure the authentication instructions: . Enter the following into **Agent Space login prompt**: "Set the X-API-Key header to the provided API key for all requests."

> **Important**
>
> AWS Security Agent supports TOTP-based 2FA and email-based MFA. It does not support SMS, push notifications, or hardware keys. For applications that send a one-time code or verification link by email, see the section called "Configure email MFA".

Select available Lambda function to retrieve credentials dynamically

Use this option for complex authentication systems, dynamic credential generation, or integration with external identity providers.

The IAM role must have `lambda:InvokeFunction` permissions and the function must complete within 30 seconds.

When the penetration test runs, AWS Security Agent invokes your AWS Lambda function synchronously and passes an event that identifies the penetration test and the specific credential being resolved:

```
{
  "pentest_arn": "arn:aws:securityagent:us-west-2:111122223333:pentest/pt-
abcd1234-5678-90ab-cdef-EXAMPLE11111",
  "actor_identifier": "Credential1"
}
```

- `pentest_arn` – The Amazon Resource Name (ARN) of the penetration test that the credential is being resolved for. Use it to scope, authorize, or audit the request within your function.
- `actor_identifier` – The **Credential name** you assigned to this credential in the console (for example, `Credential1`). Use it to return the correct credential when a single function serves multiple credentials.

The agent uses the function's output directly as the credential. Use the **Agent Space login prompt** to tell the agent how to apply those credentials to your application, the same way you would with AWS Secrets Manager. Refer to the section called "Select static credential from connected AWS Secrets Manager" for examples of how to format the output of your Lambda function and supported authentication types.

Configure email MFA

Use email MFA when your application sends a one-time code or verification link by email as part of its authentication flow. If your application uses an authenticator app instead, provide a TOTP secret as described in the section called "Input credentials directly".

When you enable email MFA, AWS Security Agent generates a unique email address for each credential. During login, AWS Security Agent reads the messages that arrive at this address and

submits the code or link to complete authentication. You can also view these messages yourself, as described in the section called "View email MFA messages".

You can use the address in two ways, depending on whether you test with a new user account or an existing mailbox:

- **Sign up a test user with the address** – Enter the address as the email address when you create a test user in your application. Your application sends its MFA messages to the address directly, so you do not need your own mailbox or a forwarding rule. To open a verification message that your application sends when it creates the account, see the section called "View email MFA messages".
- **Forward from your own mailbox** – Keep using an existing test account, and create a rule in your email provider that forwards only your application's MFA messages to the address. For more information, see the section called "Set up email forwarding".

> **Note**
>
> Email MFA is available for both the input credentials and advanced setting credential methods, and in both the console and the AWS CLI or API.

Enable email MFA in the console

1. In the **Add credential** dialog box, select your credential input method and enter your credentials as described earlier in this topic.
2. Expand **2FA - optional**, then choose **Email MFA**.
3. Create the penetration test. AWS Security Agent then displays the **Email MFA forwarding address** for the credential. You cannot choose this address yourself.
4. Copy the address, then either sign up a test user in your application with it, or configure forwarding in your email provider. For more information, see the section called "Set up email forwarding".

> **Note**
>
> If you dismiss this display before you copy the address, you can retrieve it later from the **Configurations** page of the penetration test, which shows the **Email MFA forwarding address** for each credential.

Enable email MFA with the AWS CLI or API

Set `enableEmailMfa` to `true` for the actor in the `assets` parameter of your `create-pentest` or `update-pentest` request. For example:

```
aws securityagent create-pentest \
  --title "My penetration test" \
  --agent-space-id "your-agent-space-id" \
  --assets '{"actors": [{"identifier": "test-user", "enableEmailMfa": true}]}'
```

AWS Security Agent returns an `mfaForwardingAddress` for each actor in the `assets` field of the response.

Set up email forwarding

Create a separate forwarding rule for each credential that uses email MFA. The general steps are the same for every provider:

1. Sign in to the email account that receives your application's MFA messages.
2. Create a rule that matches only your application's MFA messages. For example, match on the sender address or on a subject line that identifies the message.
3. Set the rule to forward matching messages to the MFA forwarding address that you copied.
4. Send a test MFA message from your application, then confirm that the rule forwards it.

Some providers send a verification message to the destination address and require you to open a link in that message before forwarding begins. You can complete this step by viewing the verification message as described in the section called "View email MFA messages".

The following providers support forwarding to the MFA forwarding address:

- **Gmail and Google Workspace** – Add the address under **Settings**, **Forwarding and POP/IMAP**. Open the verification message that Gmail sends to it. Then create a filter that forwards only your application's MFA messages. Leave blanket forwarding disabled so that Gmail forwards only the messages your filter matches. For more information, see Automatically forward Gmail messages to another account in the Google documentation.
- **Microsoft 365 and Outlook** – Create an inbox rule that forwards or redirects matching messages. For more information, see Use rules to automatically forward messages in the Microsoft documentation.

- **Yahoo Mail** – Use a Yahoo account dedicated to testing, because Yahoo forwards the whole account rather than selected messages. Forwarding requires a Yahoo Mail Plus subscription, and Yahoo doesn't offer it in all locales. For more information, see Enable automatic email forwarding in Yahoo Mail in the Yahoo documentation.
- **iCloud Mail** – Create a rule that forwards matching messages. Each rule forwards to one address. For more information, see Automatically forward email in Mail on iCloud.com in the iCloud User Guide.
- **Custom domains and self-hosted mail servers** – Add a server-side alias, sieve rule, or filter that forwards only the matching messages. For more information, consult your mail server's documentation.

> **Note**
>
> You can also forward MFA messages manually. Because MFA codes expire quickly, forward each message to the MFA forwarding address as soon as it arrives during login.

> **Important**
>
> Forward only your application's MFA messages. Do not forward your entire inbox. A narrow rule keeps unrelated email out of the penetration test and limits what your rule sends to AWS Security Agent.

View email MFA messages

You can view the messages that arrive at a credential's MFA forwarding address. Use this to read a verification link when you sign up a test user with the address. You can also use it to confirm that your forwarding rule works, or to troubleshoot a login that failed at the MFA step.

To view the messages in the console, open the penetration test and choose the **Configurations** page. The page lists each credential with its own **Email MFA forwarding address**. Choose **View inbox** under the address for the credential whose messages you want to read.

To view the messages with the AWS CLI, run `list-actor-messages`:

```
aws securityagent list-actor-messages \
  --agent-space-id "your-agent-space-id" \
```

```
  --pentest-id "your-pentest-id" \
  --actor-identifier "test-user"
```

The response includes the sender, subject, body, and receipt time of each message, with the most recent first.

Data retention

AWS Security Agent stores the messages that it receives at the forwarding address only to complete the penetration test login. AWS Security Agent automatically deletes these messages 24 hours after receiving them, and you can view a message only during this period.

Configure multiple credentials

To test different user roles or authentication systems:

1. On the **Credentials** step, choose **Add credential**.
2. Configure the additional credential using either input method.
3. To remove a credential, choose the remove icon in its row.

Login Optimization

Login Optimization allows AWS Security Agent to learn navigation patterns from previous penetration test runs and apply them to subsequent runs. These patterns include login flows and multi-step authentication workflows. This reduces the time the agent spends re-discovering how to authenticate, resulting in faster scans and more consistent testing coverage.

How Login Optimization works

On the first penetration test run, the agent discovers how to navigate your application's login flow using the credentials you provide. It records the successful navigation steps and generates a skill file that captures the learned login workflow.

On subsequent runs, the agent reads the saved skill file and applies the learned navigation pattern directly, skipping the discovery phase. This means:

- Faster time to authenticated testing — the agent applies proven navigation patterns immediately instead of exploring from scratch
- More consistent behavior — the agent follows the same proven path rather than exploring alternatives

> **Note**
>
> Login Optimization learns during the first login session within a run. Subsequent login sessions in the same run will benefit from what was learned in the first session. On future runs, all login sessions benefit from the saved skill.

Enable or disable Login Optimization

Login Optimization is enabled by default. To disable or re-enable it:

1. In the penetration test configuration, go to the **Additional configuration** step.
2. Locate the **Login optimization** section.
3. To enable Login Optimization, turn on the toggle. To disable, turn it off.

> **Note**
>
> The **Login optimization** section appears only when the penetration test has at least one credential. Add a credential first, then return to the **Additional configuration** step.

When disabled, the agent re-discovers the login flow from scratch on every run. Previously learned navigation skills are preserved and will be applied again if the feature is re-enabled.

> **Tip**
>
> If your application's login flow changes significantly (for example a redesigned login page or a new authentication step), the agent automatically adapts by updating its learned skill on the next run. You do not need to manually reset the optimization.

Provide trusted CA certificates for a penetration test

If a target endpoint uses a Transport Layer Security (TLS) certificate that is not publicly trusted, provide the trust anchor. This applies when the certificate is issued by a private or internal certificate authority (CA), an intermediate CA, or a self-signed certificate. AWS Security Agent uses the trust anchor to validate the connection during testing. Without the trust anchor, endpoint validation rejects the target before testing begins.

Trusted CA certificates apply to the entire penetration test. You can provide them in the AWS Security Agent web application or with the AWS CLI or API.

> **Note**
>
> Skip this task if your target endpoints use certificates issued by a publicly trusted CA.

> **Important**
>
> Provide the certificate only, never a private key. AWS Security Agent does not accept a certificate that is expired or not yet valid.

Provide a trusted CA certificate in the console

Trusted CA certificates are part of **Advanced network access** in the penetration test create and edit forms.

1. In the penetration test form, expand **Advanced network access**, and then expand **Trusted CA certificates - optional**.
2. Choose **Add certificate**.
3. For **Source**, choose one of the following:
   - **Paste PEM** - Paste a PEM-encoded X.509 certificate. An inline certificate is limited to 8,192 characters.
   - **Upload a file** - Upload a `.pem`, `.crt`, or `.cer` certificate file.
   - **S3 URI** - Enter the Amazon S3 URI of a certificate that you staged, in the form `s3://amzn-s3-demo-bucket/ca.pem`. AWS Security Agent reads the certificate from Amazon S3 when the penetration test runs, using the penetration test service role. Enter the `s3://` URI form. An Amazon S3 HTTPS URL is not accepted.
4. Review the parsed certificate details (subject, issuer, type, and validity), and then choose **Save**.

To add more than one trust anchor, choose **Add certificate** again. To change or remove a certificate later, choose the actions menu on its row, and then choose **Edit** or **Remove**. Trusted CA certificates that you save are retained when you edit the penetration test.

Provide a trusted CA certificate with the AWS CLI or API

Set `trustedCaCertificates` in the `assets` parameter of your `create-pentest` or `update-pentest` request. Each entry has exactly one `source`: an inline PEM-encoded certificate (`inlinePem`) or an Amazon S3 location (`s3Location`).

The following example attaches an inline certificate:

```
aws securityagent create-pentest \
  --title "My penetration test" \
  --agent-space-id "your-agent-space-id" \
  --assets '{"trustedCaCertificates": [{"source": {"inlinePem": "-----BEGIN
 CERTIFICATE-----\n...\n-----END CERTIFICATE-----"}}]}'
```

The following example reads the certificate from Amazon S3 when the penetration test runs. AWS Security Agent reads the object using the penetration test service role, so the role must have `s3:GetObject` permission for the object.

```
aws securityagent create-pentest \
  --title "My penetration test" \
  --agent-space-id "your-agent-space-id" \
  --assets '{"trustedCaCertificates": [{"source": {"s3Location": "s3://amzn-s3-demo-
bucket/ca.pem"}}]}'
```

To update the trust anchors on an existing penetration test, send the complete list in `trustedCaCertificates` on an `update-pentest` request. The list replaces the previously saved anchors, and an empty list removes them all.

Provide client certificates for mutual TLS

Attach a client certificate to a penetration test so that AWS Security Agent can test endpoints that require mutual Transport Layer Security (mTLS). An mTLS endpoint accepts a connection only when the client presents a certificate during the TLS handshake. Without a client certificate, the endpoint rejects the connection and AWS Security Agent cannot test it.

You store each client certificate and its private key in an AWS Secrets Manager secret in your account, and reference the secret from the penetration test. AWS Security Agent stores only the secret ARN. It reads the certificate and private key from the secret when the penetration test runs. You can attach client certificates in the AWS Security Agent web application or with the AWS CLI or API.

> **Note**
>
> Skip this task if your target endpoints do not require a client certificate.
> The endpoint's own server certificate is a separate setting. If it is self-signed, or issued by a private or internal certificate authority (CA), you also provide a trust anchor. For more information about trusted CA certificates, see the section called "Provide trusted CA certificates for a penetration test". An endpoint can need both.

How client certificates apply

- **A certificate applies to a whole host** – You attach a certificate to one or more hosts, and it applies to every path and port on each host. A host can use only one client certificate.
- **Hosts must be target endpoints** – Each host must belong to a target URL of the penetration test.
- **Hosts must use HTTPS and a hostname** – An origin uses the form `:port` with no path. An IP address is not accepted. AWS Security Agent selects the certificate from the TLS server name, and a connection to an IP address does not carry one.
- **Certificates are presented only to their hosts** – AWS Security Agent presents a certificate only when it connects to a host that you attached it to. It never presents the certificate to any other host.
- **Certificate names** – A name can be 1–100 characters: letters, numbers, hyphens, and underscores, starting with a letter or number. Names must be unique within the penetration test.

> **Important**
>
> The certificate must remain valid for the whole run. AWS Security Agent rejects a certificate that has expired or is not yet valid. It also requires the certificate to stay valid for at least 36 hours after the penetration test starts. If you set a maximum task-hours limit lower than 36, the certificate must stay valid for that number of hours instead.

AWS Security Agent reads the secret once, when the penetration test starts. Rotating the secret during a run does not affect that run.

Store the certificate in Secrets Manager

The secret must hold a JSON `SecretString` with the following fields:

```
{
  "certificate": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
  "privateKey": "-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----",
  "privateKeyPassword": "optional"
}
```

- `certificate` – The PEM-encoded client certificate, followed by any intermediate CA certificates.
- `privateKey` – The PEM-encoded private key that matches the certificate.
- `privateKeyPassword` – (Optional) Include it only if the private key is encrypted.

Each PEM value must contain real line breaks: the `BEGIN` and `END` markers each on their own line, and the base64 body on the lines between them. In JSON, write each line break as `\n`.

> **Note**
>
> If you create the secret in the Secrets Manager console, use the **Plaintext** tab and paste the JSON. The **Key/value** tab stores `\n` as two literal characters rather than a line break, and AWS Security Agent then cannot read the certificate. Alternatively, the AWS Security Agent web application can create the secret for you, as described in the section called “Provide a client certificate in the console”.

The following AWS CLI command builds the secret from local `client.crt` and `client.key` files, so that the line breaks are encoded correctly. Replace `client.crt` and `client.key` with the paths to your certificate and private key files, and replace `my-agent-space-partner-api-cert` with your secret name. Start the name with your Agent Space name so that the service role can read the secret without an extra grant:

```
aws secretsmanager create-secret \
  --name "my-agent-space-partner-api-cert" \
  --secret-string "$(jq -n --rawfile c client.crt --rawfile k client.key \
      '{certificate: $c, privateKey: $k}')"
```

Grant access to the secret

The penetration test service role reads the secret when the penetration test runs. It must have the following permissions:

- `secretsmanager:GetSecretValue` on the secret.
- `kms:Decrypt` on the secret's AWS Key Management Service (AWS KMS) key, if the secret is encrypted with a customer managed key. For more information about customer managed keys, see the section called "Customer managed keys".

If the secret name starts with your Agent Space name, the service role policy described in the section called "Create the IAM roles for AWS Security Agent" already grants `secretsmanager:GetSecretValue` on it. Otherwise, add a statement for the secret, as in the following example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:partner-api-
cert-AbCdEf"
    },
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}
```

If you let the AWS Security Agent web application create the secret, your own IAM identity must also have `secretsmanager:CreateSecret`. If you encrypt your Agent Space with a customer managed key, the web application encrypts the secret with that key, so your IAM identity also needs `kms:GenerateDataKey` and `kms:Decrypt` on it.

Provide a client certificate in the console

Client certificates are part of **Advanced network access** in the penetration test create and edit forms. Add the target URLs first, because you choose the certificate's hosts from them.

To attach a client certificate, follow these steps:

1. On the **Penetration test details** step, in the **Penetration testing scope** section, expand **Advanced network access**, and then expand **Client certificates for mTLS - optional**. In some configurations, the client certificates appear directly under **Advanced network access**, without a separate section to expand.
2. Choose **Add certificate**.
3. For **Name**, enter a name for the certificate, such as `partner-api-cert`.
4. For **Certificate source**, choose one of the following:
   - **Use an existing secret** – Enter the **Secrets Manager ARN** of a secret that you created as described in the section called "Store the certificate in Secrets Manager". If the secret name does not start with your Agent Space name, grant the service role the permission that the form shows.
   - **Create a new secret** – Paste the **Certificate** and the **Private key** in PEM format, and enter the **Private key password** if the key is encrypted. AWS Security Agent creates the secret in your account when you save the penetration test, and names it so that the service role can already read it. For the permissions this requires, see the section called "Grant access to the secret".
5. For **Hosts**, choose one or more target hosts that require this certificate.
6. Choose **Save**.

To attach another certificate, choose **Add certificate** again. To change or remove a certificate later, choose the actions menu on its row, and then choose **Edit** or **Remove**. While a certificate applies to a host, you cannot change that host's target URL. Update or remove the certificate first.

Provide a client certificate with the AWS CLI or API

Set `clientCertificates` in the `assets` parameter of your `create-pentest` or `update-pentest` request. Each entry has a `name`, the `secretArn` of the secret that holds the certificate and private key, and the `origins`, which are the hosts that the certificate covers.

The following example attaches one client certificate to a target host:

```
aws securityagent create-pentest \
  --title "My penetration test" \
  --agent-space-id "your-agent-space-id" \
  --assets '{
    "endpoints": [{"uri": "https://api.example.com/"}],
    "clientCertificates": [{
      "name": "partner-api-cert",
      "secretArn": "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-agent-
space-partner-api-cert-AbCdEf",
      "origins": ["https://api.example.com"]
    }]
  }'
```

To update the client certificates on an existing penetration test, use `update-pentest`. The `assets` parameter replaces all saved assets, not only the client certificates. Send the complete `assets` object, including the endpoints, any other assets, and the full `clientCertificates` list. An empty `clientCertificates` list removes all client certificates.

Troubleshoot client certificate errors

AWS Security Agent checks each client certificate before testing begins. If a check fails, the penetration test stops with a message that names the certificate. Find the message in the following table, and take the action for that certificate.

Message contains	What to do
Cannot assume the pentest service role	Confirm that the service role's trust policy allows AWS Security Agent to assume it for this penetration test. For instructions on configuring the service role, see the section called "Create the IAM roles for AWS Security Agent".
Cannot access the secret	Grant the service role `secretsmanager:GetSecretValue` on the secret. For instructions on granting access to the secret, see the section called "Grant access to the secret".

Message contains	What to do
Cannot decrypt the secret	Grant the service role `kms:Decrypt` on the secret's customer managed key, and confirm the key is enabled.
Secret not found, or could not be read due to an invalid parameter	Confirm that the Secrets Manager ARN is correct.
is in an invalid state	The secret is pending deletion. Restore the secret, or replace it with a new one.
does not contain a text value	The secret is stored as binary. Store the certificate and key as a JSON `SecretString` in the format shown in the section called "Store the certificate in Secrets Manager".
is not valid JSON, or must be a JSON object	Store the value as a JSON object in the format shown in the section called "Store the certificate in Secrets Manager".
is missing the 'certificate' or 'privateKey' field	Add both fields to the secret.
is not valid PEM-encoded X.509 material, or could not be loaded by the TLS stack	Check the PEM line breaks. A secret created in the Secrets Manager console **Key/value** tab stores \n as literal characters.
The private key in client certificate	Confirm that the key is PEM-encoded and that `privateKeyPassword` is correct, or remove it if the key is not encrypted.
The private key does not match the certificate	Use the private key from the same key pair as the certificate.
has expired, or is not yet valid	Replace the certificate with one that is currently valid.

Message contains	What to do
run window	Use a certificate that stays valid for at least 36 hours after the penetration test starts, or for your maximum task-hours limit if it is lower than 36.

Remediate a penetration test finding

When viewing the findings for a penetration test, you can request AWS Security Agent attempt to remediate a finding. For a connected private repository, AWS Security Agent opens a pull request (or a merge request, for GitLab) with the proposed fix. For Amazon S3 sources, the remediation is available as a downloadable code diff that you can apply locally. AWS Security Agent supports several third-party source code providers — see the section called “How integrations work with Agent Spaces”.

You must enable finding remediation in the AWS Management Console. (See the section called “Enable users to start remediation of penetration test and code review findings”.) Users can start remediation for a specific finding from the AWS Security Agent web application.

Prerequisites

Before you begin, ensure you have:

- A completed or in-progress penetration test run
- Access to the AWS Security Agent web application
- Familiarity with your application’s architecture and security requirements

Step 1: Enable or disable automatic remediation

You can configure code remediation options when you create or modify a penetration test. If you enable automatic remediation, AWS Security Agent automatically attempts to remediate a finding when the agent confirms it during the penetration test. Automatic remediation covers both connected source code repositories and Amazon S3 sources. You can also start code remediation manually with the **Remediate code** button. . In the view to edit **Penetration test details**, in the **Automatic code remediation** section, enable or disable code remediation.

Step 2: Select repositories for code remediation

1. Choose **Next** all the way to the last step **Additional learning resources**.
2. Choose **Select from resources**.
3. Choose your source code provider (for example, **GitHub repositories**).
4. Select the repositories that you want for code remediation.
5. Save the penetration test.
6. You can see the successfully associated repositories under the **Penetration test learning resources** tab.

Step 3: Start a penetration test and view findings

Run the penetration test to detect findings. For more information, see the section called “Review findings from a penetration test”.

Step 4: Start and view code remediation

> **Note**
>
> If a finding is already linked to a Jira issue, AWS Security Agent also adds the remediation pull request as a link on that issue. To make sure the remediation pull request is linked, link the finding to a Jira issue before remediation finishes. For more information, see Create Jira issues for findings.

1. Navigate to the finding.
2. If you've enabled automatic code remediation, a code remediation will be started once AWS Security Agent confirms a finding.
3. If you want to manually start a code remediation, choose the **Remediate code** button.
4. In the **Code Remediation** section of the finding, you can view the code remediation status and links to the pull requests (or merge requests). For Amazon S3 sources, the code remediation is available as a downloadable code diff instead. You can run `git apply /path/to/code_remediation_changes.diff` to apply the change locally.

Schedule a penetration test

In AWS Security Agent, you can schedule a penetration test to run automatically, either once at a date and time that you choose or on a repeating schedule. For example, you can run a penetration test every Monday at 09:00 to retest an application after each weekly release. AWS Security Agent uses Amazon EventBridge Scheduler to start each run.

Before you can schedule a penetration test, an administrator must enable scheduling for the Agent Space in the AWS Management Console. For information about how to enable it, see the section called "(Optional) Enable penetration test scheduling".

A scheduled run uses the configuration that the penetration test has when the run starts. For more information, see the section called "Create a penetration test".

Prerequisites

Before you begin, make sure that you have the following:

- Access to the AWS Security Agent web application
- A penetration test that you have created. You can't schedule a draft penetration test.
- Scheduling enabled for the Agent Space. If it isn't, the schedule page shows the message **Scheduling isn't enabled for this agent space**.

Create a schedule

Each penetration test can have one schedule, either a one-time schedule or a repeating schedule. The time zone defaults to the time zone of your browser.

1. Sign in to the AWS Security Agent web application.
2. Choose **Penetration tests**.
3. Choose the penetration test that you want to schedule.
4. On the penetration test details page, choose **Schedule**, next to **Start run**.
5. On the **Schedule pentest** page, in the **Occurrence** section, choose **Repeating schedule** or **One-time schedule**, and then enter the settings. For descriptions of the settings, see the section called "Schedule settings".
6. In the **Next runs** section, review when the penetration test runs next.

7. Choose **Save**.

The penetration test details page opens, and the **Scheduled runs** table lists the upcoming runs.

Schedule settings

For a repeating schedule, the penetration test runs repeatedly until you remove the schedule. Enter the following settings:

- **Frequency** – Choose **Daily**, **Weekly**, or **Monthly**.
- **Days of the week** – For a weekly schedule, select one or more days.
- **Days of the month** – For a monthly schedule, select one or more days.
- **Time** – Enter the start time in 24-hour `hh:mm` format, and select the time zone.
- **Advanced** – (Optional) To run the schedule only in certain months, expand this section and select the months.

For a one-time schedule, the penetration test runs once, and AWS Security Agent removes the schedule after the run starts. Enter the **Date and time** for the run, and select the time zone. The date and time must be in the future.

View scheduled runs

After you save a schedule, the **Scheduled runs** table appears on the penetration test details page. The table lists each upcoming run with its scheduled start time, the time remaining until it starts, and whether it comes from a one-time or a repeating schedule.

A run appears in the **Penetration test runs** list only after it starts. You can review its progress and findings in the same way as a run that you start yourself. For more information, see the section called “Review findings from a penetration test”.

Edit or remove a schedule

1. Sign in to the AWS Security Agent web application.
2. Choose **Penetration tests**.
3. Choose the penetration test.
4. On the penetration test details page, choose **Edit schedule**.

5. In the **Occurrence** section, do one of the following:
   - To change the schedule, update the settings. To switch between a one-time schedule and a repeating schedule, choose the other option.
   - To remove the schedule, choose **No schedule**.
6. Choose **Save**.

Removing a schedule doesn't stop a run that has already started.

Run penetration tests from your CI/CD pipeline

> ⚠ **Public preview**
>
> CI/CD pipeline integration is in preview and is subject to change.
> While it is in preview, AWS doesn't charge for continuous pentest runs, however you start them. You still pay for the other AWS resources you use, such as hosting the application you test. Other penetration tests are metered as usual. For more information, see the section called "Pricing and billing".

You can run an AWS Security Agent penetration test as a **post-deployment gate** in your continuous integration and continuous delivery (CI/CD) pipeline. You add the integration as a pipeline step that runs *after* a deployment. The step tests the change that was just deployed and can block promotion to the next stage (for example, to production) when a finding is at or above a severity threshold that you set.

Each deploy gets security coverage automatically. You don't have to define the test scope or wait for a scheduled assessment, because AWS Security Agent scopes each run to the change that the pipeline deployed.

> ⚠ **This is a post-deployment gate, not a pull request check**
>
> This integration tests code **after** it is deployed to a running environment, and it scopes the test to the deployed change — a commit range that can span more than one pull request. It is different from pull request code review, which scans the changes in an individual pull request before they merge. If you want to scan pull requests as they are opened, see the section called "Review code security findings in pull requests" and the section called "Run a

differential code scan with S3" instead. Use this CI/CD pipeline integration when you want to dynamically test a **deployed** application and gate promotion on the result.

AWS Security Agent provides an integration for common CI/CD platforms:

- GitHub Actions
- GitLab CI/CD
- Bitbucket Pipelines
- Azure DevOps Pipelines

All testing runs server-side in your AWS account. The pipeline step only starts the job, waits for the result, and reports the outcome to your pipeline; no penetration testing logic runs on your CI/CD runner.

Before you start

The pipeline integration starts a job for a penetration test that you have already configured. It does not create the target or the test for you. Complete this one-time setup first:

1. Verify the target domain you want to test. See the section called "Enable an application domain for penetration testing".
2. Create a penetration test in your Agent Space and configure its target, credentials, and network scope. See the section called "Create a penetration test". The pipeline tests whatever this penetration test is configured to test.
3. Enable CI/CD mode on the penetration test, and note the Agent Space ID (`as-...`) and penetration test ID (`pt-...`).
4. Connect your repository as a source control integration so the pipeline can resolve your repository to its AWS Security Agent integration.
5. Set up AWS authentication with OIDC and add the pipeline step, as described in the following sections. For what differs between the four providers, see the section called "How the integration differs by provider".

> **Note**
>
> AWS Security Agent is available in a subset of AWS Regions. Configure your pipeline to use a Region where your Agent Space and penetration test are located.

How the pipeline integration scopes the test

When the pipeline step runs, AWS Security Agent determines the commit range to test and scopes the penetration test to that range:

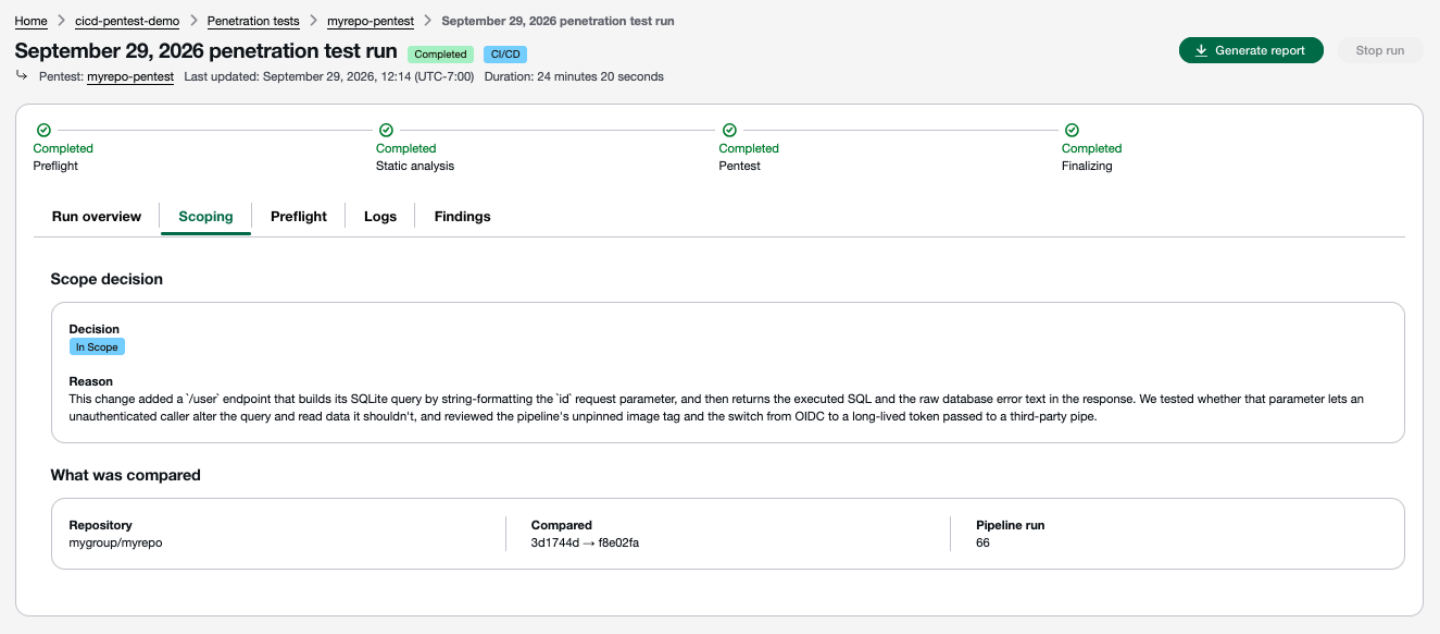
- **Head** – the commit being gated. The integration derives it from the event that triggered the pipeline.
- **Base** – the previous commit that the gate already cleared. The integration tracks this automatically and falls back to a trigger-specific source (such as the previous deployment) when there is no usable record. It never falls back to a full-scope test.

You don't normally supply commit identifiers. AWS Security Agent computes the difference server-side and validates the range, so you can't widen or evade the configured scope by changing the base that the pipeline tracks. Each integration does accept explicit base and head commit inputs, for the cases where the automatic range isn't the one you want; see the section called "Pin the commit range explicitly".

When the test completes, AWS Security Agent returns a **scope decision** that tells the pipeline how to proceed:

- **In scope** – the change modifies the tested attack surface. Findings are evaluated against your severity threshold, and any finding at or above it blocks promotion.
- **Scoped out** – the change is within the configured scope but does not modify the attack surface. There is nothing to test, and the pipeline passes.
- **Scope conflict** – the change falls outside the penetration test's configured scope, so it was not tested. You choose whether this blocks the pipeline. For more information, see the section called "Gate on scope conflicts".

Every run records its decision. The **Scoping** tab of a run in the AWS Security Agent console shows the decision, the reasoning behind it, and the commit range that was compared:



How the pipeline integration authenticates

The pipeline integration does not use long-lived AWS access keys. Instead, it uses OpenID Connect (OIDC) federation: your CI/CD provider issues a short-lived OIDC token that identifies the pipeline, and AWS Security Token Service (AWS STS) exchanges that token for temporary credentials by having the pipeline assume an IAM role that you create in your account. The temporary credentials are used only for the duration of the pipeline step.

The security of the integration depends on two things you control:

- The IAM role's **trust policy**, which decides *which* pipelines can assume the role.
- The IAM role's **permissions policy**, which decides *what* the pipeline can do after it assumes the role.

The following sections describe how to scope both to the principle of least privilege.

Scope the trust policy to your pipeline

First, create an IAM OIDC identity provider for your CI/CD provider. The provider URL and audience differ by provider, and on Bitbucket and Azure DevOps the URL contains an identifier specific to your workspace or organization. The following section gives the value for each provider.

Then create the IAM role with a trust policy that federates that provider and restricts which repository can assume the role by matching on the token's `sub` (subject) claim. Never wildcard

the part of the claim that identifies the repository, project, or service connection: a `sub` condition such as `repo:*` lets *any* repository on that provider — including forks and repositories in other organizations — assume your role and start penetration tests against your targets.

A trailing wildcard that covers only the ref or step portion of the claim, such as `repo:my-org/my-app:*`, is a different matter. The repository is still fixed, so the policy admits only that one repository. Tighten it to a specific branch or environment where your provider's claim allows it, and note that not every provider does. On Bitbucket you cannot scope to a branch at all, and repository scope, optionally plus a deployment environment, is as tight as the claim allows.

The following example trust policy allows only the `main` branch of the `my-org/my-app` repository on GitHub Actions to assume the role. Replace the account ID, organization, repository, and branch with your own values.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/
token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "token.actions.githubusercontent.com:sub": "repo:my-org/my-app:ref:refs/
heads/main"
        }
      }
    }
  ]
}
```

The claim name and the format of the `sub` value differ by provider. The following table gives both for each of the four.

CI/CD provider	OIDC provider URL and audience	OIDC subject (sub) scoping
GitHub Actions	`https://token.actions.githubusercontent.com` with audience `sts.amazonaws.com`.	Scope by repository, for example `repo:my-org/my-app:*`. Tighten to a ref, for example `repo:my-org/my-app:ref:refs/heads/main`, or to an environment, for example `repo:my-org/my-app:environment:production`.
GitLab CI/CD	`https://gitlab.com`, or your instance URL on self-managed GitLab. The audience is the value you set for the component's `oidc-aud` input, which defaults to `https://gitlab.com`.	Scope by project path, for example `project_path:my-group/my-app:*`. GitLab's claim carries the ref, so you can pin a branch, as GitHub also allows. GitLab spells the ref type out: `project_path:my-group/my-app:ref_type:branch:ref:main`.
Bitbucket Pipelines	`https://api.bitbucket.org/2.0/workspaces/my-workspace/pipelines-config/identity/oidc`, so each workspace is a separate provider. The audience is `ari:cloud:bitbucket::workspace/` followed by the workspace UUID **without** braces.	Scope by repository UUID, **with** braces: `{REPO_UUID}:*`. There is no branch claim. Declaring a deployment environment on the step adds an environment UUID you can also match on.
Azure DevOps Pipelines	`https://vstoken.dev.azure.com/` followed by your organization ID, which is a GUID rather than the organization name. The audience is `api://AzureADTokenExchange`.	The subject is the service connection, not a repository: `sc://my-org/my-project/my-connection-name`. Both claims are exact, so both belong under `StringEquals`.

Immutable subject claims on GitHub

Repositories that GitHub created or renamed on or after July 15, 2026 (and older repositories that opt in) emit an *immutable* `sub` claim that appends numeric organization and repository IDs after each name, for example `repo:my-org@123456/my-repo@789012:ref:refs/heads/main`. If your trust policy matches only the name-only form, `AssumeRoleWithWebIdentity` fails. Confirm the exact `sub` value that your workflow emits before you write the trust policy.

Grant least-privilege permissions

Attach a permissions policy that allows only the actions the pipeline step performs: starting the penetration test job, polling its status, reading findings, stopping an abandoned job during cleanup, and resolving the repository integration.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "securityagent:StartPentestJob",
        "securityagent:BatchGetPentestJobs",
        "securityagent:StopPentestJob",
        "securityagent:ListFindings",
        "securityagent:BatchGetFindings",
        "securityagent:ListIntegratedResources"
      ],
      "Resource": "*"
    }
  ]
}
```

Each action supports a specific part of the pipeline step:

Action	Why the pipeline step needs it
`securityagent:StartPentestJob`	Start the penetration test job for the deployed commit range.
`securityagent:BatchGetPentestJobs`	Poll the job until it reaches a terminal state.
`securityagent:StopPentestJob`	Stop an abandoned job if the pipeline is cancelled or times out before the job finishes.
`securityagent:ListFindings` , `securityagent:BatchGetFindings`	Read findings to evaluate them against your severity threshold.
`securityagent:ListIntegratedResources`	Resolve your repository to its AWS Security Agent integration.

Note

Some of these actions are list operations that do not support resource-level permissions, so the example uses `"Resource": "*"`. Where an action supports resource-level permissions, you can scope it to your specific Agent Space or penetration test ARN in a separate statement. For which actions support resource-level permissions, see the AWS Security Agent API Reference. Do not store AWS access keys or session tokens as CI/CD variables for this integration; OIDC federation removes the need for stored credentials.

These six actions are the complete set: the pipeline step makes no other AWS Security Agent call, and removing any one of them breaks a step that the gate performs. Do not add an `sts:` action to this policy. `sts:GetCallerIdentity`, which the credential step uses to confirm which identity it assumed, requires no permission at all, and `sts:AssumeRoleWithWebIdentity` is granted by the role's **trust** policy rather than by what the role can do once assumed.

How the integration differs by provider

The remaining setup is specific to your CI/CD provider: the OIDC provider URL and audience, how the pipeline obtains credentials, and how the gate records the commit it last cleared. The settings you pass the gate are the same everywhere, but each provider spells their names differently.

CI/CD provider	How the integration runs, and where it keeps the baseline
GitHub Actions	Runs as an action. A prior `aws-actions/configure-aws-credentials` step supplies credentials, and the baseline is a git ref in your repository.
GitLab CI/CD	Runs as a component. The component performs the credential exchange itself, and the baseline is a tag in your project — one tag per cleared commit, rather than a single moving record. Four separate timeouts have to agree; see the section called "Cancellation and cleanup".
Bitbucket Pipelines	Runs as a pipe on Bitbucket Cloud. The step must export the role ARN and write the OIDC token to a file before the pipe runs, and the baseline is a repository variable rather than anything in git history.
Azure DevOps Pipelines	Runs as a task from the AWS Toolkit for Azure DevOps, which an organization administrator installs. Credentials come from an AWS service connection, and the baseline is a repository ref for an Azure Repos source or a build tag for a repository hosted elsewhere.

Whichever provider you use, run the gate in a step that runs *after* your pre-production deployment, and gate promotion to production on its result.

> **Note**
>
> Pin the integration to an immutable version — a release tag or commit SHA for a GitHub Action, a pinned component version for GitLab, or a pinned tag or image digest for a

container-based integration — so that a change to the upstream integration cannot alter what runs in your pipeline without your review.

Defaults the integration ships with

The settings are named slightly differently on each provider, but they map to the same concepts and share the same defaults. The defaults are identical across all four integrations.

Setting	Default	What the default means
Severity threshold	`HIGH`	Findings at `HIGH` or `CRITICAL` block promotion. Lower findings are reported and do not block.
Timeout	60 minutes	The step waits up to an hour for the penetrati on test job to reach a terminal state.
On scope conflict	`block`	A change that falls outside the penetration test's configured scope fails the step, because it was not tested.
Fail on infrastructure error	`false`	**The shipped default is fail-open.** If the job cannot start or complete — for a reason that is not a finding and not a misconfiguration the integration recognizes — the step passes with a warning. Set it to `true` for any pipeline whose promotion requires security sign-off.
Dry run	`false`	The step starts a real penetration test job.

Important

Because the default is fail-open on infrastructure errors, a gate you have not validated can pass without testing anything. Run a dry run first, as described in the section called "Validate your setup before you trust the gate", and set fail-on-error to `true` if you rely on the gate to hold a release.

Validate your setup before you trust the gate

Because the gate can be configured to pass the pipeline on infrastructure errors (see the section called "Choose how the pipeline responds"), a misconfiguration — a bad OIDC trust policy, the wrong Region, or a repository that is not connected as an integration — can otherwise pass silently. Run the integration once in dry-run mode to confirm the setup end to end before you rely on it to gate promotion.

In dry-run mode, the integration checks AWS authentication, the Region, and that your repository resolves to its integration, then exits *without* starting a penetration test job. If validation fails, the step fails regardless of your error-handling setting, so the misconfiguration is surfaced.

Choose how the pipeline responds

Decide how the pipeline step should behave for each outcome. The following table summarizes the outcomes and the settings that control them:

Outcome	Pipeline behavior
Findings at or above your severity threshold	The step fails and blocks promotion. Set the threshold to `CRITICAL`, `HIGH`, `MEDIUM`, or `LOW`. The default is `HIGH`.
No findings at or above the threshold, or a scoped-out change	The step passes.
Any finding, with the threshold set to `NONE`	The step passes. The penetration test still runs and every finding is still collected, published, and counted, but no finding fails the step. See the section called "Report-only runs".
Scope conflict (change outside the configured scope)	You choose: block the pipeline, or pass with a warning. The default is to block. See the section called "Gate on scope conflicts".
Infrastructure error (the job could not start or complete)	You choose fail-closed (fail the step) or fail-open (pass the step). **The default is fail-open.** For changes that require security sign-off, choose fail-closed. A fail-open run also advances the baseline on most providers; see the section called "What advances the baseline, and why that matters".

Outcome	Pipeline behavior
AWS Security Agent rejects the request as unauthorized (HTTP 401 or 403)	The step always fails, regardless of your error-handling setting, so a permissions misconfiguration cannot silently allow a promotion. All four integrations behave this way.
A misconfiguration the integration recognizes, such as a repository that is not connected	The step always fails, regardless of your error-handling setting.

> **Note**
>
> The always-fail rule covers rejections from the AWS Security Agent control plane. A credential problem that fails earlier — when your pipeline exchanges its OIDC token for AWS credentials, before the gate calls AWS Security Agent — is handled by your CI/CD provider's own credential step, and what it does to the pipeline depends on the provider.

Three settings fail the step independently

The three settings that decide whether the step fails are independent of each other, and each covers a different kind of outcome:

- **Severity threshold** decides what happens to **findings**.
- **On scope conflict** decides what happens when the change was **not tested** because it falls outside the configured scope.
- **Fail on infrastructure error** decides what happens when the job **could not run**.

Setting the severity threshold to `NONE` relaxes only the first. A scope conflict still fails the step while on-scope-conflict is `block`, and an infrastructure error still fails it while fail-on-error is `true`. To make the step report without ever failing the pipeline, relax all three: set the threshold to `NONE`, on-scope-conflict to `warn`, and fail-on-error to `false`.

What advances the baseline, and why that matters

The gate records the commit it last cleared, and the next run tests only what changed after that commit. Any run that **passes** normally advances that record — including, on most providers, a run that passed without assessing anything.

That is intended. Relaxing a setting is you accepting the outcome it allows: a report-only run passes because you chose not to gate on findings, and a fail-open run passes because you chose to accept an infrastructure error. In both cases the commits in that run's range are recorded as cleared.

The consequence is the same in both cases, and it is easy to miss: **those commits are not tested again.** If you later tighten the threshold, or set fail-on-error to `true`, the gate assesses only commits after the last recorded one. Raising the threshold does not revisit what a report-only run already accepted.

Run outcome	Advances the baseline
Passes with no finding at or above the threshold, or the change was scoped out	Yes.
Passes under threshold `NONE` with findings reported	Yes, on all four providers.
Passes with a scope conflict while on-scope-conflict is `warn`	Yes on GitHub Actions, GitLab CI/CD, and Azure DevOps Pipelines. **Not** on Bitbucket Pipelines, for the same reason as the next row: the change was never tested.
Passes fail-open after an infrastructure error	Yes on GitHub Actions, GitLab CI/CD, and Azure DevOps Pipelines. **Not** on Bitbucket Pipelines, which advances the baseline only when AWS Security Agent actually reached a verdict on the range, and warns when it does not.
Fails for any reason — a finding at or above the threshold, a scope conflict while on-scope-conflict is `block`, a timeout, an authorization denial, or an	No.

Run outcome	Advances the baseline
infrastructure error while fail-on-error is `true`	
Runs in dry-run mode	No.
Pins the head commit explicitly	No, on any outcome. A pinned commit isn't necessarily newer than the recorded one, so advancing to it could move the baseline backward.

To reassess a range that an earlier run already cleared, pin the base commit on the run that should re-test it. See the section called "Pin the commit range explicitly".

Pin the commit range explicitly

All four integrations take the base and head commits as inputs, for when the automatic range isn't the one you want: reassessing a range an earlier run already cleared, or gating a deployment whose artifact was built from a commit other than the one the pipeline checked out. Set the base to the commit **before** the range you want tested. The integration uses the value as given, and doesn't check that the base is an ancestor of the head, so a base unrelated to the head produces a correspondingly odd range.

Pinning the base on its own is the usual case, and the head is still derived from the triggering event. Pinning the head on its own is the riskier choice, because the base still resolves through its normal sequence. The recorded baseline and the trigger-specific source are both relative to the current line, so either can sit ahead of a head you pinned at an older commit, which reverses the range. Only the last step reads from the commit you pinned, by taking its parent. Azure DevOps Pipelines refuses that one combination outright, before it starts a job: there, pin the base alone or pin both, never the head alone.

CI/CD provider	Pinning the commit range
GitHub Actions	`base-commit-sha` and `head-commit-sha`, set under the step's `with:` block. Each takes a full 40-character commit hash.

CI/CD provider	Pinning the commit range
GitLab CI/CD	`base-commit-sha` and `head-commit-sha`, set under `inputs:` where you `include:` the component. Each takes a full 40-character commit hash.
Bitbucket Pipelines	`BASE_COMMIT_SHA` and `HEAD_COMMIT_SHA`, in the Pipe's `variables:` block. Each takes a full 40-character commit hash, and the commit needs to be present in the clone — raise `clone: depth` if it's older than the default depth. See the section called "Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test".
Azure DevOps Pipelines	`baseCommitSha` and `headCommitSha` task inputs, each a full 40-character commit hash. `headCommitSha` requires `baseCommitSha`. See the section called "Tutorial: Gate an Azure DevOps deployment with a scoped penetration test".

A pinned head commit doesn't advance the baseline, whatever the run's outcome. Pinning only the base doesn't change that: a passing run still advances the baseline to the commit the pipeline built.

On GitLab, the committed settings are not the gate's contract

On GitLab CI/CD, a pipeline variable overrides the value a job sets in its own `variables:` block. The gate's settings are passed that way, so if you can run a pipeline, you can supply a pipeline variable that lowers the severity threshold, turns on dry-run mode, or relaxes error handling — and the gate then does what the variable says, not what your `.gitlab-ci.yml` says.

The settings you commit are therefore the gate's **defaults** rather than its contract, unless you also restrict who may override variables. In your project's CI/CD settings, set the minimum role required to override variables on a pipeline to a role you trust with that decision. Restrict it before you rely on the gate to hold a release.

Report-only runs

Setting the severity threshold to `NONE` gives you a report-only gate: the penetration test still runs and every finding is still collected, published, and counted, but no finding fails the step. This is a useful way to see what the integration finds on real deployments before you let it block a release.

`NONE` relaxes findings only. A scope conflict and an infrastructure error still fail the step on their own settings, as described in the section called "Three settings fail the step independently".

> **Warning**
>
> A report-only run that passes advances the baseline, so the commits it reported on are not assessed again if you later raise the threshold. See the section called "What advances the baseline, and why that matters".

Gate on scope conflicts

AWS Security Agent scopes each pipeline-triggered penetration test to the change that the pipeline deployed. When the change falls outside the penetration test's configured scope, AWS Security Agent reports a *scope conflict* and does not test the change, rather than silently testing the wrong surface.

You choose how strict to be. With the pipeline integration, you can treat a scope conflict as a pipeline-blocking condition — fail the step and require someone to review the scope — or as a non-blocking signal that records the result and lets the pipeline continue. For a change that requires security sign-off, treat a scope conflict as blocking.

Cancellation and cleanup

A penetration test job runs server-side, and it keeps running until it finishes even if your pipeline stops waiting for it. A job left running continues to act against your target, and it holds one of your account's concurrent pentest runs. When the pipeline step reaches its own timeout, the integration asks AWS Security Agent to stop the abandoned job. This cleanup uses the `securityagent:StopPentestJob` and `securityagent:BatchGetPentestJobs` permissions in the policy above, and it never fails the pipeline step.

> **Cancelling the pipeline does not reliably stop the job**
>
> How much the integration can do when you **cancel** a pipeline depends on what your CI/CD provider lets a step do once cancellation begins, and it is not the same on all four:

CI/CD provider	What happens when you cancel
GitHub Actions	The action runs a post-job step during GitHub's cancellat ion grace period, which stops the job.
GitLab CI/CD	The component's `after_script` runs and stops the job. It cannot run if your project's maximum job timeout is lower than the gate's own timeout, because the runner then kills the job first — see the duration guidance later in this section.
Bitbucket Pipelines	There is no post-step hook for a pipe. The pipe asks to stop the job from inside the step when it is signalled, but a cancelled pipeline has been observed to leave the job running to completion. Treat a cancellation as leaving the job running.
Azure DevOps Pipelines	There is no post-step hook. The task stops the job from inside the step, on both cancellation and its own timeout.

Where cancellation cannot stop the job, **stop it yourself** from the penetration test's page in the AWS Security Agent console. Until it reaches a terminal state it continues to act against your target and to hold one of your account's concurrent pentest runs. For more information about that quota, see the section called "Service Quotas".

The credentials the pipeline assumes do not refresh while the step runs, so their lifetime has to cover the whole penetration test rather than only the call that starts it. Set the IAM role's maximum session duration to at least `(timeout + 10) * 60` seconds — 4,200 seconds for the default 60-minute timeout — and, on the providers that let you request a lifetime, request that much. AWS STS **rejects** a request for more than the role allows rather than reducing it, so a role whose maximum is too low fails the credential exchange outright. A credential that expires partway through instead makes an otherwise healthy run fail as though the service were at fault, and it leaves the cleanup unable to stop the abandoned job.

Where you request that lifetime differs by provider: GitLab's component derives it from the gate timeout, GitHub takes it on the credential step, and Azure DevOps takes it as a pipeline variable. On GitLab the role's maximum session duration is one of four durations that have to agree — the gate timeout, the gate job's own timeout, the project's maximum job timeout, and the role's session ceiling — and the project maximum caps the other two.

Protect credentials on the runner

The temporary AWS credentials the pipeline obtains, and the OIDC token used to obtain them, are sensitive. Anyone who can read them while the pipeline step runs can act as your pipeline. Follow these practices on your CI/CD runner:

- **Keep debug and verbose tracing off for steps that handle credentials.** Debug or shell-trace modes (for example, GitLab CI's `CI_DEBUG_TRACE`) echo command arguments and environment values to the job log in plaintext, which can expose a token or temporary credentials.
- **Do not persist source control credentials in the workspace.** When your pipeline checks out source, disable credential persistence (for example, set `persist-credentials: false` on the checkout step in GitHub Actions) so that a token is not written to the runner's `.git` configuration where a later step could read it.
- **Do not expose the OIDC token or credentials to untrusted steps.** Grant permission to request the OIDC token only to the jobs that need it, and do not run untrusted code (such as a build from a fork) in a job that has access to the token or the assumed role.

Worked examples by provider

The following topics walk through a complete, reproducible setup for each supported CI/CD platform, using an intentionally vulnerable sample application so that you can see the gate block a real finding:

- the section called "Tutorial: Gate a GitHub Actions deployment with a scoped penetration test"
- the section called "Tutorial: Gate a GitLab CI/CD deployment with a scoped penetration test"
- the section called "Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test"
- the section called "Tutorial: Gate an Azure DevOps deployment with a scoped penetration test"

Related security topics

- the section called “Create a penetration test” – configure the penetration test that the pipeline starts, including target URLs, credentials, and network scope.
- the section called “Security best practices” – general best practices, including testing against non-production environments and validating AI-generated findings.
- the section called “Identity and access management” – how AWS Security Agent works with IAM.
- the section called “Cross-service confused deputy prevention” – how AWS Security Agent helps prevent the confused deputy problem, where a less-privileged caller tricks a more-privileged service into acting on its behalf.

Tutorial: Gate a GitHub Actions deployment with a scoped penetration test

Public preview

CI/CD pipeline integration is in preview and is subject to change.
While it is in preview, AWS doesn’t charge for continuous pentest runs, however you start them. You still pay for the other AWS resources you use, such as hosting the application you test. Other penetration tests are metered as usual. For more information, see the section called “Pricing and billing”.

This tutorial walks through a complete, reproducible setup that runs a continuous, scoped AWS Security Agent penetration test as a **post-deployment gate** in GitHub Actions. You deploy an intentionally vulnerable sample application to a staging environment, then let AWS Security Agent test the deployed change and block promotion to production when it finds a vulnerability.

Note

Hosting the sample application might result in charges to your AWS account.

For the concepts behind the integration — how the test is scoped to the deployed change, how OIDC authentication works, and how gating decisions are made — see the section called “Run

penetration tests from your CI/CD pipeline". This tutorial applies those concepts to a concrete GitHub repository.

> **Intentional vulnerability**
>
> The sample application in this tutorial contains a deliberate SQL injection vulnerability so that the gate has a real finding to block on. Deploy it only to an isolated, non-production environment that you control, and remove it when you finish. Do not expose it to the public internet or connect it to production data.

Prerequisites

Before you start, make sure you have the following:

- An AWS account, and permission in it to create an IAM OIDC identity provider, an IAM role, and AWS Security Agent resources.
- A GitHub repository that you own, on `github.com`, in which you can add workflows and configure environments. This tutorial uses the placeholder repository `my-org/my-app`; substitute your own repository throughout.
- A staging environment that the sample application can deploy to, reachable from the network where your penetration test runs (the public internet, or a VPC in your AWS account for a private target). See the section called "Connect agent to private VPC resources" for private targets.
- The AWS Security Agent one-time setup described in the section called "Run penetration tests from your CI/CD pipeline" under **Before you start**: a verified target domain, a CI/CD-enabled penetration test, your GitHub repository connected as a source control integration, and the Agent Space ID (`as-...`) and penetration test ID (`pt-...`) that identify them.

Throughout this tutorial, replace the following placeholders with your own values:

Placeholder	Replace with
`111122223333`	Your 12-digit AWS account ID.
`AGENT_SPACE_ID (as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)`	Your Agent Space ID.

Placeholder	Replace with
`PENTEST_ID (pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx )`	Your CI/CD-enabled penetration test ID.
`my-org/my-app`	Your GitHub organization and repository.
`us-west-2`	The AWS Region where your Agent Space and penetration test are located.

Step 1: Create the vulnerable sample application

Create a small Flask application with one endpoint that builds a SQL query by string concatenation — a classic SQL injection. In your repository, add `app.py`:

```
import sqlite3
from flask import Flask, request

app = Flask(__name__)


@app.route("/products")
def products():
    # Intentionally vulnerable: user input is concatenated directly into SQL.
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = '" + category + "'"
    rows = cursor.execute(query).fetchall()
    conn.close()
    return {"products": rows}


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Add a `deploy.sh` script that deploys this application to your staging environment. The details depend on your infrastructure; the gate does not require any particular deployment mechanism, only that the application is running and reachable when the test runs.

Step 2: Set up GitHub OIDC and an IAM role

The workflow authenticates to AWS with GitHub OIDC — no long-lived access keys. Do this once in your AWS account.

1. If you do not already have one, create the GitHub OIDC identity provider:

```
aws iam create-open-id-connect-provider \
  --url https://token.actions.githubusercontent.com \
  --client-id-list sts.amazonaws.com
```

2. Create the IAM role with a trust policy scoped to your repository and branch. Save the following as `trust-policy.json`, replacing the account ID and repository:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/
token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "token.actions.githubusercontent.com:sub": "repo:my-org/my-app:ref:refs/
heads/main"
        }
      }
    }
  ]
}
```

   For the reasoning behind scoping the `sub` claim and the immutable-subject caveat for newer repositories, see the section called “Scope the trust policy to your pipeline”.

3. Save the least-privilege permissions policy as `permissions.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "securityagent:StartPentestJob",
        "securityagent:BatchGetPentestJobs",
        "securityagent:StopPentestJob",
        "securityagent:ListFindings",
        "securityagent:BatchGetFindings",
        "securityagent:ListIntegratedResources"
      ],
      "Resource": "*"
    }
  ]
}
```

For what each action does, see the section called “Grant least-privilege permissions”.

4. Create the role and attach the policy. A penetration test job can run for tens of minutes, so create the role with a maximum session duration long enough to outlast the step's timeout:

```
aws iam create-role --role-name my-pentest-gate-role \
  --assume-role-policy-document file://trust-policy.json \
  --max-session-duration 7200

aws iam put-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest \
  --policy-document file://permissions.json
```

The role ARN for the workflow is `arn:aws:iam::111122223333:role/my-pentest-gate-role`.

Step 3: Add the workflow

Add `.github/workflows/deploy-and-pentest.yml` to your repository. The `deploy` job deploys the sample application to staging; the `security-pentest` job then runs the gate and, because production promotion depends on it, blocks production when the gate fails.

```
name: Deploy and pentest
on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    environment: staging
    steps:
      - uses: actions/checkout@v4
        with:
          persist-credentials: false
      - run: ./deploy.sh staging

  security-pentest:
    needs: [deploy]
    runs-on: ubuntu-latest
    permissions:
      # Required for the OIDC credential exchange.
      id-token: write
      # Write rather than read: the gate records the commit it last cleared as a
      # ref in this repository and reads it back to test only what changed since.
      # With read access the marker never advances and every run tests full scope.
      contents: write
      # Lets the gate find the last successful run of this workflow to use as the
      # base commit. Without it that lookup is denied and the gate falls back to
      # the parent commit, testing a narrower range than intended.
      actions: read
      # Same, for deployment and deployment_status triggers, where the base is
      # the last successful deployment.
      deployments: read
    steps:
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::111122223333:role/my-pentest-gate-role
          aws-region: us-west-2
          role-duration-seconds: 7200
      - uses: aws-actions/aws-continuum-run-pentest@v1
        with:
          agent-space-id: as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
          pentest-id: pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
          severity-threshold: HIGH
```

```
          on-scope-conflict: block
          timeout-minutes: "45"

  deploy-prod:
    needs: [security-pentest]
    runs-on: ubuntu-latest
    environment: production
    steps:
      - uses: actions/checkout@v4
        with:
          persist-credentials: false
      - run: ./deploy.sh production
```

Because `deploy-prod` declares `needs: [security-pentest]`, GitHub skips it whenever the gate job fails, so a change with a `HIGH` (or higher) finding never reaches production.

The `security-pentest` job's four workflow permissions each serve a specific part of the gate:

- `id-token: write` mints the OIDC token for the credential exchange.
- `contents: write` lets the gate read **and advance** the baseline marker it keeps as a ref in your repository. Read access is not enough: the marker would never advance, and every run would test a wider range than intended.
- `actions: read` lets the gate use the last successful run of this workflow as the base commit when it has no marker to read.
- `deployments: read` does the same for `deployment` and `deployment_status` triggers, where the base is the last successful deployment.

Grant the last two even though this tutorial's first run has no marker yet. They are what the documented fallbacks need, and without them the lookup is denied and the gate silently falls back to the parent commit — testing one commit instead of the range you deployed.

This tutorial does not publish findings to GitHub's Security tab, so it does not grant `security-events: write`. Add that permission only if you also set the `upload-sarif` input. The upload runs only on a change that was tested and produced findings, so enabling it on a scoped-out or findings-free run does nothing.

> **Note**
>
> Pin `aws-actions/aws-continuum-run-pentest` to a specific commit SHA (for example, `aws-actions/aws-continuum-run-pentest@<commit-sha>`) rather than a moving tag, so that a change to the upstream action cannot alter what runs in your pipeline without your review. See the section called "How the integration differs by provider".

Step 4: Validate your setup with a dry run

Before you rely on the gate, run it once in dry-run mode. Add `dry-run: "true"` to the action inputs and trigger the workflow:

```
      - uses: aws-actions/aws-continuum-run-pentest@v1
        with:
          agent-space-id: as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
          pentest-id: pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
          dry-run: "true"
```

The dry run checks AWS authentication, the Region, and that your repository resolves to its integration — without starting a penetration test job. If it fails, it surfaces the common setup mistakes (a bad OIDC trust policy, a Region mismatch, or a repository that is not connected) in seconds. When it passes, remove the `dry-run` input and continue. For more information, see the section called "Validate your setup before you trust the gate".

Step 5: Trigger the gate and read the result

Push a commit to `main` that changes the vulnerable endpoint. This triggers the workflow: the `deploy` job deploys the change to staging, then the `security-pentest` job starts a penetration test scoped to that change.

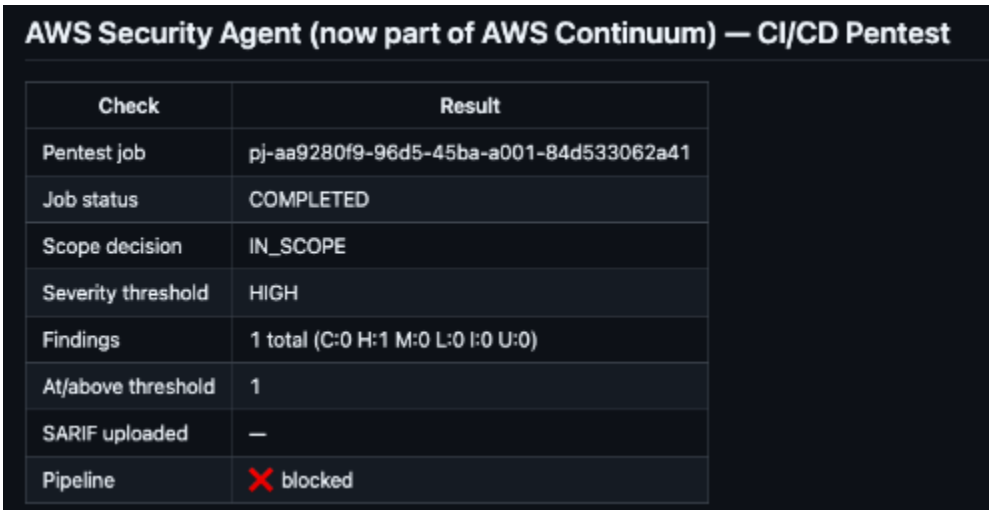
Open the workflow run under the repository's **Actions** tab and watch the `security-pentest` job:

1. **Configure AWS credentials** – GitHub mints an OIDC token and the runner assumes your role. No stored secrets are involved.
2. **Run AWS Security Agent pentest** – the action resolves your repository, starts a scoped job, polls it to completion, and evaluates the findings against your `severity-threshold`.

A scoped penetration test typically runs for tens of minutes, so the step sits **in progress** for a while. That is expected, not a hang. The step ends when the job reaches a terminal state or the `timeout-minutes` elapses.

Because the deployed change introduced a SQL injection vulnerability, AWS Security Agent returns an **in scope** decision with a `HIGH` (or higher) finding. The `security-pentest` job fails, and because the production job depends on it, promotion to production is blocked.

The job summary records the verdict: the scope decision, the severity threshold the run was evaluated against, the finding counts, and whether the gate blocked.



The action also sets outputs you can consume in later steps, including `scope-decision` (`IN_SCOPE`, `SCOPED_OUT`, or `SCOPE_CONFLICT`), `status`, `blocking-findings-count` (the findings at or above your threshold — the ones that fail the step), and `total-findings-count` (every finding the run inspected, at any severity). Both counts are absent on a run that never evaluated findings, such as a dry run, so read `status` and `scope-decision` to tell that apart from a run that genuinely found nothing. For the full gating behavior — including how authentication failures always block and how you choose fail-closed or fail-open for infrastructure errors — see the section called “Choose how the pipeline responds”.

Step 6: Fix the finding and confirm the gate passes

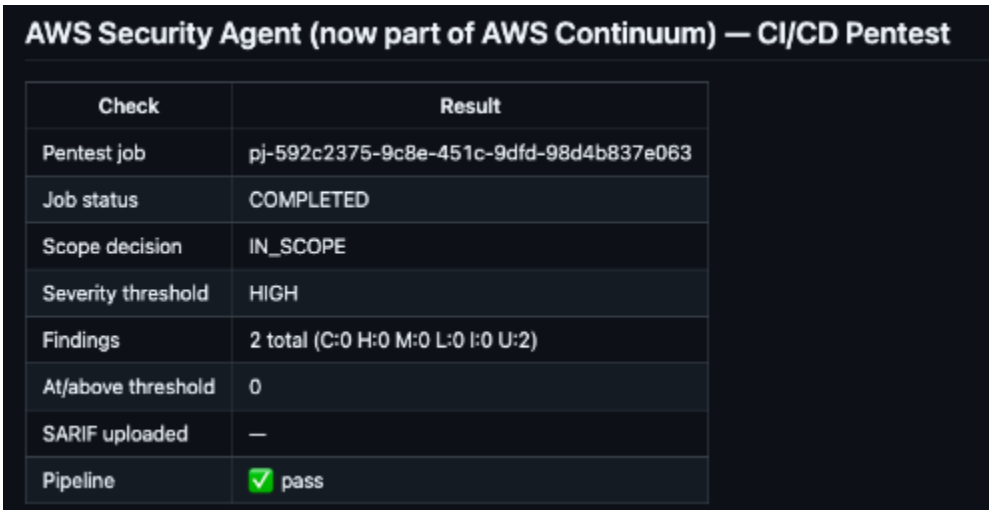
Fix the vulnerability by using a parameterized query instead of string concatenation. In `app.py`, replace the query construction:

```
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
```

```
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = ?"
    rows = cursor.execute(query, (category,)).fetchall()
    conn.close()
    return {"products": rows}
```

Push the fix to `main`. The workflow redeploys the change and runs the gate again against the new commit. With the injection removed, AWS Security Agent finds no findings at or above `HIGH`, the `security-pentest` job passes, and promotion to production proceeds.

The job summary shows the same checks for the new commit, with no findings at or above the threshold and a passing verdict.



Clean up

To avoid ongoing cost and to remove the intentionally vulnerable application:

1. Tear down the sample application from your staging environment.
2. Delete the workflow file, or disable it, if you do not intend to keep the gate.
3. If you created the IAM role and OIDC provider only for this tutorial, delete them:

```
aws iam delete-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest
aws iam delete-role --role-name my-pentest-gate-role
aws iam delete-open-id-connect-provider \
  --open-id-connect-provider-arn arn:aws:iam::111122223333:oidc-provider/
token.actions.githubusercontent.com
```

Related topics

- the section called “Run penetration tests from your CI/CD pipeline” – concepts and reference for the CI/CD pipeline integration.
- the section called “Tutorial: Gate a GitLab CI/CD deployment with a scoped penetration test”, the section called “Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test”, the section called “Tutorial: Gate an Azure DevOps deployment with a scoped penetration test” – the same tutorial for other CI/CD platforms.
- the section called “Security best practices” – general best practices, including testing against non-production environments.

Tutorial: Gate a GitLab CI/CD deployment with a scoped penetration test

Public preview

CI/CD pipeline integration is in preview and is subject to change.
While it is in preview, AWS doesn't charge for continuous pentest runs, however you start them. You still pay for the other AWS resources you use, such as hosting the application you test. Other penetration tests are metered as usual. For more information, see the section called “Pricing and billing”.

This tutorial walks through a complete, reproducible setup that runs a continuous, scoped AWS Security Agent penetration test as a **post-deployment gate** in GitLab CI/CD. You deploy an intentionally vulnerable sample application to a pre-production environment, then let AWS Security Agent test the deployed change and block promotion to production when it finds a vulnerability.

Note

Hosting the sample application might result in charges to your AWS account.

For the concepts behind the integration — how the test is scoped to the deployed change, how OIDC authentication works, and how gating decisions are made — see the section called “Run

penetration tests from your CI/CD pipeline". This tutorial applies those concepts to a concrete GitLab project.

> **Intentional vulnerability**
>
> The sample application in this tutorial contains a deliberate SQL injection vulnerability so that the gate has a real finding to block on. Deploy it only to an isolated, non-production environment that you control, and remove it when you finish. Do not expose it to the public internet or connect it to production data.

Prerequisites

Before you start, make sure you have the following:

- An AWS account, and permission in it to create an IAM OIDC identity provider, an IAM role, and AWS Security Agent resources.
- A GitLab project that you own, on `gitlab.com`, in which you can edit `.gitlab-ci.yml` and set CI/CD variables. This tutorial uses the placeholder project `my-group/my-app`; substitute your own group and project throughout.
- A GitLab access token (a Project or Group Access Token, or a personal access token) with the `read_api` and `write_repository` scopes and at least the Developer role. The integration uses it to maintain the diff baseline tag in your project.
- A pre-production environment that the sample application can deploy to, reachable from the network where your penetration test runs (the public internet, or a VPC in your AWS account for a private target). See the section called "Connect agent to private VPC resources" for private targets.
- The AWS Security Agent one-time setup described in the section called "Run penetration tests from your CI/CD pipeline" under **Before you start**: a verified target domain, a CI/CD-enabled penetration test, your GitLab project connected as a source control integration, and the Agent Space ID (`as-…`) and penetration test ID (`pt-…`) that identify them.
- A project **maximum job timeout** higher than the gate job's own timeout. The gate needs a long-running job, and GitLab's runner enforces whichever limit is lower. This tutorial's gate job runs for up to 55 minutes, so raise the project maximum above that under **Settings**, **CI/CD**, **General pipelines** before your first run.

A pipeline variable can override the gate's settings

On GitLab a pipeline variable takes precedence over the values a job sets itself, and the component's settings are passed that way. If you can run a pipeline, you can therefore supply a variable that lowers the severity threshold, turns on dry-run mode, or relaxes error handling, and the gate will honor it over your `.gitlab-ci.yml`.
Before you rely on this gate to hold a release, restrict who may override variables: under **Settings**, **CI/CD**, **Variables**, set the minimum role required to override variables on a pipeline. See the section called "On GitLab, the committed settings are not the gate's contract".

Throughout this tutorial, replace the following placeholders with your own values:

Placeholder	Replace with
`111122223333`	Your 12-digit AWS account ID.
`AGENT_SPACE_ID (as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx )`	Your Agent Space ID.
`PENTEST_ID (pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx )`	Your CI/CD-enabled penetration test ID.
`my-group/my-app`	Your GitLab group and project path.
`us-west-2`	The AWS Region where your Agent Space and penetration test are located.

Step 1: Create the vulnerable sample application

Create a small Flask application with one endpoint that builds a SQL query by string concatenation — a classic SQL injection. In your project, add `app.py`:

```
import sqlite3
from flask import Flask, request
```

```
app = Flask(__name__)


@app.route("/products")
def products():
    # Intentionally vulnerable: user input is concatenated directly into SQL.
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = '" + category + "'"
    rows = cursor.execute(query).fetchall()
    conn.close()
    return {"products": rows}


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Add a `deploy.sh` script that deploys this application to your pre-production environment. The details depend on your infrastructure; the gate does not require any particular deployment mechanism, only that the application is running and reachable when the test runs.

Step 2: Set up GitLab OIDC and an IAM role

The component authenticates to AWS with GitLab OIDC ID tokens — no long-lived access keys. Do this once in your AWS account.

1. If you do not already have one, create the GitLab OIDC identity provider. The client ID is the audience (aud) that the component sets on the ID token, which defaults to `https://gitlab.com`:

   ```
   aws iam create-open-id-connect-provider \
     --url https://gitlab.com \
     --client-id-list https://gitlab.com
   ```

2. Create the IAM role with a trust policy scoped to your project. Save the following as `trust-policy.json`, replacing the account ID and project path:

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
   ```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/gitlab.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "gitlab.com:aud": "https://gitlab.com"
        },
        "StringLike": {
          "gitlab.com:sub": "project_path:my-group/my-app:ref_type:branch:ref:main"
        }
      }
    }
  ]
}
```

The GitLab OIDC `sub` claim has the form `project_path:GROUP/PROJECT:ref_type:branch:ref:BRANCH`. Keep it as specific as your workflow allows. For the reasoning behind scoping the `sub` claim, see the section called "Scope the trust policy to your pipeline".

3. Save the least-privilege permissions policy as `permissions.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "securityagent:StartPentestJob",
        "securityagent:BatchGetPentestJobs",
        "securityagent:StopPentestJob",
        "securityagent:ListFindings",
        "securityagent:BatchGetFindings",
        "securityagent:ListIntegratedResources"
      ],
      "Resource": "*"
    }
  ]
}
```

For what each action does, see the section called “Grant least-privilege permissions”.

4. Create the role and attach the policy. The component asks AWS STS for a credential lifetime derived from the gate timeout, so the role’s maximum session duration has to cover it. Size it as `(timeout-minutes + 10) * 60` seconds — the extra ten minutes cover findings collection and cleanup. IAM’s own minimum for a role’s maximum session duration is 3,600 seconds, so that formula asks for more than IAM’s floor only once your timeout passes 50 minutes. This tutorial’s `timeout-minutes: "45"` is already covered by 3,600; the example sets 4,200 so that raising the timeout to the 60-minute default needs no change to the role:

```
aws iam create-role --role-name my-pentest-gate-role \
  --assume-role-policy-document file://trust-policy.json \
  --max-session-duration 4200

aws iam put-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest \
  --policy-document file://permissions.json
```

> **Important**
>
> AWS STS **rejects** a request for a longer session than the role allows rather than shortening it, so a role whose maximum is too low fails the credential exchange outright — before any penetration test starts. Recompute the maximum whenever you change `timeout-minutes`.
>
> This is also why you cannot reuse a role built for GitHub Actions here without checking it. A role left at IAM’s 3,600-second default works on GitHub, because `configure-aws-credentials` asks for less, but it fails on GitLab as soon as the timeout needs more.

The role ARN for the component is `arn:aws:iam::111122223333:role/my-pentest-gate-role`.

Step 3: Store the GitLab access token

The component keeps the baseline — the record of the commit it last cleared — as a tag in your project, and it needs a token to read and write one. `CI_JOB_TOKEN` cannot create tags or read a merge base, so it is not sufficient.

Create a Project or Group Access Token (or a personal access token) with the `read_api` and `write_repository` scopes and the **Developer** role. In your project, go to **Settings**, **CI/CD**, **Variables** and add it as a masked variable named `SA_GITLAB_TOKEN`.

Keep the `continuum/pentest/cicd/baseline/*` tag namespace unprotected, which is the default, so that the Developer-role token can create the baseline tags. If you protect that namespace, add the token's identity to its *Allowed to create* list instead.

> **Important**
>
> Without the token the gate still runs, but it has no persistent baseline: each run falls back to the parent of the commit being gated, so it tests one commit rather than the range you deployed. That is a narrower test than you asked for, not a wider one.

Step 4: Add the component to your pipeline

Edit `.gitlab-ci.yml`. Place the gate in a stage **after** your deploy stage, and gate the production job on it with `needs`. Because a failed job fails the pipeline, the production job does not run when the gate fails.

```
stages: [build, deploy-preprod, security-gate, deploy-prod]

# The baseline is a tag in your own project, and creating a tag starts a
# pipeline, so without this rule the gate re-triggers itself on every pass —
# and each passing run writes another tag. Exclude the baseline namespace.
workflow:
  rules:
    - if: '$CI_COMMIT_TAG =~ /^continuum\/pentest\/cicd\/baseline\//'
      when: never
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
      when: never
    - if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'

include:
  - component: $CI_SERVER_FQDN/aws/continuum/pentest/run-pentest@1
    inputs:
      agent-space-id: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
      pentest-id: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
      stage: security-gate
      aws-role-arn: "arn:aws:iam::111122223333:role/my-pentest-gate-role"
```

```
      aws-region: "us-west-2"
      gitlab-token: "$SA_GITLAB_TOKEN"
      severity-threshold: "HIGH"
      on-scope-conflict: "block"
      timeout-minutes: "45"
      job-timeout: "55 minutes"

deploy-preprod:
  stage: deploy-preprod
  environment: preprod
  script: ./deploy.sh preprod

deploy-prod:
  stage: deploy-prod
  needs: ["aws-continuum-pentest"]
  when: manual
  environment: production
  script: ./deploy.sh prod
```

The component defines a job named `aws-continuum-pentest`. The `stage` input must name a stage that runs after your deploy stage, and the `deploy-prod` job's `needs` must reference that job name so that promotion is gated on the result.

Exclude the baseline tag namespace, or the gate re-triggers itself

Each time the gate passes it creates a tag under `continuum/pentest/cicd/baseline/`, and on GitLab creating a tag is a pipeline trigger. If any job in your pipeline is tag-driven — the common `rules: if: $CI_COMMIT_TAG` idiom, `only: tags`, or release automation — the gate fires again on its own tag, passes, writes another tag, and repeats. Each of those runs starts another penetration test against your target, and each one holds one of your account's concurrent pentest runs.

The `workflow: rules` block above prevents that for the whole pipeline. If you cannot use a `workflow` rule, exclude the namespace on each tag-driven job instead, with `rules: if: '$CI_COMMIT_TAG !~ /^continuum\/pentest\/cicd\/baseline\//'`.

Note

Pin the component to a released version rather than a moving reference, so that a change to the upstream component cannot alter what runs in your pipeline without your review.

`@1` tracks the latest 1.x release; pin the exact version, such as `@v1.0.4`, to hold it still. Note that GitLab looks up an exact version as a literal ref, so the `v` prefix is required for one — `@1.0.4` resolves to nothing, while `@v1.0.4`, `@1`, and `@1.0` all resolve. See the section called "How the integration differs by provider".

Step 5: Validate your setup with a dry run

Before you rely on the gate, run it once in dry-run mode. Add `dry-run: "true"` to the component inputs and run the pipeline:

```
- component: $CI_SERVER_FQDN/aws/continuum/pentest/run-pentest@1
  inputs:
    agent-space-id: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    pentest-id: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    stage: security-gate
    aws-role-arn: "arn:aws:iam::111122223333:role/my-pentest-gate-role"
    aws-region: "us-west-2"
    dry-run: "true"
```

The dry run checks AWS authentication, the Region, and that your project resolves to its integration — without starting a penetration test job. It surfaces the common setup mistakes (a bad OIDC trust policy, a Region mismatch, or a project that is not connected) in seconds. When it passes, remove the `dry-run` input and continue. For more information, see the section called "Validate your setup before you trust the gate".

Step 6: Trigger the gate and read the result

Push a commit to your deploy branch that changes the vulnerable endpoint. This triggers the deploy pipeline: the `deploy-preprod` job deploys the change, then the `aws-continuum-pentest` job starts a penetration test scoped to that change.

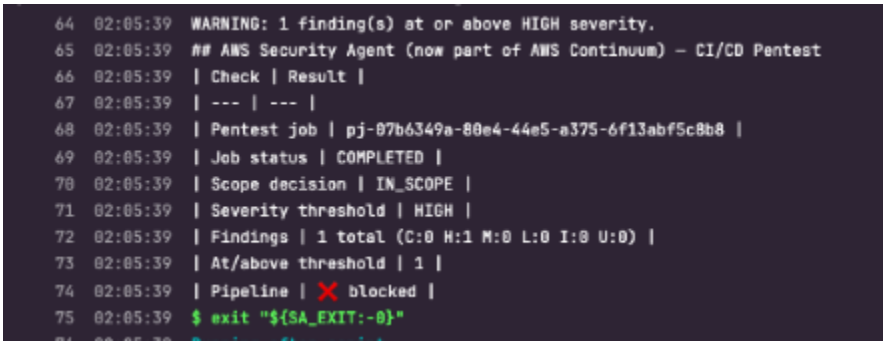
Open the pipeline and watch the `aws-continuum-pentest` job:

1. In `before_script`, GitLab mints an ID token and the job exchanges it for AWS credentials with `AssumeRoleWithWebIdentity`. No stored AWS keys are involved.
2. In `script`, the component resolves your project, starts a scoped job, polls it to completion, evaluates findings against your `severity-threshold`, and exits non-zero to fail the pipeline if the gate blocks.

A scoped penetration test typically runs for tens of minutes, so the job runs for a while. That is expected, not a hang.

Because the deployed change introduced a SQL injection vulnerability, AWS Security Agent returns an **in scope** decision with a `HIGH` (or higher) finding. The `aws-continuum-pentest` job fails, and because `deploy-prod` needs it, promotion to production is blocked.

The job log records the verdict: the scope decision, the severity threshold the run was evaluated against, the finding counts, and whether the gate blocked.



The job writes its outputs as a dotenv report artifact that downstream jobs can read as variables, including `SCOPE_DECISION` (`IN_SCOPE`, `SCOPED_OUT`, or `SCOPE_CONFLICT`), `FINDINGS_COUNT`, and `STATUS`. If you set `sast-report: "true"`, the job also emits `gl-sast-report.json` so the findings appear in the merge request security widget. For the full gating behavior, see the section called “Choose how the pipeline responds”.

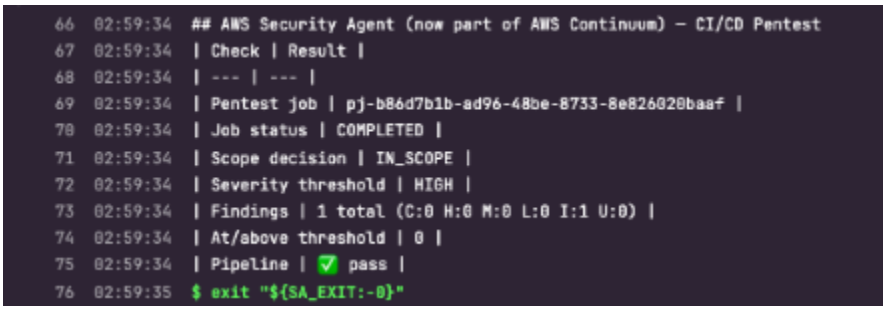
Step 7: Fix the finding and confirm the gate passes

Fix the vulnerability by using a parameterized query instead of string concatenation. In `app.py`, replace the query construction:

```
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = ?"
    rows = cursor.execute(query, (category,)).fetchall()
    conn.close()
    return {"products": rows}
```

Push the fix to your deploy branch. The pipeline redeploys the change and runs the gate again against the new commit. With the injection removed, AWS Security Agent finds no findings at or above `HIGH`, the gate job passes, and the manual `deploy-prod` job becomes available to promote.

The job log shows the same checks for the new commit, with no findings at or above the threshold and a passing verdict.



Clean up

To avoid ongoing cost and to remove the intentionally vulnerable application:

1. Tear down the sample application from your pre-production environment.
2. Remove the component include from `.gitlab-ci.yml`, or disable the pipeline, if you do not intend to keep the gate.
3. Delete the `SA_GITLAB_TOKEN` CI/CD variable if it was created only for this tutorial.
4. If you created the IAM role and OIDC provider only for this tutorial, delete them:

```
aws iam delete-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest
aws iam delete-role --role-name my-pentest-gate-role
aws iam delete-open-id-connect-provider \
  --open-id-connect-provider-arn arn:aws:iam::111122223333:oidc-provider/gitlab.com
```

Related topics

- the section called “Run penetration tests from your CI/CD pipeline” – concepts and reference for the CI/CD pipeline integration.
- the section called “Tutorial: Gate a GitHub Actions deployment with a scoped penetration test”, the section called “Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test”, the section called “Tutorial: Gate an Azure DevOps deployment with a scoped penetration test” – the same tutorial for other CI/CD platforms.
- the section called “Security best practices” – general best practices, including testing against non-production environments.

Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test

Public preview

CI/CD pipeline integration is in preview and is subject to change.
While it is in preview, AWS doesn't charge for continuous pentest runs, however you start them. You still pay for the other AWS resources you use, such as hosting the application you test. Other penetration tests are metered as usual. For more information, see the section called "Pricing and billing".

This tutorial walks through a complete, reproducible setup that runs a continuous, scoped AWS Security Agent penetration test as a **post-deployment gate** in Bitbucket Pipelines. You deploy an intentionally vulnerable sample application to a staging environment, then let AWS Security Agent test the deployed change and block promotion to production when it finds a vulnerability.

Note

Hosting the sample application might result in charges to your AWS account.

For the concepts behind the integration — how the test is scoped to the deployed change, how OIDC authentication works, and how gating decisions are made — see the section called "Run penetration tests from your CI/CD pipeline". This tutorial applies those concepts to a concrete Bitbucket repository.

Intentional vulnerability

The sample application in this tutorial contains a deliberate SQL injection vulnerability so that the gate has a real finding to block on. Deploy it only to an isolated, non-production environment that you control, and remove it when you finish. Do not expose it to the public internet or connect it to production data.

Prerequisites

Before you start, make sure you have the following:

- An AWS account, and permission in it to create an IAM OIDC identity provider, an IAM role, and AWS Security Agent resources.
- A Bitbucket Cloud repository that you own, in which you can edit `bitbucket-pipelines.yml` and repository settings. This tutorial uses the placeholder repository `my-workspace/my-app`; substitute your own workspace and repository throughout.
- A staging environment that the sample application can deploy to, reachable from the network where your penetration test runs (the public internet, or a VPC in your AWS account for a private target). See the section called “Connect agent to private VPC resources” for private targets.
- The AWS Security Agent one-time setup described in the section called “Run penetration tests from your CI/CD pipeline” under **Before you start**: a verified target domain, a CI/CD-enabled penetration test, your Bitbucket repository connected as a source control integration (see the section called “Connect AWS Security Agent to Bitbucket repositories”), and the Agent Space ID (`as-…`) and penetration test ID (`pt-…`) that identify them.
- A Bitbucket access token with the **Pipelines variable** scope, for the baseline. On Bitbucket the gate cannot keep its baseline in git, because the REST API exposes only branches and tags, so it stores the commit it last cleared in a repository **pipelines variable** named `SECURITY_AGENT_BASELINE_` followed by a 40-character identifier derived from your Agent Space and penetration test IDs. Writing that variable needs a token.

  Create a repository, project, or workspace access token with the Pipelines variable scope, then add it under **Repository settings**, **Repository variables** as a secured variable named `SA_BITBUCKET_TOKEN`, and pass it to the Pipe as `BITBUCKET_ACCESS_TOKEN`.

  > **Important**
  >
  > Without that token the gate still runs and still passes or fails correctly, but it keeps no baseline: every run falls back to the parent of the commit being gated, so it tests a single commit instead of the range you deployed. Nothing fails, which is what makes the omission easy to miss — you get a narrower test than you asked for.

Throughout this tutorial, replace the following placeholders with your own values:

Placeholder	Replace with
`111122223333`	Your 12-digit AWS account ID.
`AGENT_SPACE_ID (as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx )`	Your Agent Space ID.
`PENTEST_ID (pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx )`	Your CI/CD-enabled penetration test ID.
`my-workspace/my-app`	Your Bitbucket workspace and repository.
`us-west-2`	The AWS Region where your Agent Space and penetration test are located.

Step 1: Create the vulnerable sample application

Create a small Flask application with one endpoint that builds a SQL query by string concatenation — a classic SQL injection. In your repository, add `app.py`:

```
import sqlite3
from flask import Flask, request

app = Flask(__name__)


@app.route("/products")
def products():
    # Intentionally vulnerable: user input is concatenated directly into SQL.
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = '" + category + "'"
    rows = cursor.execute(query).fetchall()
    conn.close()
    return {"products": rows}
```

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Add a `deploy.sh` script that deploys this application to your staging environment. The details depend on your infrastructure; the gate does not require any particular deployment mechanism, only that the application is running and reachable when the test runs.

Step 2: Set up Bitbucket OIDC and an IAM role

Bitbucket Pipelines authenticates to AWS with OIDC — no long-lived access keys. Bitbucket acts as a web identity provider: a step with `oidc: true` receives a signed OIDC token in the `$BITBUCKET_STEP_OIDC_TOKEN` variable, which you exchange for temporary AWS credentials.

1. In your repository, go to **Repository settings**, **OpenID Connect** (under **Pipelines**). Copy the **Identity provider URL** and the **Audience**. The provider URL has the form `https://api.bitbucket.org/2.0/workspaces/my-workspace/pipelines-config/identity/oidc`, and the audience is specific to your workspace.
2. Create the IAM OIDC identity provider in your AWS account using the identity provider URL and audience you copied:

   ```
   aws iam create-open-id-connect-provider \
     --url https://api.bitbucket.org/2.0/workspaces/my-workspace/pipelines-config/
   identity/oidc \
     --client-id-list ari:cloud:bitbucket::workspace/YOUR_WORKSPACE_UUID
   ```

   Use the exact audience value from the **OpenID Connect** settings page as the `--client-id-list` value.
3. Create the IAM role with a trust policy scoped to your repository. Bitbucket includes the repository UUID in the token's `sub` claim, so scope the role to that UUID rather than to a wildcard. Save the following as `trust-policy.json`, replacing the account ID, the provider host portion, the audience, and the repository UUID:

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
   ```

```
        "Federated": "arn:aws:iam::111122223333:oidc-provider/api.bitbucket.org/2.0/
workspaces/my-workspace/pipelines-config/identity/oidc"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "api.bitbucket.org/2.0/workspaces/my-workspace/pipelines-config/identity/
oidc:aud": "ari:cloud:bitbucket::workspace/YOUR_WORKSPACE_UUID"
        },
        "StringLike": {
          "api.bitbucket.org/2.0/workspaces/my-workspace/pipelines-config/identity/
oidc:sub": "{YOUR_REPOSITORY_UUID}:*"
        }
      }
    }
  ]
}
```

For the reasoning behind scoping the `sub` claim to the repository (and not using a wildcard), see the section called “Scope the trust policy to your pipeline”. Confirm the exact `sub` format your workspace emits from the **OpenID Connect** settings page before you write the trust policy.

4. Save the least-privilege permissions policy as `permissions.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "securityagent:StartPentestJob",
        "securityagent:BatchGetPentestJobs",
        "securityagent:StopPentestJob",
        "securityagent:ListFindings",
        "securityagent:BatchGetFindings",
        "securityagent:ListIntegratedResources"
      ],
      "Resource": "*"
    }
  ]
}
```

For what each action does, see the section called “Grant least-privilege permissions”.

5. Create the role and attach the policy. A penetration test job can run for tens of minutes, so create the role with a maximum session duration long enough to outlast the step:

```
aws iam create-role --role-name my-pentest-gate-role \
  --assume-role-policy-document file://trust-policy.json \
  --max-session-duration 7200

aws iam put-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest \
  --policy-document file://permissions.json
```

   The role ARN for the pipeline is `arn:aws:iam::111122223333:role/my-pentest-gate-role`.

Step 3: Add the pipeline

Edit `bitbucket-pipelines.yml`. Run the deploy step, then the gate step. Use a **Pipe** to run the integration, and enable OIDC on the gate step. Store the role ARN as a repository variable named `AWS_PENTEST_ROLE_ARN`, and the baseline token as `SA_BITBUCKET_TOKEN`.

```
image: atlassian/default-image:4

pipelines:
  branches:
    main:
      - step:
          name: Deploy to staging
          deployment: staging
          script:
            - ./deploy.sh staging
      - step:
          name: Security Agent pentest gate
          oidc: true
          script:
            - pipe: docker://public.ecr.aws/aws-continuum/run-pentest:1-bitbucket
              variables:
                AGENT_SPACE_ID: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
                PENTEST_ID: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
                AWS_REGION: "us-west-2"
                AWS_ROLE_ARN: "$AWS_PENTEST_ROLE_ARN"
                BITBUCKET_ACCESS_TOKEN: "$SA_BITBUCKET_TOKEN"
```

```
            SEVERITY_THRESHOLD: "HIGH"
            ON_SCOPE_CONFLICT: "block"
            TIMEOUT_MINUTES: "45"
    - step:
        name: Deploy to production
        deployment: production
        script:
          - ./deploy.sh production
```

The `oidc: true` flag is required: it is what makes Bitbucket issue the OIDC token, and without it authentication to AWS cannot succeed.

Pass **every** value the Pipe reads through `variables:`, including `AWS_ROLE_ARN`. A Pipe runs in its own container, and a variable exported in the step's `script` does not cross into it, so a role ARN exported in the shell is not reachable by the Pipe at all. The OIDC token is the one exception: Bitbucket places `$BITBUCKET_STEP_OIDC_TOKEN` inside the Pipe's container directly, so the Pipe reads it there and exchanges it for temporary credentials. You do not need to write the token to a file.

Pinning the commit range

The Pipe reads the commit range through `variables:` like everything else. To test a specific range instead of the one the gate resolves, add `BASE_COMMIT_SHA`, `HEAD_COMMIT_SHA`, or both:

```
            - pipe: docker://public.ecr.aws/aws-continuum/run-pentest:1-
bitbucket
              variables:
                AGENT_SPACE_ID: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
                PENTEST_ID: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
                AWS_REGION: "us-west-2"
                AWS_ROLE_ARN: "$AWS_PENTEST_ROLE_ARN"
                BASE_COMMIT_SHA: "<full 40-character commit hash>"
```

Each takes the full 40-character hash, and the commit needs to be in the clone. Raise `clone: depth` if it's older than the default depth. Either variable works on its own, but pinning only `HEAD_COMMIT_SHA` is the riskier choice. When the Pipe has no recorded baseline, it takes the parent of the commit you pinned, and a pinned commit missing from the clone has no parent to take. The Pipe then starts no penetration test at all, and under the default `FAIL_ON_ERROR` of `false` the step passes without testing anything. Set both

variables when you need an exact range. For more information, see the section called “Pin the commit range explicitly”.

Because Bitbucket Pipelines steps run in sequence and a failed step fails the pipeline, the gate step blocks the production step when it fails.

A moving tag can change what runs in your pipeline

Pin the Pipe to an immutable image digest (for example, `docker://public.ecr.aws/aws-continuum/run-pentest@sha256:…`) rather than a tag, so that a change to the upstream image cannot alter what runs in your pipeline without your review. The `1-bitbucket` tag above tracks the latest 1.x release, so it moves as patches are published. See the section called “How the integration differs by provider”.

Step 4: Validate your setup with a dry run

Before you rely on the gate, run it once in dry-run mode. Add the dry-run variable to the Pipe and run the pipeline:

```
- pipe: docker://public.ecr.aws/aws-continuum/run-pentest:1-bitbucket
  variables:
    AGENT_SPACE_ID: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    PENTEST_ID: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    AWS_REGION: "us-west-2"
    AWS_ROLE_ARN: "$AWS_PENTEST_ROLE_ARN"
    DRY_RUN: "true"
```

The dry run checks AWS authentication, the Region, and that your repository resolves to its integration — without starting a penetration test job. It surfaces the common setup mistakes (a bad OIDC trust policy, a Region mismatch, or a repository that is not connected) in seconds. When it passes, remove the dry-run variable and continue. For more information, see the section called “Validate your setup before you trust the gate”.

Step 5: Trigger the gate and read the result

Push a commit to `main` that changes the vulnerable endpoint. This triggers the pipeline: the deploy step deploys the change to staging, then the gate step starts a penetration test scoped to that change.

Open the pipeline and watch the **Security Agent pentest gate** step:

1. Bitbucket issues the OIDC token, and the Pipe exchanges it for AWS credentials with `AssumeRoleWithWebIdentity`. No stored AWS keys are involved.
2. The Pipe resolves your repository, starts a scoped job, polls it to completion, and evaluates the findings against your severity threshold.

A scoped penetration test typically runs for tens of minutes, so the step runs for a while. That is expected, not a hang.

Because the deployed change introduced a SQL injection vulnerability, AWS Security Agent returns an **in scope** decision with a `HIGH` (or higher) finding. The gate step fails, and the pipeline stops before any later promotion step runs.

The step log records the verdict: the scope decision, the severity threshold the run was evaluated against, the finding counts, and whether the gate blocked.

```
113  — Collect and evaluate findings —
114  Findings: 2 total; 1 at/above HIGH.
115  ## AWS Security Agent (now part of AWS Continuum) — CI/CD Pentest
116
117  | Check | Result |
118  | --- | --- |
119  | Pentest job | pj-b41e6251-e262-44d8-91f8-d3c499b2336c |
120  | Job status | COMPLETED |
121  | Report | [View in AWS Security Agent](https://app-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.securityagent.global.a
122  | Scope decision | IN_SCOPE |
123  | Severity threshold | HIGH |
124  | Findings | 2 total (C:0 H:1 M:0 L:0 I:0 U:1) |
125  | At/above threshold | 1 |
126  | Pipeline | ❌ blocked |
127
128  ❌ 1 finding(s) at or above HIGH severity.
```

For the full gating behavior — including how authentication failures always block and how you choose fail-closed or fail-open for infrastructure errors — see the section called “Choose how the pipeline responds”.

Step 6: Fix the finding and confirm the gate passes

Fix the vulnerability by using a parameterized query instead of string concatenation. In `app.py`, replace the query construction:

```
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = ?"
    rows = cursor.execute(query, (category,)).fetchall()
    conn.close()
    return {"products": rows}
```

Push the fix to `main`. The pipeline redeploys the change and runs the gate again against the new commit. With the injection removed, AWS Security Agent finds no findings at or above `HIGH`, the gate step passes, and the pipeline continues.

The step log shows the same checks for the new commit, with no findings at or above the threshold and a passing verdict.



Clean up

To avoid ongoing cost and to remove the intentionally vulnerable application:

> **⚠ Stopping the pipeline does not stop the penetration test**
>
> If you stop a running pipeline from the Bitbucket UI or API, the penetration test job keeps running server-side and keeps acting against your target. A Pipe has no post-step hook on

Bitbucket, so the gate cannot reliably request a stop once the pipeline is being torn down — unlike its own timeout, which it handles itself.
If you stop a pipeline while the gate is waiting, open the penetration test in the AWS Security Agent console and stop the job there. Check for a running job before you delete the IAM role, because the role is what the cleanup would have used.

1. Tear down the sample application from your staging environment.
2. Remove the gate step from `bitbucket-pipelines.yml`, or disable the pipeline, if you do not intend to keep the gate.
3. Delete the baseline repository variable, which is named `SECURITY_AGENT_BASELINE_` followed by a 40-character identifier, under **Repository settings**, **Repository variables**. Also delete the `SA_BITBUCKET_TOKEN` variable and revoke the token if you created it only for this tutorial.
4. If you created the IAM role and OIDC provider only for this tutorial, delete them:

```
aws iam delete-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest
aws iam delete-role --role-name my-pentest-gate-role
aws iam delete-open-id-connect-provider \
  --open-id-connect-provider-arn arn:aws:iam::111122223333:oidc-provider/
api.bitbucket.org/2.0/workspaces/my-workspace/pipelines-config/identity/oidc
```

Related topics

- the section called “Run penetration tests from your CI/CD pipeline” – concepts and reference for the CI/CD pipeline integration.
- the section called “Tutorial: Gate a GitHub Actions deployment with a scoped penetration test”, the section called “Tutorial: Gate a GitLab CI/CD deployment with a scoped penetration test”, the section called “Tutorial: Gate an Azure DevOps deployment with a scoped penetration test” – the same tutorial for other CI/CD platforms.
- the section called “Security best practices” – general best practices, including testing against non-production environments.

Tutorial: Gate an Azure DevOps deployment with a scoped penetration test

Public preview

CI/CD pipeline integration is in preview and is subject to change.
While it is in preview, AWS doesn't charge for continuous pentest runs, however you start them. You still pay for the other AWS resources you use, such as hosting the application you test. Other penetration tests are metered as usual. For more information, see the section called "Pricing and billing".

This tutorial walks through a complete, reproducible setup that runs a continuous, scoped AWS Security Agent penetration test as a **post-deployment gate** in Azure DevOps Pipelines. You deploy an intentionally vulnerable sample application to a staging environment, then let AWS Security Agent test the deployed change and block promotion to production when it finds a vulnerability.

Note

Hosting the sample application might result in charges to your AWS account.

For the concepts behind the integration — how the test is scoped to the deployed change, how OIDC authentication works, and how gating decisions are made — see the section called "Run penetration tests from your CI/CD pipeline". This tutorial applies those concepts to a concrete Azure DevOps project.

Intentional vulnerability

The sample application in this tutorial contains a deliberate SQL injection vulnerability so that the gate has a real finding to block on. Deploy it only to an isolated, non-production environment that you control, and remove it when you finish. Do not expose it to the public internet or connect it to production data.

Prerequisites

Before you start, make sure you have the following:

- An AWS account, and permission in it to create an IAM OIDC identity provider, an IAM role, and AWS Security Agent resources.
- An Azure DevOps organization and project that you own, in which you can create service connections, install extensions from the Visual Studio Marketplace, and edit pipelines.
- The **AWS Toolkit for Azure DevOps** extension installed in your organization. This tutorial uses an AWS service connection from that extension for authentication.
- The `RunPentest` task available in your organization. The task is not yet carried by a published release of the AWS Toolkit for Azure DevOps, so installing the extension alone does not provide it. Confirm the task appears in your pipeline task list before you start; if it does not, contact AWS Support for the current way to obtain it.
- The build service identity granted permission to record the baseline. The gate stores the commit it last cleared using the pipeline's own `System.AccessToken`, and the identity that token represents needs **Contribute** on the repository when your source is Azure Repos, or **Tag builds** when your source is GitHub, GitLab, or Bitbucket. Grant it under **Project settings**, **Repositories** or **Pipelines**, **Settings**.

  > **Important**
  >
  > Without that permission the gate still runs and still blocks correctly, but the baseline never advances: every run resolves its base from a fallback rung instead, testing a narrower range than you deployed, and nothing in the pipeline fails to tell you.
  > You do not need to map the token into the task with `env: SYSTEM_ACCESSTOKEN: $(System.AccessToken)`. The task reads `System.AccessToken` directly, and the agent provides it without a mapping. That mapping is needed only for script steps, so adding it does no harm but fixes nothing — if the baseline is not advancing, the permission is the cause.

- A staging environment that the sample application can deploy to, reachable from the network where your penetration test runs (the public internet, or a VPC in your AWS account for a private target). See the section called "Connect agent to private VPC resources" for private targets.
- An agent pool whose job timeout exceeds the gate's own timeout. A free Microsoft-hosted agent is killed at 60 minutes, which is also the task's default `timeoutMinutes`, so on that

tier the agent is terminated before the gate can report a timeout of its own. This tutorial sets `timeoutMinutes: "45"`, which fits. If you raise it, use a paid or self-hosted pool.

- The AWS Security Agent one-time setup described in the section called "Run penetration tests from your CI/CD pipeline" under **Before you start**: a verified target domain, a CI/CD-enabled penetration test, your repository connected as a source control integration, and the Agent Space ID (`as-...`) and penetration test ID (`pt-...`) that identify them.

Throughout this tutorial, replace the following placeholders with your own values:

Placeholder	Replace with
`111122223333`	Your 12-digit AWS account ID.
`AGENT_SPACE_ID (as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)`	Your Agent Space ID.
`PENTEST_ID (pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)`	Your CI/CD-enabled penetration test ID.
`us-west-2`	The AWS Region where your Agent Space and penetration test are located.

Step 1: Create the vulnerable sample application

Create a small Flask application with one endpoint that builds a SQL query by string concatenation — a classic SQL injection. In your repository, add `app.py`:

```
import sqlite3
from flask import Flask, request

app = Flask(__name__)


@app.route("/products")
def products():
    # Intentionally vulnerable: user input is concatenated directly into SQL.
```

```
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = '" + category + "'"
    rows = cursor.execute(query).fetchall()
    conn.close()
    return {"products": rows}


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Add a `deploy.sh` script that deploys this application to your staging environment. The details depend on your infrastructure; the gate does not require any particular deployment mechanism, only that the application is running and reachable when the test runs.

Step 2: Set up the IAM role for Azure DevOps OIDC

Azure DevOps authenticates to AWS through an AWS service connection that uses **workload identity federation** — no long-lived access keys. Azure DevOps issues a signed OIDC token for the pipeline, and AWS STS exchanges it for temporary credentials through an IAM role you create.

1. Create the IAM OIDC identity provider for Azure DevOps in your AWS account. The provider URL is your organization's Azure DevOps issuer, and the audience is `api://AzureADTokenExchange`:

   ```
   aws iam create-open-id-connect-provider \
     --url https://vstoken.dev.azure.com/YOUR_ORGANIZATION_ID \
     --client-id-list api://AzureADTokenExchange
   ```

   Confirm your organization's exact issuer URL and audience from the service connection setup dialog in Azure DevOps (you obtain the **Issuer** and **Subject identifier** there, and the service connection also gives you an **Issuer** and **Subject** to paste into the trust policy).
2. Create the IAM role with a trust policy scoped to your Azure DevOps organization, project, and pipeline through the token's subject. Save the following as `trust-policy.json`, replacing the account ID, issuer host, and subject:

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
   ```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/vstoken.dev.azure.com/
YOUR_ORGANIZATION_ID"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "vstoken.dev.azure.com/YOUR_ORGANIZATION_ID:aud": "api://
AzureADTokenExchange",
          "vstoken.dev.azure.com/YOUR_ORGANIZATION_ID:sub": "sc://YOUR_ORGANIZATION/
YOUR_PROJECT/YOUR_SERVICE_CONNECTION_NAME"
        }
      }
    }
  ]
}
```

The subject identifies the specific service connection, which scopes the role to your organization, project, and pipeline. Use the exact **Subject identifier** that the service connection dialog shows. For the reasoning behind scoping the subject rather than using a wildcard, see the section called "Scope the trust policy to your pipeline".

3. Save the least-privilege permissions policy as `permissions.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "securityagent:StartPentestJob",
        "securityagent:BatchGetPentestJobs",
        "securityagent:StopPentestJob",
        "securityagent:ListFindings",
        "securityagent:BatchGetFindings",
        "securityagent:ListIntegratedResources"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

For what each action does, see the section called “Grant least-privilege permissions”.

4. Create the role and attach the policy. A penetration test job can run for tens of minutes, so create the role with a maximum session duration long enough to outlast the task:

```
aws iam create-role --role-name my-pentest-gate-role \
  --assume-role-policy-document file://trust-policy.json \
  --max-session-duration 7200

aws iam put-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest \
  --policy-document file://permissions.json
```

The role ARN for the pipeline is `arn:aws:iam::111122223333:role/my-pentest-gate-role`.

Step 3: Create the AWS service connection

In Azure DevOps, go to **Project settings**, **Service connections**, and create a new **AWS** service connection provided by the AWS Toolkit. Choose **workload identity federation** (OIDC) rather than access keys, and set:

- **Role to assume** – `arn:aws:iam::111122223333:role/my-pentest-gate-role`.
- **AWS Region** – `us-west-2`.

Name the connection (for example, `aws-pentest-gate`). Azure DevOps generates the issuer and subject values that you used in the trust policy in Step 2; if you created the role before the connection, update the trust policy’s subject to match the value the connection shows.

Step 4: Add the pipeline task

Edit your pipeline YAML. Run the deploy stage, then a gate stage that depends on it. In the gate stage, run the AWS Toolkit task for the integration, referencing your service connection.

```
trigger:
  branches:
    include: [main]
```

```
stages:
  - stage: DeployStaging
    jobs:
      - job: deploy
        pool:
          vmImage: ubuntu-latest
        steps:
          - checkout: self
            persistCredentials: false
          - script: ./deploy.sh staging
            displayName: Deploy to staging

  - stage: SecurityGate
    dependsOn: DeployStaging
    jobs:
      - job: pentest
        pool:
          vmImage: ubuntu-latest
        steps:
          - task: RunPentest@1
            inputs:
              awsCredentials: aws-pentest-gate
              regionName: us-west-2
              agentSpaceId: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
              pentestId: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
              severityThreshold: HIGH
              onScopeConflict: block
              timeoutMinutes: "45"

  - stage: DeployProd
    dependsOn: SecurityGate
    jobs:
      - job: deploy_prod
        pool:
          vmImage: ubuntu-latest
        steps:
          - checkout: self
            persistCredentials: false
          - script: ./deploy.sh production
            displayName: Deploy to production
```

Because `DeployProd` declares `dependsOn: SecurityGate` and a failed task fails its stage, the production stage does not run when the gate fails. The `awsCredentials` input references the service connection you created in Step 3, which supplies temporary credentials through workload identity federation.

Step 5: Validate your setup with a dry run

Before you rely on the gate, run it once in dry-run mode. Add the dry-run input to the task and run the pipeline:

```
      - task: RunPentest@1
        inputs:
          awsCredentials: aws-pentest-gate
          regionName: us-west-2
          agentSpaceId: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
          pentestId: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
          dryRun: "true"
```

The dry run checks AWS authentication, the Region, and that your repository resolves to its integration — without starting a penetration test job. It surfaces the common setup mistakes (a bad OIDC trust policy, a Region mismatch, or a repository that is not connected) in seconds. When it passes, remove the dry-run input and continue. For more information, see the section called "Validate your setup before you trust the gate".

> ⚠ **A dry run can become the next run's base commit**
>
> On Azure DevOps the gate falls back to the last successful build of this pipeline when it has no baseline to read, and a dry-run build succeeds. So on a pipeline with no baseline yet — which is the case the first time you run this tutorial — the commit you dry-ran on can become the base of the next real run. The gate then tests only what changed after it, rather than the range you meant to gate.
>
> Dry-run on the commit you intend to gate next, so that the range is the one you want either way. If you've already dry-run on an earlier commit, pin the base on your first real run by adding `baseCommitSha` to the task:
>
> ```
>       - task: RunPentest@1
>         inputs:
>           awsCredentials: aws-pentest-gate
>           regionName: us-west-2
>           agentSpaceId: "as-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
> ```

```
    pentestId: "pt-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    baseCommitSha: "<full 40-character commit hash>"
```

The input pins the base for that run only, and the integration still resolves the head from the commit the pipeline built. For more information, see the section called “Pin the commit range explicitly”.

Step 6: Trigger the gate and read the result

Push a commit to `main` that changes the vulnerable endpoint. This triggers the pipeline: the `DeployStaging` stage deploys the change, then the `SecurityGate` stage starts a penetration test scoped to that change.

Open the pipeline run and watch the gate task:

1. The AWS Toolkit task uses the service connection to obtain temporary AWS credentials through workload identity federation. No stored AWS keys are involved.
2. The task resolves your repository, starts a scoped job, polls it to completion, and evaluates the findings against your severity threshold.

A scoped penetration test typically runs for tens of minutes, so the task runs for a while. That is expected, not a hang.

Because the deployed change introduced a SQL injection vulnerability, AWS Security Agent returns an **in scope** decision with a `HIGH` (or higher) finding. The gate task fails, the `SecurityGate` stage fails, and any later production stage that depends on it does not run.

The run’s **Extensions** tab records the verdict: the scope decision, the severity threshold the run was evaluated against, the finding counts, and whether the gate blocked.



For the full gating behavior — including how authentication failures always block and how you choose fail-closed or fail-open for infrastructure errors — see the section called “Choose how the pipeline responds”.

Step 7: Fix the finding and confirm the gate passes

Fix the vulnerability by using a parameterized query instead of string concatenation. In `app.py`, replace the query construction:

```
    category = request.args.get("category", "")
    conn = sqlite3.connect("products.db")
    cursor = conn.cursor()
    query = "SELECT name, price FROM products WHERE category = ?"
    rows = cursor.execute(query, (category,)).fetchall()
    conn.close()
    return {"products": rows}
```

Push the fix to `main`. The pipeline redeploys the change and runs the gate again against the new commit. With the injection removed, AWS Security Agent finds no findings at or above `HIGH`, the gate task passes, and the production stage proceeds.

The **Extensions** tab shows the same checks for the new commit, with no findings at or above the threshold and a passing verdict.


aws-security-agent-pentest-1790655774468

AWS Security Agent (now part of AWS Continuum) — CI/CD Pentest

Check	Result
Pentest job	pj-bf1e06a2-fc87-4f32-bfbf-890f083acf66
Job status	COMPLETED
Scope decision	IN_SCOPE
Severity threshold	HIGH
Findings	3 total (C:0 H:0 M:1 L:0 I:0 U:2)
At/above threshold	0
Pipeline	✅ pass

Clean up

To avoid ongoing cost and to remove the intentionally vulnerable application:

1. Tear down the sample application from your staging environment.
2. Remove the gate stage from your pipeline, or disable the pipeline, if you do not intend to keep the gate.

3. Delete the AWS service connection if it was created only for this tutorial.
4. If you created the IAM role and OIDC provider only for this tutorial, delete them:

```
aws iam delete-role-policy --role-name my-pentest-gate-role \
  --policy-name securityagent-run-pentest
aws iam delete-role --role-name my-pentest-gate-role
aws iam delete-open-id-connect-provider \
  --open-id-connect-provider-arn arn:aws:iam::111122223333:oidc-provider/
vstoken.dev.azure.com/YOUR_ORGANIZATION_ID
```

Related topics

- the section called “Run penetration tests from your CI/CD pipeline” – concepts and reference for the CI/CD pipeline integration.
- the section called “Tutorial: Gate a GitHub Actions deployment with a scoped penetration test”, the section called “Tutorial: Gate a GitLab CI/CD deployment with a scoped penetration test”, the section called “Tutorial: Gate a Bitbucket Pipelines deployment with a scoped penetration test” – the same tutorial for other CI/CD platforms.
- the section called “Security best practices” – general best practices, including testing against non-production environments.

Review code security findings in pull requests

After enabling code review for pull requests for your repositories, AWS Security Agent automatically analyzes pull requests and posts security findings directly in your source control provider. This allows developers to address security issues within their normal workflow without leaving the pull request.

> **Note**
>
> This page applies to GitHub pull requests, GitLab merge requests, and Bitbucket pull requests. The experience is similar across all providers.

How code review works in pull requests

When you submit a pull request (or merge request in GitLab) in a repository with code review enabled, AWS Security Agent evaluates it against the configured review triggers. If the pull request matches a trigger, analysis begins.

1. **Pull request analysis trigger** – By default, AWS Security Agent starts a code review when a pull request is ready for review. Administrators can limit reviews by target branch or label. They can also start reviews when a label is added. To start a review at any time, comment `@AWS-Continuum-Pentesting review` on the pull request. For more information, see Configure pull request code review triggers.
2. **Analysis acknowledgment** – When the review starts, AWS Security Agent posts a comment to let you know that the review is in progress.
3. **Review completion** – After analysis completes, AWS Security Agent posts a review to your pull request with the results. All security findings are batched together in a single review to keep your pull request organized and minimize notifications.

Understanding code review results

AWS Security Agent provides different types of results depending on what it finds during analysis.

When security issues are found

If AWS Security Agent identifies security issues in your code changes, it posts a review that includes:

- **Summary** - A high-level overview of all security findings, describing the types of issues identified and their potential impact
- **Individual findings** - Detailed security findings appear as threaded comments under the main review, with each finding including:
  - Description of the security issue
  - Location in your code where the issue was found
  - Remediation guidance explaining how to address the issue
  - Relevant context based on your code review settings (security requirement violations, common vulnerabilities, or both)

> **Note**
>
> The types of security issues analyzed depend on your code review settings. If you configured security requirement validation, findings will reference your organization's custom security requirements. If you configured security vulnerability findings, findings will identify common security vulnerabilities. For more information about code review settings, see the section called "Enable pull request code review for GitHub repositories".

When no security issues are found

If AWS Security Agent completes analysis and finds no security issues in your code changes, it posts a comment: "No issues identified." This confirms the review finished successfully and your code changes did not trigger any security findings based on your configured code review settings.

Responding to security findings

After reviewing the security findings posted by AWS Security Agent, you can take action directly in your source control provider.

- **Address findings** - Update your code based on the remediation guidance provided in the findings, then push new commits to the pull request. AWS Security Agent will analyze the updated code.
- **Resolve conversations** - After addressing a security finding, mark the conversation as resolved to track your progress.

> **Tip**
>
> Each finding includes specific remediation guidance tailored to the security issue identified. Review this guidance carefully to understand the security risk and how to address it effectively.

Filtering code review findings

You can customize how AWS Security Agent analyzes your code by adding a `filtering.md` file to your repository. This file allows you to reduce false positives by providing context about your codebase and excluding files or folders from analysis.

Creating the filtering file

Create a file named `filtering.md` in the `.awssecurityagent` directory at the root of your repository:

```
.awssecurityagent/filtering.md
```

AWS Security Agent reads this file from the main branch of your repository (for example, `main` or `mainline`) when analyzing pull requests.

File structure

The `filtering.md` file uses standard Markdown formatting with specific sections that AWS Security Agent recognizes. The file must include a `Code Review` heading followed by one or both of the following sections: `IgnorePatterns` and `ContextHints` (no space).

The following example shows the complete structure of a `filtering.md` file:

```
# filtering.md

## Code Review

### IgnorePatterns

**/*.md

/myapp/src/**/*.snap

/myapp/config/README

### ContextHints

- The backend is a trusted system and won't return non-standard protocols.
- URL is generated from server with presigned token, so no SSRF security
 vulnerabilities.
- AppSec has verified that we are allowed to use cache with an eviction policy.
```

Ignore patterns

The `IgnorePatterns` section specifies files and folders that AWS Security Agent should skip during code review. Use `glob patterns` to define which paths to exclude from analysis.

Format requirements:

- Each pattern must be on its own line.
- Separate each pattern with an empty line between them. This ensures the file renders correctly when viewed in GitHub or code review tools.
- Patterns follow the standard glob format. For example, `**/*.md` matches all markdown files, and `/myapp/src/**/*.snap` matches all `.snap` files inside the `/myapp/src/` folder at the root.
- We support upto 1000 ignore patterns in this section.

Context hints

The `ContextHints` section provides additional context about your codebase that helps AWS Security Agent make more accurate assessments. Use context hints to explain architectural decisions, security exceptions, or other information that might affect how findings are interpreted.

Format requirements:

- Each hint must start with a dash (-) followed by a space.
- Write each hint as a single line of free-form text limited to 500 characters.
- Each hint should describe one specific piece of context about your codebase.
- We support upto 20 context hints in this section.

Context hints are applied after AWS Security Agent completes its initial analysis, helping to filter findings that don't apply to your specific use case.

Next steps

After reviewing code security findings:

- Update your code based on remediation guidance
- Push new commits to trigger re-analysis of your changes
- Adjust code review settings if needed (see the section called "Enable pull request code review for GitHub repositories")
- Review your organization's security requirements to understand validation criteria
- Consider penetration testing for comprehensive security validation of deployed applications

Create a code review

Create code reviews in the AWS Security Agent web application to scan your source code repositories and S3 sources for security vulnerabilities. Code reviews perform comprehensive static analysis across your entire codebase, identifying security issues and providing remediation guidance.

Unlike pull request-based code review which analyzes individual code changes (see the section called “Review code security findings in pull requests”), on-demand code reviews scan your full source code to identify security vulnerabilities and validate compliance with your organization’s security requirements.

In this procedure, you’ll create a code review by selecting source code inputs, configuring permissions, and running the review.

Prerequisites

Before you begin, ensure you have:

- Access to the AWS Security Agent web application
- At least one connected GitHub repository or S3 bucket in your Agent Space

> **Tip**
>
> If you already have GitHub repositories connected to your Agent Space, code review is ready to use — no additional setup is required. Choose **Start in web app** from the **Code review** card on your Agent Space page, or launch the web application directly.
> If you need to connect additional sources or configure S3 buckets, see the section called “Enable code review”.

Access the code reviews page

Navigate to the code reviews section in the web application.

1. Log in to the AWS Security Agent web application.
2. In the left sidebar, choose **Code reviews**.

3. You see a list of existing code reviews with their source information, last run status, and findings summary.

Create a code review

Set up a new code review by configuring its source code inputs and permissions.

1. On the **Code reviews** page, choose **Create code review**.

Configure code review details

Provide a title and select the source code to review.

1. In the **Title** field, enter a descriptive name for your code review.

> **Tip**
>
> Use a name that identifies the application, repository, or scope of the review. For example, "billing-service-security-review" or "infrastructure-code-audit".

2. In the **Sources** section, select the source code inputs for this review. Choose from two tabs:

GitHub repositories

Select from repositories connected to your Agent Space.

1. Choose the **GitHub repositories** tab.
2. In the **Integrated repositories** table, select the checkbox next to each repository you want to include in the review.
3. Use the search field to find specific repositories by name.

> **Note**
>
> Only repositories connected to your Agent Space through the code review configuration appear here. To add more repositories, choose **Manage in your Admin console** or ask your administrator to update the Agent Space configuration.

> **Tip**
>
> When you select a repository, you can specify a branch. By default, the service uses the primary branch. To use a different branch, enter the branch name in the field next to the repository.

S3 sources

Select ZIP files from the S3 buckets connected to your Agent Space. Your Agent Space administrator configures which S3 buckets are available. Any ZIP file stored in one of those buckets can be used as a source for a code review.

1. Choose the **S3 sources** tab.
2. Enter the S3 URI of each ZIP file you want to include in the review. You can add up to 30 S3 sources.

> **Note**
>
> S3 sources must be ZIP files stored in S3 buckets that are connected to your Agent Space. To make additional buckets available, see the section called “Enable code review”.

Configure permissions

Select the IAM service role and optional CloudWatch log group for this code review.

1. In the **Permissions** section, locate the **Service role** dropdown.
2. Select the IAM role from your configured service roles.

> **Note**
>
> The service role must have permissions to access your source code in S3 and write to CloudWatch logs, and any other AWS resources needed for the code review. Service roles are configured during code review setup in the AWS Management Console.

3. (Optional) In the **CloudWatch log group** dropdown, select a log group to store code review execution logs.

> **Note**
>
> If you don't select a log group, AWS Security Agent creates a default log group for storing code review logs.

Configure automatic code remediation

Enable automatic remediation to have AWS Security Agent generate code fixes for all findings as soon as the review completes.

1. In the **Automatic code remediation** section, select the **Enable automatic code remediation** checkbox.

How AWS Security Agent delivers the fix depends on the source:

- **Connected private repository** – AWS Security Agent submits a pull request (or a merge request, for GitLab) with the fix.
- **Public repository** – To avoid disclosing the vulnerability before it's fixed, AWS Security Agent does not open a pull request. Delivery varies by provider — for example, public GitHub repositories receive a downloadable diff you can apply privately; see the provider's connection topic.
- **Amazon S3 sources** – AWS Security Agent attaches a downloadable code diff to the finding (there's no connected repository to open a pull request against). Download it from the web application and apply it locally.

> **Important**
>
> Everyone with read access to the private repository can see the remediation pull request. Review the changes before merging.

> **Note**
>
> When automatic code remediation is disabled, you can still trigger it manually for individual findings. This works for findings from a connected repository or an Amazon S3 source.

Configure simulated validation

Enable simulated validation to dynamically confirm whether discovered vulnerabilities are exploitable in a running application.

1. In the **Simulated validation** section, select the **Enable simulated validation** checkbox.

When enabled, AWS Security Agent provisions a simulated environment after static analysis completes. It onboards your source code, starts services inside the environment, and attempts to exploit vulnerabilities found during the scan. Findings are then updated with a validation status indicating whether the vulnerability was successfully exploited.

> **Note**
>
> Simulated validation is currently available for self-contained dockerizable applications only.

> **Important**
>
> Simulated validation adds processing time to your code review run. The validation step provisions an environment and runs exploitation attempts. This typically takes 1–3 hours depending on the number of findings and application complexity.

Allowed network destinations

The simulated environment has outbound network access restricted to a predefined allowlist. Your application can reach the following domains during validation:

The following table lists the allowed domains and their purposes.

Domain	Purpose
`.amazonaws.com` , `.aws.amazon.com`	AWS services
`.public.ecr.aws`	Amazon ECR Public
`.docker.com` , `.docker.io`	Docker Hub
`.github.com` , `.githubusercontent.com`	GitHub
`.gitlab.com`	GitLab
`.npmjs.com` , `.npmjs.org`	npm registry
`.pypi.org` , `.pypi.python.org` , `.pythonhosted.org`	Python Package Index
`.crates.io` , `.rustup.rs`	Rust packages
`.maven.org` , `.gradle.org`	Java/Gradle packages
`.nuget.org`	.NET packages
`.rubygems.org` , `.ruby-lang.org`	Ruby packages
`.golang.org` , `.pkg.go.dev` , `.goproxy.io`	Go packages
`.nodejs.org` , `.yarnpkg.com`	Node.js

Domain	Purpose
`.alpinelinux.org` , `.debian.org` , `.ubuntu.com` , `.centos.org` , `.fedoraproject.org`	Linux distribution repositories
`.cloudfront.net`	CloudFront distributions
`.google.com` , `.googleapis.com`	Google APIs
`.microsoft.com` , `.visualstudio.com`	Microsoft services
`.sourceforge.net` , `.bitbucket.org`	Source hosting

Note

If your application requires network access to domains not on this list, the connection will be blocked by the network firewall. Applications that depend on external APIs or services not listed above may not start correctly in the simulated environment.

Create the code review

1. Review your configuration to ensure accuracy.
2. Choose **Create code review**.

You are redirected to the code review detail page where you can start a review run.

Run a code review

After creating a code review, start a run to begin the analysis.

1. On the code review detail page, choose **Start review**.

2. AWS Security Agent begins analyzing your source code.

You can also start a review from the **Code reviews** list page by choosing **Start review** next to the code review you want to run.

Monitor a code review run

Track the progress of your code review as it executes.

Review run phases

A code review run progresses through the following phases, displayed as a progress indicator:

1. **Preflight** – AWS Security Agent validates access to your source code and sets up the testing environment. The preflight checks include:
   - Service infrastructure setup
   - S3 source access validation
   - Setup testing environment
2. **Static analysis** – AWS Security Agent scans your source code for security vulnerabilities and requirement violations.
3. **Simulated validation** (optional) – If simulated validation is enabled, AWS Security Agent provisions a simulated environment and deploys your application. It then attempts to exploit the vulnerabilities discovered during static analysis. This step confirms whether findings are exploitable in the target runtime. This phase only appears when you enable simulated validation during code review creation.
4. **Finalizing** – AWS Security Agent compiles findings and generates the results summary.

View run details

On the run detail page, navigate between tabs to monitor progress:

- **Code review run** – View the run summary including run ID, creation time, status, duration, task hours, severity level breakdown, and risk types chart.
- **Preflight** – View the preflight check progress and status of each validation step.
- **Code review logs** – View the tasks AWS Security Agent identified and conducted during the review, with detailed task logs for each step.

- **Simulated validation** – When simulated validation is enabled, view the provisioning status, validation tasks for individual findings, and their results. The provisioning task includes a log of the simulated environment build, including each setup step and any build errors. AWS Security Agent updates the log while the environment builds. It also writes the log to the code review's CloudWatch log group.
- **Findings** – View security findings after the review completes (see the section called "Review findings from a code review").

Run history

Each code review maintains a history of all runs. On the code review detail page:

- The **Latest run** section shows the most recent run with its start time, status, duration, and ID.
- The **All runs** table lists all previous runs with their start time, status, duration, findings summary, and ID.
- Choose **Monitor run** to view the details of the latest active run.
- Choose any run's start time link to view its full details.

Next steps

After running a code review:

- Review security findings and their remediation guidance (see the section called "Review findings from a code review")
- Remediate findings through automated pull requests or manual fixes (see the section called "Remediate code review findings")
- Run additional reviews after implementing fixes to verify remediation
- Adjust your code review configuration or sources as your codebase evolves

Review findings from a code review

After a code review run completes, review the run summary and security findings to understand vulnerabilities in your source code. Each finding contains a description, severity rating, code locations, risk reasoning, and suggested fixes to help you prioritize and address security issues.

Prerequisites

Before you begin, ensure you have:

- A completed or in-progress code review run
- Access to the AWS Security Agent web application

Step 1: Access the code review run

Navigate to your completed code review run to view the summary and findings.

1. Log in to the AWS Security Agent web application.
2. In the left sidebar, choose **Code reviews**.
3. Select the code review you want to examine.
4. In the **All runs** table, select the completed run by clicking its start time link. Alternatively, choose **Monitor run** to view the latest run.

Step 2: Monitor run progress

Track the progress of your code review run using the step indicator.

1. Locate the horizontal step indicator below the page header.
2. Review the status of each phase:
   - **Preflight** – Validates access to your source code and sets up the scanning environment. Checks include service infrastructure setup, S3 source access validation, and testing environment setup, which includes GitHub access checks.
   - **Static analysis** – Scans your source code for security vulnerabilities and requirement violations.
   - **Simulated validation** (optional) – When enabled, provisions a simulated environment and attempts to exploit discovered vulnerabilities against the running application.
   - **Finalizing** – Compiles findings and generates the results summary.

> **Note**
>
> Each step displays a status indicator (Completed or In progress). The run is complete when all phases show Completed. The simulated validation step only appears when you enable simulated validation during code review creation.

Step 3: Review the run summary

Navigate to the **Code review run** tab to view the high-level results.

1. Select the **Code review run** tab.
2. The **Run summary** section provides run status, duration and other high level details. It also provides a dashboard of security findings categorized by severity level and risk-types.
3. Application overview by AWS Security Agent provides a summary of the code review scan when the run is completed

Step 4: Review preflight results

Navigate to the **Preflight** tab to verify that all source access checks passed.

1. Select the **Preflight** tab.
2. Review the **Preflight progress** indicator showing the number of checks completed.
3. Verify that each check shows a success status:
   - **Service infrastructure setup** – Confirms the testing environment is ready
   - **S3 Source Access Validation** – Confirms access to S3 source code (if applicable)
   - **Setup Testing Environment** – Confirms the analysis environment is configured

> **Note**
>
> If any preflight check fails, the run will not proceed to static analysis. Review the check details to identify and resolve access issues, then start a new run.

Step 5: Review code review logs

Navigate to the **Code review logs** tab to view the tasks AWS Security Agent performed during analysis.

1. Select the **Code review logs** tab.
2. Browse the list of tasks in the left panel.
3. Select a task to view its detailed log in the right panel.

> **Tip**
>
> Use the search field to filter tasks by name. The logs provide insight into what AWS Security Agent examined and how it reached its conclusions.

Step 6: Navigate to findings

Select the **Findings** tab to view all security findings from the run.

> **Note**
>
> **Default confidence filter**
>
> By default, you see only findings with **High** agent confidence. To also show findings with **Medium** or **Low** agent confidence and false positives, turn off the **Hiding unverified findings** toggle.

1. Select the **Findings** tab.
2. The findings display in a split view with the findings list on the left and the selected finding's details on the right.
3. Review the information displayed on each finding card:
   - **Finding title** – A descriptive name summarizing the security issue
   - **Severity badge** – Color-coded risk level (Critical, High, Medium, or Low). For what each level means, see the section called "Severity ratings and CVSS metrics".
   - **Last updated** – Timestamp of when the finding was last updated

Step 7: Review finding details

Select individual findings to view comprehensive information about each vulnerability.

1. Select a finding in the left panel to display its details in the right panel.
2. Review the available actions:
   - **Resolve finding** – Mark the finding as resolved after you've addressed it
   - **Remediate code** – Generate a fix: a pull request or merge request for a connected private repository, or a downloadable code diff for an Amazon S3 source
3. Provide feedback using **Was this finding accurate?** – Select **Yes** or **No** to help improve future analysis accuracy.
4. Review the key attributes in the **Overview** section:
   - **Agent confidence** – The confidence level AWS Security Agent has in this finding
   - **Severity** – The risk level with a color-coded badge
   - **Risk Type** – The category of security risk
   - **Validation status** – When simulated validation is enabled, indicates the current state or outcome of attempting to reproduce a finding in an isolated runtime environment.
     - **Validating** – Simulated validation is currently in progress.
     - **Confirmed** – The vulnerability was successfully exploited in the simulated environment.
     - **Not reproduced** – The vulnerability could not be exploited in the simulated environment.
     - **Validation failed** – Simulated validation was inconclusive because of an error or timeout. See the finding description for details.
     - **Not validated** – Simulated validation was not performed for this finding. This occurs when validation is not enabled or the validation step did not reach this finding.
   - **Resolved** – Whether the finding has been resolved (Yes/No)
   - **Last updated** – Timestamp of the most recent update
5. Expand the **Description** section to read:
   - A detailed explanation of the security issue
   - How the vulnerability could be exploited
   - The potential impact on your application
   - Verification steps the agent performed to confirm the finding
6. Expand the **Code locations** section to view:
   - Specific file paths where the issue was identified

- A brief description of what was found at each location
- Line number badges linking to the relevant code

7. Expand the **Evidence** section to review:
   - The specific code, configuration, or observations that support the finding
   - A short explanation of why each item demonstrates the vulnerability
   - The affected files and line numbers for each piece of evidence
8. Expand the **Suggested fix** section to view:
   - The changes AWS Security Agent recommends to remediate the finding
   - Example code or configuration for each recommended change

> **Tip**
>
> Use the code locations to quickly navigate to the affected files in your repository. Each location includes enough context to understand the issue without reading the entire file.

Step 8: Prioritize and address findings

Validate and take action on findings to remediate vulnerabilities and improve your application's security posture.

For **High** severity findings with **High** agent confidence:

1. Review the Description and Code locations sections thoroughly.
2. Use **Remediate code** to generate a pull request with a fix, or review automatic remediation PRs if you enabled that option.
3. Plan a follow-up code review run to verify the fix is effective.

For **Medium** severity findings with **High** agent confidence:

1. Prioritize based on your risk tolerance and business context.
2. Include remediation tasks in your regular development sprint planning.
3. Consider whether multiple medium-severity findings together create higher risk.

For **Medium** or **Low** confidence findings:

1. Review the Description and Code locations sections to validate the assumptions and conditions under which the vulnerability occurs.
2. If vulnerability is valid, prioritize based on severity and risk tolerance and consider remediation tasks.

Step 9: Track remediation progress

Use the findings interface and re-runs to track which vulnerabilities have been addressed.

1. As you implement fixes, use **Resolve finding** to mark findings as addressed.
2. Run a new code review to verify that fixes resolve the findings and don't introduce new issues.
3. Review the **All runs** table on the code review detail page to compare findings across runs over time.

> **Tip**
>
> After merging remediation pull requests, start a new run of the same code review to confirm the vulnerabilities are resolved. Compare the findings count between runs to track your progress.

Next steps

After reviewing your code review findings:

- Prioritize high-severity findings with high confidence for immediate remediation
- Create a Jira issue for a finding so your team can track it in Jira. For information about creating a Jira issue to track a finding, see Create Jira issues for findings.
- Use **Remediate code** to generate fixes for findings from a connected source code repository or an Amazon S3 source (see the section called "Remediate code review findings")
- Run additional code reviews after implementing fixes to verify remediation
- Adjust your code review sources or settings as your codebase evolves (see the section called "Enable code review")

Remediate code review findings

After reviewing security findings from a code review, you can use AWS Security Agent to generate code fixes. Supported sources include connected third-party source code providers (such as GitHub, GitLab, or Bitbucket) and Amazon S3. For a connected private repository, AWS Security Agent opens a pull request (or a merge request, for GitLab) with the proposed fix. For Amazon S3 sources, it attaches a downloadable code diff that you can apply locally. For the list of supported providers and how to connect them, see the section called “How integrations work with Agent Spaces”.

Prerequisites

Before you begin, ensure you have:

- A completed code review run with findings
- A source for the code review — a connected source code repository (see the section called “How integrations work with Agent Spaces”) or source code uploaded to Amazon S3
- Access to the AWS Security Agent web application
- Familiarity with your application’s architecture and security requirements

How code remediation works

When you trigger code remediation for a finding, AWS Security Agent analyzes the finding and its code locations, then generates a code fix. How the fix is delivered depends on the source:

- **Connected private repository** – AWS Security Agent opens a pull request (or a merge request, for GitLab) on the repository. It includes a description of the security issue and the changes made. This applies to the source code providers you’ve connected — see the section called “How integrations work with Agent Spaces”.
- **Amazon S3 source** – AWS Security Agent attaches a suggested code diff to the finding. Because there’s no connected repository to open a pull request against, download the code diff from the web application and apply it locally with `git apply /path/to/code_remediation_changes.diff`.
- **Public repository** – To avoid disclosing the vulnerability before it’s fixed, AWS Security Agent doesn’t open a pull request. Delivery depends on the provider. For example, public GitHub repositories receive a downloadable code diff. Automated remediation isn’t available for public GitLab or Bitbucket repositories. See the provider’s connection topic for details.

Important

Pull requests created by AWS Security Agent are visible to all users who have read access to the repository. Review the changes before merging to ensure they align with your application's requirements.

Automatic code remediation

If you enabled **automatic code remediation** when creating the code review, AWS Security Agent generates fixes for all eligible and confirmed findings as soon as the review completes. You don't need to take any additional action. AWS Security Agent delivers each fix the same way as an on-demand fix, using the delivery methods described earlier in this topic.

Automatic code remediation covers both connected source code repositories and Amazon S3 sources.

Manual code remediation

You can trigger remediation for an individual finding on demand, whether its source is a connected repository or an Amazon S3 upload. Use this when automatic code remediation is disabled, or to remediate a specific finding.

1. Navigate to the **Findings** tab of a completed code review run.
2. Select the finding you want to remediate.
3. In the finding detail panel, choose **Remediate code**.
4. AWS Security Agent generates a code fix and delivers it based on the source: a pull request or merge request for a connected private repository, or a downloadable code diff for an Amazon S3 source.

Review remediation pull requests

Note

If a finding is already linked to a Jira issue, AWS Security Agent also adds the remediation pull request as a link on that issue. To make sure the remediation pull request is linked, link

the finding to a Jira issue before remediation finishes. For more information, see Create Jira issues for findings.

After AWS Security Agent submits a remediation pull request (or merge request) to a connected private repository:

1. Navigate to the repository on your source code provider.
2. Locate the pull request (or merge request) created by AWS Security Agent.
3. Review the changes, including:
   - The description explaining the security finding and fix
   - The code changes addressing the vulnerability
   - Any relevant context about the remediation approach
4. Merge the pull request if the fix is appropriate, or close it and implement an alternative solution.

Tip

After merging remediation pull requests, start a new code review run to verify that the fixes resolve the findings and don't introduce new security issues.

Apply remediation diffs

For findings delivered as a downloadable code diff (Amazon S3 sources, and public repositories where supported), apply the diff locally.

1. In the finding detail panel, download the code diff from the **Code remediation** section.
2. In your local copy of the source code, run `git apply /path/to/code_remediation_changes.diff`.
3. Review the applied changes, test them against your application, and commit them through your normal development workflow.

Limitations

- Code remediation is available for findings from connected source code repositories and Amazon S3 sources, both automatically and on demand. For public repositories, availability varies by

provider — see the section called "How integrations work with Agent Spaces" and the provider's connection topic.

- AWS Security Agent generates fixes based on its analysis of the vulnerability. Review all changes before merging or committing them to ensure they're appropriate for your application.
- Some complex findings may require manual intervention beyond the automated fix.

Next steps

After remediating findings:

- Review and merge pull requests (or merge requests) in your connected repositories, or commit applied diffs through your normal workflow
- Run a new code review to verify fixes and check for remaining issues
- Resolve findings in the web application after confirming remediation
- Adjust your code review sources or settings as needed (see the section called "Enable code review")

Create Jira issues for findings

AWS Security Agent provides the ability to create Jira issues directly from security findings, so your team can track and resolve them in Jira. You can create issues for findings from a penetration test or a code review. For information about connecting Jira, see Connect AWS Security Agent to Jira.

When you create an issue, AWS Security Agent creates one Jira issue for each finding you select, in the project you choose. The issue includes the finding's details, such as its description and suggested fix.

Before you begin, make sure you have connected a Jira project to an Agent Space, and that the project allows the **Task** issue type.

> **Issue type requirement**
>
> The Jira project you choose must allow the **Task** issue type.

Creating issues for findings

Create Jira issues from your findings so your team can track and remediate security and code review issues directly in Jira. This procedure assumes you have already connected a Jira project to an Agent Space.

1. Sign in to the AWS Security Agent web application.
2. Open the penetration test or code review run that contains the findings.
3. On the **Findings** tab, choose **Multiselect Findings**.
4. In the findings list, select the check box next to each finding you want to create an issue for.
5. On the **Findings** tab, choose **Create Issues**.
6. Select the Jira project to create the issues in, then confirm.

   AWS Security Agent creates a Jira issue for each selected finding. This might take a few moments if you selected multiple findings. After an issue is created, the finding shows an **Issue** section with the issue details.

Opening a linked issue in Jira

If a finding has an associated Jira issue, you can open it to view or update its status. To open a linked issue in Jira:

1. Open the finding.
2. Expand the **Issue** section.
3. Choose the **Issue key** to open the issue in Jira.

Unlinking an issue from a finding

If you no longer want a finding linked to its Jira issue, you can remove the link. Unlinking removes the link only; it does not delete the issue in Jira. To unlink an issue from a finding:

1. On the **Findings** tab, choose **Multiselect Findings**.
2. On the **Findings** tab, select the findings you want to unlink.
3. Choose **Unlink Issues**, then confirm.

Run a differential code scan with S3

Run a differential (diff) code scan to analyze only the changed lines in your source code, rather than performing a full repository scan. Differential scans are faster than full scans and produce findings targeted to specific code changes, making them ideal for pre-merge validation in development workflows.

Unlike pull request-based code review which is triggered automatically by third-party source control provider events (see the section called “Review code security findings in pull requests”), differential scans are initiated programmatically through the AWS Security Agent API or SDK. You upload a unified diff file to S3 and reference it when starting a code review job.

How differential scans work

A differential scan analyzes your code changes in the full context of your repository. When you create a code review resource with your source code uploaded to S3, the differential scan uses the complete repository as context while focusing findings specifically on the changed lines in your diff. This enables the scan to identify security issues that arise from how your changes interact with existing code — such as broken authentication flows, insecure data handling across modules, or changes that expose existing vulnerabilities.

The scan process: . You upload a unified diff file (the output of `git diff`) to an S3 bucket connected to your Agent Space. . You call the `StartCodeReviewJob` API with a `diffSource` parameter pointing to the S3 location of your diff. . AWS Security Agent analyzes the changed code in the diff for security vulnerabilities and compliance with your organization’s security requirements. . Findings focus on the changed lines and can also point to existing code that the changes make reachable, with severity ratings, code locations, and remediation guidance.

Prerequisites

Before you begin, ensure you have:

- An Agent Space with code review enabled (see the section called “Enable code review”)
- A code review resource already created for the target repository, with the full source code uploaded to the S3 bucket connected to your Agent Space. The full repository provides context that enables deeper analysis of how your changes interact with existing code. See the section called “Create a code review” for instructions on creating a code review and uploading source code.

- An S3 bucket connected to your Agent Space
- IAM permissions to upload to the S3 bucket and call `securityagent:StartCodeReviewJob`
- A unified diff file generated from your code changes (for example, the output of `git diff main..feature-branch`)

> **Tip**
>
> For the most accurate results, ensure your code review resource has the latest version of your full source code uploaded to S3. The diff scan uses this full repository as context to understand your application architecture, data flows, and existing security controls when analyzing the changed lines.

Step 1: Generate a diff file

Generate a unified diff that represents the code changes you want to scan.

```
git diff main..feature-branch > changes.diff
```

Alternatively, generate a diff for staged changes:

```
git diff --cached > changes.diff
```

> **Tip**
>
> The diff file should be in standard unified diff format. AWS Security Agent parses the file paths and changed lines from the diff headers to scope its analysis.

Step 2: Upload the diff to S3

Upload the diff file to an S3 bucket that is connected to your Agent Space.

```
aws s3 cp changes.diff s3://my-security-agent-bucket/diffs/changes.diff
```

Important

The S3 bucket must be one that is already connected to your Agent Space. The IAM service role associated with your Agent Space must have read access to this location. See the section called "Enable code review" for instructions on connecting S3 buckets.

Step 3: Start a differential code review job

Call the `StartCodeReviewJob` API with the `diffSource` parameter to initiate a differential scan.

Using the AWS CLI

```
aws securityagent start-code-review-job \
  --agent-space-id "your-agent-space-id" \
  --code-review-id "your-code-review-id" \
  --diff-source '{"s3Uri": "s3://my-security-agent-bucket/diffs/changes.diff"}'
```

Using the AWS SDK (Python)

```
import boto3

client = boto3.client('securityagent')

response = client.start_code_review_job(
    agentSpaceId='your-agent-space-id',
    codeReviewId='your-code-review-id',
    diffSource={
        's3Uri': 's3://my-security-agent-bucket/diffs/changes.diff'
    }
)

print(f"Job started: {response['codeReviewJobId']}")
```

Using the AWS SDK (JavaScript/TypeScript)

```
import { SecurityAgentClient, StartCodeReviewJobCommand } from '@aws-sdk/client-
securityagent';

const client = new SecurityAgentClient({ region: 'us-east-1' });
```

```
const response = await client.send(new StartCodeReviewJobCommand({
  agentSpaceId: 'your-agent-space-id',
  codeReviewId: 'your-code-review-id',
  diffSource: {
    s3Uri: 's3://my-security-agent-bucket/diffs/changes.diff',
  },
}));

console.log(`Job started: ${response.codeReviewJobId}`);
```

The API returns immediately with a `codeReviewJobId` and `status` of `IN_PROGRESS`.

Step 4: Monitor the scan

Poll the code review job status until it completes.

```
aws securityagent get-code-review-job \
  --agent-space-id "your-agent-space-id" \
  --code-review-id "your-code-review-id" \
  --code-review-job-id "the-job-id"
```

The job progresses through the same phases as a full code review: Preflight → Static analysis → Finalizing. Differential scans typically complete faster than full scans because they analyze fewer lines of code.

Step 5: Review findings

After the job completes, list findings for the code review job.

```
aws securityagent list-findings \
  --agent-space-id "your-agent-space-id" \
  --code-review-id "your-code-review-id" \
  --code-review-job-id "the-job-id"
```

Each finding includes:

- A description of the security issue
- The severity level (Critical, High, Medium, or Low)
- Code locations referencing specific lines in the diff
- Risk reasoning explaining the potential impact

- Remediation guidance

For more information about understanding and acting on findings, see the section called "Review findings from a code review".

Quotas and limits

- Maximum number of changed files in a diff: 3,000 files or 5 MB total diff size
- Findings are capped at 30 per scan, prioritized by severity (Critical > High > Medium > Low)

Next steps

After running a differential scan:

- Review findings and apply remediation guidance (see the section called "Review findings from a code review")
- Enable pull request comments for automated code review on every pull request (see the section called "Enable pull request code review for GitHub repositories")
- Run a full code review periodically to catch issues outside of individual changes (see the section called "Create a code review")
- Use the IDE integration to run differential scans directly from your development environment (see the section called "Run code security scans from your IDE")

Run code security scans from your IDE

Run code security scans directly from your IDE using Kiro or Claude Code. The IDE integration lets you scan your local source code for security vulnerabilities, run differential scans on only changed code, and perform threat model reviews on design documents — all without leaving your development environment. Findings appear alongside your code with remediation guidance, and you can apply automated fixes directly from the IDE.

How IDE integration works

The IDE integration uses the AWS Security Agent MCP (Model Context Protocol) server to connect your IDE to the AWS Security Agent service:

1. The MCP server packages your source code into a ZIP archive.

2. The archive is uploaded to an S3 bucket that the server auto-provisions in your AWS account.
3. The server calls the AWS Security Agent API to start a scan.
4. AWS Security Agent analyzes the code for security vulnerabilities and compliance with your organization's security requirements.
5. Findings are returned to the IDE with code locations, descriptions, severity ratings, and remediation suggestions.

The MCP server handles all orchestration — Agent Space provisioning, S3 bucket creation, IAM role setup, code packaging, and scan polling — so you only need AWS credentials configured locally.

> **Note**
>
> The only prerequisite is AWS credentials configured in your local environment (for example, through `aws configure`, AWS SSO, or environment variables). The MCP server auto-provisions the Agent Space, IAM service role, and S3 bucket on first use.

Prerequisites

Before you begin, ensure you have:

- AWS credentials configured locally with the following permissions:
  - `iam:CreateRole`, `iam:PutRolePolicy` (for one-time setup of the service role)
  - `s3:CreateBucket`, `s3:PutObject`, `s3:PutPublicAccessBlock`, `s3:PutLifecycleConfiguration`
  - `securityagent:CreateAgentSpace`, `securityagent:UpdateAgentSpace`, `securityagent:ListAgentSpaces`, `securityagent:BatchGetAgentSpaces`
  - `securityagent:CreateCodeReview`, `securityagent:StartCodeReviewJob`, `securityagent:BatchGetCodeReviewJobs`
  - `securityagent:ListFindings`, `securityagent:BatchGetFindings`
  - `securityagent:StartCodeRemediation` (for automated fixes)
  - `sts:GetCallerIdentity`
- uv installed (Python package runner)
- Python 3.10 or later

- One of the following IDEs:
  - Kiro (install the AWS Security Agent Power)
  - Claude Code (install the AWS Security Agent plugin)

Install the MCP server

Kiro

Install the **AWS Security Agent** Power from the Kiro marketplace. The Power bundles the MCP server configuration, steering instructions, and lifecycle hooks for automatic security scan suggestions.

Alternatively, configure the MCP server manually in your project's `.kiro/mcp.json`:

```
{
  "mcpServers": {
    "security-agent": {
      "command": "uvx",
      "args": ["awslabs.security-agent-mcp-server@latest"],
      "env": {
        "AWS_PROFILE": "default",
        "AWS_REGION": "us-east-1",
        "FASTMCP_LOG_LEVEL": "ERROR"
      }
    }
  }
}
```

Claude Code

Install the AWS Security Agent plugin, which provides skills for setup, full scans, diff scans, threat model reviews, and remediation.

Configure the MCP server in your project's `.mcp.json`:

```
{
  "mcpServers": {
    "awslabs.security-agent-mcp-server": {
      "command": "uvx",
      "args": ["awslabs.security-agent-mcp-server@latest"],
      "env": {
```

```
        "AWS_PROFILE": "default",
        "AWS_REGION": "us-east-1",
        "FASTMCP_LOG_LEVEL": "ERROR"
      }
    }
  }
}
```

Tip

You can also run the MCP server via Docker if you prefer not to install Python locally. See the MCP server documentation for Docker configuration.

First-time setup

On first use, the MCP server automatically provisions the required AWS resources:

Resource	Name convention	Purpose
Agent Space	User-chosen or auto-created	Container for scans, reviews, and penetration tests
IAM service role	`SecurityAgentScanRole`	Assumed by `securityagent.amazonaws.com` to read uploaded code
S3 bucket	`security-agent-scans-{account}-{region}`	Stores zipped source code (30-day auto-expiry lifecycle)

To trigger setup explicitly, ask your IDE:

```
Set up the security agent
```

The setup verifies your AWS credentials, creates or reuses an Agent Space, provisions the IAM role and S3 bucket, and confirms readiness. If you already have Agent Spaces from the AWS Management Console, the setup shows them and asks which one to use.

> **Note**
>
> Setup runs automatically on your first scan if not already configured. You don't need to run it separately.

Run a full security scan

Scan your entire project for security vulnerabilities:

```
Scan this project for security issues
```

The IDE:

1. Archives your source code (excluding `.git`, `node_modules`, build artifacts, and other non-essential directories)
2. Uploads the archive to S3
3. Starts a full code review job
4. Polls for completion (typically around 1 hour)
5. Presents findings grouped by severity

Scan progress

Full scans typically take around 1 hour depending on codebase size. The IDE checks progress every 5 minutes and notifies you when results are ready. You can ask for status at any time:

```
How's the scan going?
```

Run a differential scan

For faster feedback during development, scan only the code that changed since a git ref:

```
Scan my changes against main for security issues
```

By default, if you don't specify a base ref, the scan compares your uncommitted changes against HEAD. You can explicitly specify a different base:

```
Diff scan my uncommitted changes
```

Differential scans upload both the full repository context and the git diff patch, then run analysis focused only on the changed lines. Results typically arrive in 5–15 minutes.

For more information about the S3 diff scan API, see the section called “Run a differential code scan with S3”.

Run a threat model review

Analyze design documents for security-posture changes using STRIDE methodology:

```
Run a threat model review on my spec
```

The IDE identifies your `requirements.md` and `design.md` files (typically under `.kiro/specs/`), uploads them alongside your source code, and runs a threat model analysis. Results identify:

- Security threats categorized by STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege)
- Severity ratings for each threat
- Impact on specific assets in your codebase
- Recommendations for mitigation

> **Tip**
>
> Run a threat model review before generating implementation tasks from a spec. This catches security design issues before code is written.

Review findings

When a scan completes, findings appear in your IDE grouped by severity:

```
# CRITICAL: SQL Injection in user-service.ts
   File: src/api/user-service.ts:45
   User input flows directly into SQL query without parameterization
```

```
# HIGH: Hardcoded credentials in config.ts
   File: src/config/database.ts:12
   Database password stored in source code
```

A detailed report is also written to `.security-agent/findings-{scan_id}.md` with full information for each finding including risk type, confidence score, code locations, and remediation code.

To view findings from a previous scan:

```
Show my security findings
```

Apply automated fixes

After reviewing findings, apply automated remediation:

```
Fix the security findings
```

The IDE works through findings top-down by severity (Critical → High → Medium → Low), applying code fixes using the remediation guidance from each finding. After all fixes are applied, it automatically runs a verification diff scan to confirm the issues are resolved.

You can also fix individual findings:

```
Fix the SQL injection in user-service.ts
```

> **Important**
>
> Always review automated fixes before committing. Verify that fixes don't break existing functionality or introduce regressions.

Automatic scan suggestions (Kiro)

When using the Kiro Power, AWS Security Agent can automatically suggest diff scans after you make security-sensitive code changes. The Power installs a hook that evaluates each completed coding turn and suggests a scan when changes affect:

- Authentication or authorization logic

- Cryptography or secrets handling
- Input validation or data sanitization
- Network requests or external integrations
- Security configuration (CORS, headers, session handling)

You can opt out of automatic suggestions at any time by deleting `.kiro/hooks/security-diff-scan-suggester.kiro.hook`.

Available scan types

Scan type	Duration	Use case	Command
Full scan	Around 1 hour	Comprehensive review of entire codebase	"Scan my project for security issues"
Diff scan	5–15 minutes	Changed code only (pre-commit, pre-PR)	"Scan my changes" or "Diff scan against main"
Threat model	5–30 minutes	Design docs (requirements.md, design.md)	"Run a threat model review on my spec"

Environment variables

Variable	Description	Default
`AWS_REGION`	AWS region for SecurityAgent API calls	`us-east-1`
`AWS_PROFILE`	AWS credential profile name	Default profile
`FASTMCP_LOG_LEVEL`	MCP server log level (DEBUG, INFO, WARNING, ERROR)	`WARNING`

Troubleshooting

"Not configured. Run setup first."

Your `.security-agent/config.json` is missing or the Agent Space no longer exists. Ask the IDE to run setup:

```
Set up the security agent
```

Scan fails with AccessDenied

Ensure your AWS credentials have the required IAM permissions listed in the Prerequisites section. The most common missing permission is `iam:CreateRole` (needed only for first-time setup).

Scan times out or takes too long

- Full scans on large codebases can take more than 1 hour
- Use differential scans for iterative development — they complete in minutes
- Check that your source archive doesn't include unnecessary directories (the MCP server excludes common patterns automatically)

Empty diff — no changes to scan

If you run a diff scan with no uncommitted changes, the MCP server reports "No changes vs HEAD" and does not start a scan. Commit or stage your changes first, or specify a different base ref.

Quotas and limits

- Maximum source archive size: 2 GB
- S3 bucket lifecycle: uploaded code auto-deleted after 30 days

Next steps

After running your first IDE security scan:

- Enable pull request code review comments for automated GitHub integration (see the section called "Enable pull request code review for GitHub repositories")

- Configure security requirements for organization-specific policy validation (see the section called "Manage security requirements")
- Run periodic full scans to catch issues across your entire codebase (see the section called "Create a code review")
- Use threat model reviews on new feature specs before implementation

Create a threat model

Create threat models in the AWS Security Agent web application to analyze your application's architecture and identify security threats. A threat model produces two main outputs: a **system overview** that describes how your system is built and behaves, and a set of **threats** that describe how the system could be attacked, each with a severity level, STRIDE classification, and actionable recommendations.

A threat model accepts two kinds of inputs, and you can provide either or both:

- **Scope docs** – Technical design documents, API specifications, or architecture documentation that define what the threat model focuses on. When provided, the agent scopes its analysis to the context described in these documents. You can upload files (DOC, DOCX, JPEG, MD, PDF, PNG, TXT), provide S3 URIs, or connect Confluence pages through an integration.
- **Sources** – Source code from connected repositories (GitHub, GitHub Enterprise Server, GitLab, GitLab Self-Managed, or Bitbucket) or from S3 locations. AWS Security Agent reads source code to understand your existing system. When combined with scope docs, the source code provides context about the current implementation.

In this procedure, you create a threat model by configuring its inputs and permissions, then start a run.

Prerequisites

Before you begin, ensure you have:

- Access to the AWS Security Agent web application
- At least one of the following:
  - A connected repository (GitHub, GitHub Enterprise Server, GitLab, GitLab Self-Managed, or Bitbucket) to use as a source (see the section called "Enable threat modeling")

- An S3 bucket containing source code
- Design documents to upload as scope docs, or a Confluence integration

> **Tip**
>
> If you already have repositories or S3 buckets connected to your Agent Space (for example, through code review or penetration testing setup), you can create a threat model immediately — no additional setup is required.

How AWS Security Agent runs a threat model

The inputs you provide determine how AWS Security Agent runs the threat model. There are three scenarios.

Inputs you provide	How AWS Security Agent runs the threat model
Sources only (source code)	AWS Security Agent analyzes the source code to understand your application's structure, classifies components, extracts data flows, builds a threat model, and identifies threats based on how the system is actually implemented.
Scope docs only (design documents)	AWS Security Agent runs a threat model focused on the design described in your documents. It identifies threats based on the architecture, data flows, and components described in the scope docs — useful for threat modeling a design before any code exists.
Both sources and scope docs	AWS Security Agent scopes the threat model to the context described in the scope docs (for example, a design document for a new feature). It uses the source code to understand your existing system, then threat models the design described in the scope docs — whether that design is already implemented or not.

Access the threat models page

Navigate to the threat modeling section in the web application.

1. Log in to the AWS Security Agent web application.
2. In the left sidebar, choose **Threat models**.
3. You see a list of existing threat models with their title, latest run status, and threat counts by severity.

Create a threat model

Set up a new threat model by configuring its inputs and permissions.

1. On the **Threat models** page, choose **Create threat model**.

Configure threat model details

Provide a title for your threat model.

1. In the **Title** field, enter a descriptive name for your threat model.

> **Tip**
>
> Use a name that identifies the application or service being modeled. For example, "billing-service" or "checkout-api".

Add scope documents

Provide the technical design documents that define what the threat model focuses on, such as architecture documentation, API specifications, or design proposals. When scope docs are provided, the agent scopes its analysis to the context described in these documents. Scope docs are optional if you provide sources.

1. In the **Scope documents** section, add documents using one or more of the following methods:
   - **Upload files** – Upload design documents directly (DOC, DOCX, JPEG, MD, PDF, PNG, or TXT).
   - **S3 URIs** – Enter S3 URIs pointing to documents stored in connected S3 buckets.
   - **Confluence pages** – If you have a Confluence integration connected to your Agent Space, select pages from your Confluence workspace.

> **Tip**
>
> For best results, include architecture diagrams, API specifications, and technical documentation that describe your system's components, data flows, and trust boundaries.

Add sources

Select the source code AWS Security Agent uses to understand your existing system. When combined with scope docs, the source code provides context about the current implementation — the agent uses it to understand what already exists. Sources are optional if you provide scope docs.

1. In the **Sources** section, add source code using one or more of the following methods:

Integrated repositories

Select from repositories connected to your Agent Space (GitHub, GitHub Enterprise Server, GitLab, GitLab Self-Managed, or Bitbucket).

1. Select the checkbox next to each repository you want to include as a source.
2. Use the search field to find specific repositories by name.

> **Note**
>
> Only repositories connected to your Agent Space appear here. To add more repositories, ask your administrator to update the Agent Space configuration.

> **Tip**
>
> When you select a repository, you can specify a branch. By default, the service uses the primary branch. To use a different branch, enter the branch name in the field next to the repository.

S3 sources

Add S3 URIs pointing to source code stored in connected S3 buckets.

1. Enter the S3 URI for each source code location you want to include.

Configure AWS resources

Select the IAM service role and optional CloudWatch log group for this threat model.

1. In the **Service role** dropdown, select the IAM role from your configured service roles.

> **Note**
>
> The service role must have permissions to access your source code in S3 and write to CloudWatch logs. Service roles are configured during Agent Space setup in the AWS Management Console.

2. (Optional) In the **Log group** dropdown, select a CloudWatch log group to store threat model execution logs.

Create the threat model

1. Review your configuration.
2. Choose **Create threat model**.

You are redirected to the threat model detail page where you can start a run.

Run a threat model

After creating a threat model, start a run to begin the analysis.

1. On the threat model detail page, choose **Start run**.
2. AWS Security Agent begins analyzing your sources and scope docs.

You can also start a run from the **Threat models** list page by choosing **Start run** next to the threat model you want to run.

Monitor a threat model run

Track the progress of your threat model as it executes.

Run status

The run page header displays a status indicator:

- **In progress** – The threat model agent is running.
- **Stopping** – A cancel was requested; the agent is finishing its current task before halting.
- **Completed** – The run finished successfully.
- **Failed** – The run encountered an error. An error message is displayed with details. Start a new run after resolving the issue.
- **Stopped** – The run was canceled by the user. Any threats generated before the stop are preserved.

Run page tabs

On the run detail page, navigate between tabs:

- **Overview** – View the run summary (job ID, start time, status, duration) with a severity level pie chart showing the breakdown of threats by severity (High, Medium, Low). Below the summary, view a STRIDE threat categories bar chart showing how many threats fall into each STRIDE category. After the run completes, view the system overview produced by the agent.
- **Configuration** – View the sources, documents, and permissions (service role, CloudWatch log group) that were used for this run.
- **Preflight** – View the status of preflight checks that run before threat analysis begins, including logging infrastructure setup, S3 source access validation (when applicable), and testing environment setup. A progress bar shows how many checks have completed.
- **Logs** – View a filterable list of tasks the agent performed during the run. Select any task to view its detailed log output in a side panel.
- **Threats** – View the threats identified during the run (see the section called “Review threats from a threat model”).

Run history

Each threat model maintains a history of all runs. On the threat model detail page:

- The runs table lists all runs with their start time, status, duration, threats count, and ID.
- Choose any run’s start time link to view its full details.

Tip

Re-run the same threat model after you update your source code or revise your scope docs to see how the system overview and threats change over time.

Edit a threat model

To modify your threat model configuration:

1. On the threat model detail page, choose **Edit**.
2. Update the title, sources, scope docs, or AWS resources as needed.
3. Choose **Save changes**.

Note

Editing a threat model does not affect previously completed runs. Those preserve the configuration snapshot used at the time they ran.

Delete a threat model

To delete a threat model and all its associated runs and threats:

1. On the threat model detail page, open the actions menu and choose **Delete**.
2. Confirm the deletion.

Important

If a run is currently in progress, you must stop it before deleting the threat model.

Next steps

After running a threat model:

- Review the system overview and threats (see the section called “Review threats from a threat model”)
- Re-run the threat model after making changes to verify threats are addressed
- Adjust your sources and scope docs as your application evolves

Review threats from a threat model

After a threat model run completes, review the system overview and threats to understand how your application could be attacked and what to do about it. The system overview is a comprehensive document describing your application’s architecture, trust boundaries, data flows, and security posture. Each threat includes a statement, severity level, STRIDE classification, affected assets, and a recommendation for addressing it.

Prerequisites

Before you begin, ensure you have:

- A completed threat model run
- Access to the AWS Security Agent web application

Step 1: Access the threat model run

Navigate to your completed threat model run.

1. Log in to the AWS Security Agent web application.
2. In the left sidebar, choose **Threat models**.
3. Select the threat model you want to examine.
4. In the runs table, select the completed run by choosing its start time link.

Step 2: Review the system overview

The system overview is a comprehensive description of how AWS Security Agent understands your system. It is a structured document that can include:

- **Purpose** – What the system does and who it serves.
- **Capabilities** – Key functionality the system provides.

- **Design intent** – The design change or feature being threat modeled (when scope docs are provided).
- **Architecture** – How the system is built, including deployment patterns and communication protocols.
- **Components** – A table of system components with their purpose and key interactions.
- **Trust boundaries** – Where security contexts change, including what protections exist at each crossing.
- **Data flows** – Detailed descriptions of how data moves through the system, including protocols, credentials, and protections at each step.
- **Security posture** – Current authentication, encryption, and access control mechanisms.
- **Sensitive assets** – Data and credentials that require protection, with their classification and exposure points.
- **Key assumptions** – Security-relevant assumptions the agent made about the system.

To review the system overview:

1. Select the **Overview** tab.
2. Review the **Run summary** section, which shows the job ID, start time, status, and duration.
3. Review the **Severity level** chart, which shows a breakdown of threats by severity (High, Medium, Low) with counts and percentages.
4. Review the **Threat categories** chart, which shows how many threats fall into each STRIDE category (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege).
5. In the **System overview** section, read the agent's full analysis.

> **Tip**
>
> If the system overview doesn't accurately reflect your system, refine your inputs — add relevant repositories as sources or upload more complete scope docs — and run the threat model again.

Step 3: Review threats

Navigate to the **Threats** tab to view all threats identified during the run.

1. Select the **Threats** tab.
2. Threats display as a list with each card showing the threat title, severity badge, status, and last updated timestamp. You can filter threats by severity, status, or search by title.
3. Select a threat from the list to view its full details in the right panel.

Threat severity

- **High** – Requires prompt attention; exploitation could result in significant security impact, up to full system compromise.
- **Medium** – Should be addressed in a reasonable timeframe; contributes to overall security risk.
- **Low** – Can be addressed as part of regular maintenance; minimal immediate risk.

Threat details

Select a threat to view its details in the right panel. The details are organized into the following sections:

Metadata row:

- **Severity** – The risk level assigned by the agent (High, Medium, or Low).
- **STRIDE categories** – The threat classification: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, or Elevation of Privilege.
- **Created at** – When the threat was identified.
- **Last updated** – When the threat was last modified.

Description section:

- **Statement** – A natural-language description of the threat: what the threat source can do, what the impact is, and what conditions enable it.
- **Source** – The actor or origin of the threat (for example, "authenticated user" or "external attacker").

- **Action** – What the threat source can do (for example, "inject SQL queries into the search parameter").
- **Impact** – The direct consequence of the threat action (for example, "unauthorized access to customer records").

References section:

- **Affected assets** – Specific assets affected by the threat (for example, "customer payment records" or "DynamoDB table").
- **Prerequisites** – Conditions that must be true for the threat to be exploitable.
- **Impacted goals** – Security goals affected: Confidentiality, Integrity, Availability, Authorization, Authentication, or Non-repudiation.
- **Evidence** – Source code file paths that support the threat, linking back to specific files in your repository.
- **Anchor** – A code reference linking the threat to a specific node in your source code.

Recommendation section:

- **Recommendation** – Actionable guidance for addressing the threat.

Step 4: Create a threat manually

You can add threats that the agent did not identify — for example, threats discovered during manual review or from external sources.

1. On the **Threats** tab of a completed run, choose **Create threat**.
2. Fill in the threat details:
   - **Statement** – A natural-language description of the threat.
   - **Severity** – Select High, Medium, or Low.
   - **Source** – The actor or origin of the threat.
   - **Prerequisites** – Conditions required for the threat to be exploitable.
   - **Action** – What the threat source can do.
   - **Impact** – The direct consequence of the threat action.
   - **Affected assets** – Specific assets affected (comma-separated).

- **Impacted security goals** – Select from Confidentiality, Integrity, Availability, Authorization, Authentication, or Non-repudiation.
- **STRIDE categories** – Select applicable categories.
- **Recommendation** – Guidance for addressing the threat.

3. Choose **Create**.

The manually created threat appears in the threats list alongside agent-generated threats.

Step 5: Edit and triage threats

As you review threats, you can edit their details and update their status to track progress.

1. Select a threat from the list.
2. Choose the edit icon in the threat detail panel to open the edit form.
3. You can modify the following fields:
   - **Status** – Track the threat lifecycle:
     - **Open** – The threat is acknowledged and needs attention (default).
     - **Resolved** – You have fixed the issue.
     - **Dismissed** – You reviewed the threat and determined it is not applicable.
   - **Severity** – Adjust the severity level if the agent's assessment doesn't match your context.
   - **Statement** – Refine the threat description.
   - **Source, Prerequisites, Action, Impact** – Update the threat details based on your domain knowledge.
   - **Affected assets** – Add or remove affected assets (comma-separated).
   - **Impacted security goals** – Select the security goals affected (Confidentiality, Integrity, Availability, Authorization, Authentication, Non-repudiation).
4. Choose **Save** to apply your changes.

Step 6: Download a report

After a run completes, you can download a PDF report summarizing the system overview and all identified threats.

1. On the completed run page, choose **Generate report**.

2. The PDF downloads to your computer.

Step 7: Review preflight checks and logs

If you need to investigate how the agent reached its conclusions or debug a partial failure:

1. Select the **Preflight** tab to view the status of preflight checks that run before threat analysis begins. Each check shows its status (complete, in progress, or pending), and a progress bar indicates how many checks have completed. You can expand a completed check to view its detailed log output.
2. Select the **Logs** tab to view a filterable list of tasks the agent performed during the run. Select any task from the list to view its detailed log output in the side panel.

Next steps

After reviewing your threat model results:

- Address high-severity threats first based on the agent's recommendations
- Update threat statuses as you implement fixes
- Run a new threat model to verify your changes address the identified threats
- Adjust your sources and scope docs as your application evolves (see the section called "Create a threat model")

Create a design review

Assess your design documents against organization security requirements by uploading files for AWS Security Agent to review. Design reviews help identify security issues early in the development lifecycle, enabling you to address architectural concerns when they are most cost-effective to resolve.

AWS Security Agent analyzes your design documents against your organization's security requirements, providing detailed security findings to improve security posture before implementation begins.

In this procedure, you'll create a design review by uploading design files for analysis.

Prerequisites

Before you begin, ensure you have:

- Access to the AWS Security Agent web application
- Design documents ready for upload (DOC, DOCX, JPEG, MD, PDF, PNG and TXT)
- Each file must be 2 MB or smaller (0.5 MB for Markdown and text files), with a combined total of 6 MB across all files
- Understanding of which security requirements are enabled for your organization

Step 1: Start creating a design review

Navigate to the design review creation page in the web application.

1. Log in to the AWS Security Agent web application.
2. Navigate to the **Design reviews** section.
3. Choose **Create Design Review**.

> **Tip**
>
> You can view your organization's enabled security requirements by navigating to the **Security requirements** page in the AWS Security Agent console. Select any enabled requirement to view its details. These requirements are used to analyze your design files.

Step 2: Name your design review

Provide a descriptive name that helps identify the purpose and scope of this design review.

1. In the **Design review name** section, locate the **Name** field.
2. Enter a descriptive name for your design review.

> **Note**
>
> The name should clearly identify the project, feature, or component being reviewed. Maximum 80 characters.

Step 3: Upload design files

Upload the design documents you want AWS Security Agent to analyze for security compliance.

1. In the **Files to review** section, review the file requirements:

> **Important**
>
> A maximum of 5 files may be uploaded per design review. Each file must be 2 MB or smaller (0.5 MB for Markdown and text files), with a combined total of 6 MB across all files. Supported formats: DOC, DOCX, JPEG, MD, PDF, PNG and TXT.

2. Upload your files using one of these methods:
   a. **Drag and drop** – Drag files directly into the file dropzone area
   b. **Browse** – Use the **Choose files** button to browse and select files from your computer
3. Verify that all required files are uploaded.

> **Tip**
>
> For best results, include architecture diagrams, design specifications, and technical documentation that describe your system's security-relevant components and data flows.

Step 4: Initiate the design review

After configuring all required information, initiate the security analysis of your design documents.

1. Review all uploaded files and settings to ensure accuracy.
2. Choose **Start design review**.
3. AWS Security Agent analyzes your design documents against enabled security requirements.

> **Note**
>
> The design review process typically completes within minutes, depending on the number and size of files uploaded. You receive security findings based on your organization's security requirements.

Next steps

After starting your design review:

- Monitor the review progress in the web application
- Review security findings
- Share findings with your development team
- Address identified security findings in your design
- Update design documents and resubmit if needed

Review findings from a design review

Review findings help you understand which security requirements are met, which need attention, and what actions to take to improve your design's security posture before implementation begins.

In this procedure, you'll learn how to access, filter, and interpret design review findings to address security findings effectively.

Prerequisites

Before you begin, ensure you have:

- A completed design review
- Access to the AWS Security Agent web application
- Familiarity with your organization's enabled security requirements

Step 1: Access the design review

Navigate to your design review to view the findings and summary information.

1. Log in to the AWS Security Agent web application.
2. Navigate to the **Design reviews** section.
3. Select the design review you want to examine from the list.

Tip

The design review details page displays a summary of review status, completion date, and the number of files reviewed.

Step 2: Review the findings summary

Examine the high-level summary to understand the overall security posture of your design.

1. Locate the **Summary** section.
2. Review the count for each compliance status category: **Compliant**, **Non-compliant**, **Insufficient data**, and **Not applicable**.

Note

The summary provides counts for each status type, helping you quickly assess the number of findings requiring attention. For detailed explanations of each status, see Step 4.

Step 3: Filter and navigate findings

Use the filtering and search capabilities to focus on specific findings or compliance statuses.

1. In the **Review findings** section, locate the filter controls.
2. To filter by status:
   a. Choose the status dropdown menu.
   b. Select a specific compliance status to view only findings with that status.
3. To search for specific security requirements:
   a. Enter keywords in the search field.
   b. Results update automatically as you type.
4. Use the pagination controls to navigate through multiple pages of findings.

Tip

Filter by **Non-compliant** status first to prioritize findings that require immediate attention in your design.

Step 4: Understand compliance statuses

Each finding displays a compliance status that indicates how your design addresses a specific security requirement:

- **Compliant** – Your design meets the security requirement based on the analysis
- **Non-compliant** – Your design violates or inadequately addresses the security requirement
- **Insufficient data** – The uploaded files lack enough information to determine compliance
- **Not applicable** – The security requirement doesn't apply to your system design

Important

Focus on **Non-compliant** and **Insufficient data** statuses, as these require action. Address non-compliant findings by updating your design, and resolve insufficient data findings by uploading additional design documentation.

Step 5: View finding details

Select individual findings to view detailed justification and remediation guidance.

1. In the findings table, select a security requirement name.
2. Review the finding details, which include:
   - The specific security requirement being evaluated
   - A comment explaining why the finding received its compliance status, including specific details about what's missing or non-compliant
   - Recommended remediation guidance to address the finding
   - Links to your organization's internal documentation or standards for the security requirement

Note

The comment explains AWS Security Agent's reasoning with specific details. For insufficient data findings, the comment identifies what information is missing, such as "The design documents don't mention authentication mechanisms" or "No information found about data encryption at rest."

Step 6: Address findings

Take action on findings that require attention to improve your design's security posture.

For **Non-compliant** findings:

1. Review the recommended remediation guidance.
2. Review any linked internal documentation for additional context.
3. Update your design documents to address the security requirement.
4. Document the changes you make for future reference.

For **Insufficient data** findings:

1. Read the comment carefully to understand what specific information is missing.
2. Create or update design documents with the missing details.
3. Prepare the updated files for resubmission.

Next steps

After reviewing your design findings:

- Download the findings report as a CSV file for sharing with your team
- Update design documents to address non-compliant findings
- Create additional documentation for insufficient data findings
- Share findings with your development team for discussion
- Clone this design review to create a new review with the original documents pre-loaded, allowing you to update the name and run the analysis again to verify improvements
- Proceed with implementation for designs that meet compliance requirements

For more information about managing security requirements, see the section called “Manage security requirements”.

For more information about creating design reviews, see the section called “Create a design review”.

Security and reference

Reference material for AWS Security Agent: how to interpret finding severity and CVSS metrics, security guidance, service quotas, and the document history.

Severity ratings and CVSS metrics

AWS Security Agent assigns each finding a severity rating, and penetration test findings also include a CVSS (Common Vulnerability Scoring System) metrics breakdown. Use this reference to interpret those values when you review findings.

Severity levels

Each finding includes a color-coded severity badge that shows its risk level:

- **Critical** (red) – Requires immediate action; exploitation could lead to system compromise
- **High** (red) – Requires prompt attention; exploitation could result in significant security impact
- **Medium** (orange) – Should be addressed in a reasonable timeframe; contributes to overall security risk
- **Low** (yellow) – Can be addressed as part of regular maintenance; minimal immediate risk
- **Informational** (blue) – For informational purposes; minimal to no immediate risk

CVSS metrics

Penetration test findings include a CVSS metrics breakdown in the finding's **Risk Reasoning** section. These metrics explain how AWS Security Agent calculated the severity and help you assess the true risk:

- **Attack Vector** (Network, Adjacent, Local, or Physical) – How remotely the attack can be executed
- **Attack Complexity** (Low or High) – Whether specialized conditions are required to exploit the vulnerability
- **Privileges Required** (None, Low, or High) – The access level an attacker needs
- **User Interaction** (None or Required) – Whether the exploit needs user involvement
- **Scope** – Whether the vulnerability affects components beyond the vulnerable one

- **Confidentiality, Integrity, and Availability impact** (None, Low, or High) – The impact on your system's security

> **Important**
>
> Findings with a Network attack vector, Low attack complexity, and High confidentiality or integrity impact represent the most dangerous vulnerabilities and require immediate attention.

Security in AWS Security Agent

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is 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 Security Agent in the AWS Cloud. This includes the service infrastructure, AI models, and penetration testing agents. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs.
- **Security in the cloud** – Your responsibility includes the following areas:
  - Managing access to AWS Security Agent through IAM policies and permissions
  - Protecting the content you provide to the service, including design documents, code repositories, and application URLs for penetration testing
  - Configuring which repositories and applications are monitored
  - Reviewing and acting on security findings provided by the service
  - Securing your applications based on the remediation guidance provided
  - 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 AWS Security Agent.

Security Considerations for AWS Security Agent and AI assisted penetration testing

AWS Security Agent is a frontier agent that proactively secures your applications throughout the development lifecycle across all your environments. It conducts automated security reviews customized to your requirements, with security teams centrally defining standards that are automatically validated during reviews. Security Agent performs on-demand penetration testing customized to your application, discovering and reporting verified security risks. This approach scales security expertise across your applications to match development velocity while providing comprehensive security coverage. By integrating security from design to deployment, it helps prevent vulnerabilities early and at scale.

Security teams define organizational security requirements once in the AWS Console: approved authorization libraries, logging standards, and data access policies. AWS Security Agent automatically enforces these security requirements throughout development, evaluating architectural documents and code against your standards and providing specific guidance when it detects violations. This delivers consistent security enforcement across teams and scales reviews to match development velocity.

For deployment validation, AWS Security Agent transforms penetration testing from a periodic bottleneck into an on-demand capability. Security teams provide target URLs, authentication details, source code and documentation. AWS Security Agent develops deep application understanding and executes sophisticated attack chains to discover and validate vulnerabilities, enabling teams to test whenever needed.

Key capabilities

AWS Security Agent provides comprehensive security capabilities spanning the entire development lifecycle.

Continuum for design security review

AWS Security Agent (now part of AWS Continuum) provides on-demand security feedback on design documents and assesses compliance with organizational security requirements before code is written. Security teams upload design documents through the web application, where the agent analyzes them against your security requirements and surfaces findings with remediation guidance. This accelerates hours-long manual reviews into focused analysis, enabling teams to address security concerns when remediation is most efficient.

Continuum for code security review

AWS Security Agent (now part of AWS Continuum) analyzes pull requests or uploaded code for organizational security requirements and common security issues like missing input validation and SQL injection risks. The agent provides remediation guidance directly within your code repository platform. Security teams configure which repositories to monitor, scaling evaluation across all codebases while maintaining oversight on critical issues.

Continuum for on-demand penetration testing

AWS Security Agent (now part of AWS Continuum) provides on-demand penetration testing that discovers and reports validated security vulnerabilities through tailored multi-step attack scenarios. AWS Security Agent deploys specialized AI agents that develop application context from provided documentation and credentials, then execute sophisticated attack chains to identify complex vulnerabilities that conventional tools miss. It documents findings with impact analysis, reproducible attack paths, and ready-to-implement code fixes, accelerating penetration testing from weeks to hours and scaling validation across your application portfolio.

FAQs

Security & Control

How does AWS Security Agent authenticate and maintain access to systems?

Penetration testing is the only capability in AWS Security Agent that can authenticate to a user's system at runtime. The AWS Security Agent accepts credentials in the form of static username and password credentials (stored in Secrets Manager), or a credential vendor (as a Lambda Function) as configuration before starting the penetration test. These credentials are used to exercise the normal functionality of the user's system/application through the lifecycle of the penetration test. We encourage users to create new credentials with appropriately scoped permissions for the purposes of penetration testing.

Can users control the scope and depth of testing to prevent unintended system impacts?

AWS Security Agent allows customers to select a specific category of vulnerability to explore in an endpoint. Users can specify out-of-scope URLs to prevent AWS Security Agent from performing penetration testing against those targets. https://docs.aws.amazon.com/securityagent/latest/userguide/perform-penetration-test.html

Can AWS Security Agent itself pose a security risk?

AWS Security Agent is instructed to discover security risks, but to do so using intentionally minimal impacting payloads (like extracting the SQL version instead of dropping a table when a SQL injection attack is discovered). AWS Security Agent is also confined to deterministic guardrails to prevent risky behavior like creating excessive load against the target application. While guardrails are in place, there could still be unintentional or non-obvious business logic interactions, therefore, we always recommend doing penetration testing against a pre-production environment.

What data does AWS Security Agent collect and where is it stored?

AWS Security Agent allows users to upload artifacts to provide context about their application being tested. For more information on data protection, see the section called “Data protection”. AWS Security Agent will automatically select the optimal region within your geography to process your inference requests. This maximizes available compute resources, model availability, and delivers the best customer experience. Your data will remain stored only in the region where the request originated, however, input prompts and output results may be processed outside that region. All data will be transmitted encrypted across Amazon’s secure network. For more information, see Cross Region Inference.

What controls are present to block unauthorized testing against an endpoint?

Endpoints that are specified as target URLs for penetration testing will require DNS validation or HTTP validation as a measure of ownership. AWS Security Agent will ask the customer to add a TXT record to the endpoint’s DNS or expose an HTTP Route returning validation string as proof of ownership. Only after demonstrating proof of ownership will the user be able to proceed with a penetration test.

During a penetration test, the network blocks requests to URLs other than your target URLs and accessible URLs. Credential tests do not apply this restriction, so that they can discover the domains your application reaches while signed in. For instructions on testing credentials, see Test credentials.

Customers are responsible for ensuring they have proper authorization to test all systems that may be affected by their penetration testing activities. All use of AWS Security Agent must comply with the AWS Acceptable Use Policy (https://aws.amazon.com/aup/).

How do users block and report any abuse using AWS Security Agent?

AWS Security Agent continuously monitors requests and attempts to access URLs that are outside of the target URLs. If abuse is detected, such as attempting to use AWS Security Agent to conduct

unauthorized testing on a third party endpoint, any ongoing penetration tests in the account will be terminated. Customers can reach out to AWS Support or their AWS account team for help.

Can AWS Security Agent replace penetration testing workflow?

AWS Security Agent is not a professional penetration testing service, and we encourage users to integrate AWS Security Agent into their security review workflow. AWS Security Agent can provide accessibility to penetration testing on-demand during the development phase of the software lifecycle when engaging with penetration testing professionals would be too early, impractical, or need to be re-evaluated too frequently. Security professionals can review findings from AWS Security Agent to validate them, explain them, or extend upon them for new novel findings (if they exist).

Can users set up role-based access control (RBAC) for different team members?

Yes. AWS Security Agent integrates with AWS IAM Identity Center, allowing admins to manage team members who can access the AWS Security Agent web application which allows users to create, manage and view design reviews and penetration tests.

Testing Capabilities

What vulnerabilities does a web application test scope cover?

For a **Web application** test scope, AWS Security Agent detects vulnerabilities in the OWASP Top 10 for Web Applications. AWS Security Agent provides specific risk types that you can include or exclude in testing, shown in the following list. Findings could arise within these risk categories or from novel findings discovered by following leads from a combination of these risk categories. For more information, see the section called “Choose a test scope”.

- Arbitrary File Upload
  - Arbitrary File Upload confirms that the application should be able to fend off bogus and malicious files in a way to keep the application and the users safe
- Code Injection
  - Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application
- Command Injection
  - Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application

- Cross-Site Scripting (XSS)
  - Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites
- Insecure Direct Object Reference
  - Insecure Direct Object References (IDOR) occur when an application provides direct access to objects based on user-supplied input
- JSON Web Token Vulnerabilities
  - JWTs are a common source of vulnerabilities, both in how they are implemented in applications, and in the underlying libraries
- Local File Inclusion
  - File Inclusion vulnerability allows an attacker to include a file, usually exploiting a “dynamic file inclusion” mechanisms implemented in the target application
- Path Traversal
  - Path Traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the web root folder
- Privilege Escalation
  - Privilege escalation occurs when a user gets access to more resources or functionality than they are normally allowed, and such elevation or changes should have been prevented by the application
- Server-Side Request Forgery (SSRF)
  - Server-Side Request Forgery (SSRF) occurs when the attacker can abuse functionality on the server to read or update internal resources
- Server-Side Template Injection
  - Server Side Template Injection vulnerabilities (SSTI) occur when user input is embedded in a template in an unsafe manner and results in remote code execution on the server
- SQL Injection
  - SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application
- XML External Entity
  - XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser

What vulnerabilities does a Gen AI application test scope cover?

For a **Gen AI application** test scope, AWS Security Agent discovers the application's large language model (LLM) surfaces and tests them for the following risk types from the OWASP Top 10 for LLM Applications. The Gen AI scope tests a fixed set of risk types. Unlike the web application scope, it doesn't support including or excluding them. For more information, see the section called "Choose a test scope".

- Prompt Injection
  - Crafted input makes the model cross a boundary it would normally refuse, such as following attacker content over its own rules or making a decision it should not
- Sensitive Information Disclosure
  - The model reveals data it should withhold, such as another user's data, secrets, or internal details
- Improper Output Handling
  - Unsanitized model output flows into a downstream sink and is interpreted there, such as cross-site scripting or SQL injection
- Excessive Agency
  - The model is driven into an unauthorized tool call or backend action
- System Prompt Leakage
  - The model discloses its hidden system prompt, developer instructions, or configured rules

Does AWS Security Agent report open network ports?

Yes. During a penetration test, AWS Security Agent scans each target host for open network ports. When a host has open TCP ports beyond the standard web ports your application uses, it adds a single informational Exposed Network Ports finding. The finding lists each open port with its detected service and version. AWS Security Agent identifies these ports for your awareness. It does not attempt to exploit the services on them.

Can AWS Security Agent test endpoints on non-standard ports?

Yes. AWS Security Agent tests HTTP and HTTPS endpoints on any port. Include the port in the target URL, for example `https://example.com:8443`. Domain verification applies to the host. You don't need to verify the domain again to test a different port.

What authentication methods does AWS Security Agent support?

AWS Security Agent supports common authentication methods, including OAuth and JWT. For more information, see the documentation.

How does AWS Security Agent handle rate limiting and Denial of Service (DOS) prevention?

AWS Security Agent has guardrails to prevent it from disrupting or taking down endpoints under test, including DOS. It has internal velocity controls to detect and handle unexpected traffic patterns.

Can AWS Security Agent test both REST and GraphQL APIs?

Yes, AWS Security Agent can test API endpoints. We encourage customers to provide API documentation as **Additional Learning Resources** allowing AWS Security Agent to have better context on the shape and functionality of each API being tested.

How can users verify that AWS Security Agent has covered all critical application logic and endpoints?

AWS Security Agent will do a breadth-first exploration of the target application(s) and attempt to exercise it normally before attempting any exploits. This allows it to build a working understanding of the application at runtime and discover critical application logic and endpoints. Given its stochastic nature, AWS Security Agent is not guaranteed to discover and test all critical applications and endpoints for any target application. The AWS Security Agent web application provides visibility into all discovered endpoints and actions taken in the Penetration test logs.

Accuracy & Reliability

How does AWS Security Agent validate findings before reporting?

AWS Security Agent uses deterministic validators to help validate the reported finding. In the risk types where it is not possible to use deterministic validators, AWS Security Agent will independently replay the finding steps to gain confidence in the validity of the finding. AWS Security Agent only reports the high or medium confidence findings and hides the unverified findings by default.

How does AWS Security Agent validate findings that require an outbound request from the target?

Some findings can only be confirmed by inducing the target application to make an outbound request. Server-Side Request Forgery (SSRF) is one example. To validate these findings, AWS

Security Agent directs these requests to an AWS-operated collaborator endpoint rather than to arbitrary destinations. Each request carries a unique per-test identifier so the callback maps to that specific test, and the agent is instructed to confirm the finding against the collaborator and to stop further exploitation once it is validated. The collaborator captures DNS and HTTP callbacks using data the service generates internally; no customer data is used.

Can AWS Security Agent adapt to custom application logic?

AWS Security Agent optionally accepts source code, threat model, design documents, and API documentation as **Additional Learning Resources** to gain user-directed context on the target application used in the lifecycle of a penetration test.

Can users review AWS Security Agent testing methodology before execution?

Currently there is no way to preview the course of action for AWS Security Agent. The AWS Security Agent plan is dynamic in nature based on its exploration of the target application. Customers can monitor AWS Security Agent as it goes through its exploration in real time by observing the penetration test logs. If logs show an invalid or undesirable trajectory, customers can stop ongoing penetration test run.

Integration & Deployment

Does AWS Security Agent integrate with security tools (SIEM, vulnerability management) or CI/CD pipelines?

AWS Security Agent does not integrate with any existing security tools or CI/CD pipelines.

How does AWS Security Agent handle environment-specific configurations?

AWS Security Agent can be configured to run with specific IAM roles, inside VPCs, with customer-specified application relevant credentials, and with Github source repositories as source code reference to the target application.

Can AWS Security Agent run in air-gapped or isolated environments?

AWS Security Agent can be configured to have connectivity to VPCs, including ones that do not have outbound internet access.

Can multiple team members run tests simultaneously?

AWS Security Agent supports 5 concurrent penetration test runs per account, independent of who starts the test. Customers can create a maximum of 100 Agent Spaces and 1,000 penetration test projects.

Operational Impact

What's the performance impact on tested systems?

AWS Security Agent has guardrails to prevent it from disrupting or taking down endpoints under test. This includes velocity controls on number of calls that AWS Security Agent can make to an endpoint. System or the endpoint under test should expect some increase in traffic and potential monitoring alerts being triggered due to the penetration test activity. Our recommendation is to only run AWS Security Agent or any penetration testing activity in pre-production environment.

Can users schedule or throttle AWS Security Agent?

AWS Security Agent does not have public APIs or the ability to schedule the penetration test runs. AWS Security Agent also does not offer a concurrency control on requests to the target endpoint when starting the penetration test run. If AWS Security Agent is causing problems for target endpoints, customers can stop the ongoing penetration test(s).

What's the typical duration for a complete security assessment?

The runtime for each penetration test depends on the breadth of the target application, and the risk types configured to be assessed. Most penetration test runs complete within 16 hours.

AWS managed policies for AWS Security Agents

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: SecurityAgentWebAppAPIPolicy

Grants permissions to interact with the Security Agent web application API. This policy enables users to configure and execute automated security penetration tests, manage test executions, view security findings, and access Security Agent resources. This policy references the legacy Agent Instance resource type and specific legacy IAM actions.

Permissions details

This policy grants permissions to interact with the Security Testing Control service (securityagent:*) for:

- Penetration Test Management: Create, update, delete, and list penetration tests and their execution jobs
- Security Findings: View, describe, and update security findings from completed tests, including related content and metadata.
- Task Management: List and retrieve code review and documentation review tasks
- Resource Discovery: List and view agent instances, artifacts, integrations, and discovered endpoints
- Test Execution: Start and stop penetration test executions with real-time monitoring capabilities
- Code Remediation: Start automated code remediation for security findings

To view the latest version of the JSON policy document, see SecurityAgentWebAppAPIPolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AWSSecurityAgentWebAppPolicy

Grants permissions to interact with the Security Agent web application API. This policy enables users to configure and execute automated security penetration tests, manage test executions, view security findings, and access Security Agent resources.

Permissions details

This policy grants permissions to interact with the Security Testing Control service (securityagent:*) for:

- Penetration Test Management: Create, update, delete, and list penetration tests and their jobs
- Security Findings: View, describe, and update security findings from completed tests, including related content and metadata.

- Task Management: List and retrieve code review and design review tasks
- Resource Discovery: List and view Agent Spaces, artifacts, integrations, and discovered endpoints
- Test Execution: Start and stop penetration test jobs with real-time monitoring capabilities
- Code Remediation: Start automated code remediation for security findings

This policy also grants permissions in other AWS services so that you can run penetration tests on a recurring schedule. Each of these permissions is restricted to resources in your own account.

- `scheduler` – Create the Amazon EventBridge Scheduler schedule group for an Agent Space, and create, view, update, and delete the penetration test schedules in that group
- `iam` – View the EventBridge Scheduler execution role for an Agent Space and pass that role to EventBridge Scheduler so that your scheduled penetration tests can start. You can pass the role only to the `scheduler.amazonaws.com` service

To view the latest version of the JSON policy document, see AWSSecurityAgentWebAppPolicy in the AWS Managed Policy Reference Guide.

AWS Security Agents updates to AWS managed policies

View details about updates to AWS managed policies for AWS Security Agents since this service began tracking these changes.

To receive notifications of all source file changes to this specific documentation page, you can subscribe to the following URL with an RSS reader:

Change	Description	Date
Added permissions to AWSSecurityAgentWebAppPolicy.	Added Amazon EventBridge Scheduler and IAM permissions so that you can run penetration tests on a recurring schedule.	October 6, 2026
Added permissions to AWSSecurityAgentWebAppPolicy.	Added `TargetDomain` and `DesignReviewFeedback`	March 31, 2026

Change	Description	Date
	resource permissions for the new resource types.	
Added a new managed policy AWSSecurityAgentWebAppPolicy.	Added managed policy `AWSSecurityAgentWebAppPolicy` for the new AgentSpace resource type and IAM action name changes.	February 9, 2026
Added permissions to SecurityAgentWebApAPIPolicy.	Added `securityagent:StartCodeRemediation` to allow users to start automated code remediation for security findings.	January 20, 2026
Added permissions to SecurityAgentWebApAPIPolicy.	Added `securityagent:BatchGetSecurityTestContentMetadata` to allow users to view images in the console.	December 5, 2025
AWS Security Agents started tracking changes.	AWS Security Agents started tracking changes for its AWS managed policies.	December 2, 2025

Data protection in AWS Security Agent

The AWS shared responsibility model applies to data protection in AWS Security Agent. 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 information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*. 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 AWS Security Agent 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.

Encryption at rest

AWS Security Agent encrypts all data at rest using AWS-managed encryption keys by default. This includes:

- **Design documents and code** – All design documents, code repositories, and application artifacts you provide for security reviews are encrypted using AES-256 encryption.
- **Security findings** – All security findings, vulnerability reports, and remediation recommendations are encrypted at rest.
- **Configuration data** – Security requirements, custom policies, and service configurations are encrypted.
- **Audit logs** – All service activity logs and audit trails are encrypted.

AWS Security Agent uses AWS Key Management Service (AWS KMS) to manage encryption keys. You can optionally use a customer managed key to encrypt your data, giving you full control over the encryption keys that protect your resources. For more information, see the section called "Customer managed keys".

Encryption in transit

AWS Security Agent encrypts all data in transit using Transport Layer Security (TLS) 1.2 or higher. This applies to:

- **API communications** – All API calls between your applications and AWS Security Agent use HTTPS with TLS encryption.
- **Console access** – The AWS Security Agent console is accessed over HTTPS.
- **Repository connections** – Connections to GitHub and other code repositories use encrypted protocols.
- **Agent communications** – All communications between the AWS Security Agent service and penetration testing agents use encrypted channels.

Key management

AWS Security Agent uses AWS Key Management Service (AWS KMS) to manage encryption keys. By default, data is encrypted using AWS-managed keys. You can optionally specify a customer managed key when creating resources such as Agent Spaces and integrations. For more information, see the section called "Customer managed keys".

Internetwork traffic privacy

AWS Security Agent uses the public internet to communicate with cloud-hosted source control providers (GitHub, GitLab, Azure DevOps, and Bitbucket) and Confluence Cloud.

For self-hosted providers (GitLab Self-Managed, GitHub Enterprise Server, and Bitbucket Data Center), you can configure private connections using Amazon VPC Lattice to keep all traffic within the AWS network. For more information, see the section called "Connect to privately hosted source control".

In the default configuration, AWS Security Agent uses the public internet to reach your app for penetration testing. You can optionally configure penetration tests to use a VPC to access your application. For more information, see the section called "Connect agent to private VPC resources".

Data from CI/CD pipeline integrations

When you run penetration tests from a CI/CD pipeline, the pipeline integration sends AWS Security Agent metadata that identifies the change to test: the base and head commit identifiers of the deployed commit range, and the identifier of the pipeline run that triggered the test. AWS Security Agent uses this metadata only to determine the security-relevant surface to test; it computes the difference between the two commits server-side from the repository you connected.

This metadata is encrypted in transit and at rest in the same way as other penetration test data. To avoid disclosing sensitive information, do not embed secrets, credentials, or personally identifiable information (PII) in commit messages or branch names. The pipeline integration authenticates to AWS with short-lived credentials obtained through OpenID Connect (OIDC) federation and does not require you to store long-lived AWS access keys in your pipeline. For more information, see the section called “Run penetration tests from your CI/CD pipeline”.

Cross-Region data processing

AWS Security Agent uses cross-region inference to optimize available compute resources and model availability. Depending on the Region where the request originates, we might process input prompts and output results in a different Region.

- In US East (N. Virginia) – `us-east-1`, US East (Ohio) – `us-east-2`, US West (Oregon) – `us-west-2`, Canada (Central) – `ca-central-1`, 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 (Paris) – `eu-west-3`, and Europe (Stockholm) – `eu-north-1`, AWS Security Agent uses geographic cross-region inference. For most features, data processing remains within the geographic boundary (such as US, EU, Australia, or Japan) where the request originated. Requests from Canada (Central) are processed in the United States and Canada (Central). For Code Remediation, requests from Australia, Canada, and Japan are processed in the European Union. For feature-specific routing details, see the Cross Region Inference table.
- In Asia Pacific (Mumbai) – `ap-south-1`, Asia Pacific (Seoul) – `ap-northeast-2`, Asia Pacific (Singapore) – `ap-southeast-1`, and South America (São Paulo) – `sa-east-1`, AWS Security Agent uses global cross-region inference. We might process input prompts and output results in any commercial AWS Region.

In all cases, your data remains stored only in the Region where the request originated. All data transmitted during cross-Region operations remains on the AWS network and does not traverse the public internet. We encrypt data in transit between AWS Regions.

Cross-Region inference is always enabled and cannot be opted out of. For details on which Regions process requests for each feature, see the Cross Region Inference section in the section called "Security best practices".

> **Note**
>
> For GitHub Enterprise Cloud with data residency, AWS Security Agent processes your repository content in the Region of your AWS Security Agent instance. If that Region differs from your tenant's GitHub data residency Region, AWS Security Agent processes your content outside that Region. To connect a data residency tenant, see the section called "Connect AWS Security Agent to GitHub Enterprise".

Data deletion

When you delete data from AWS Security Agent:

- The data is marked for deletion and is no longer accessible through the service.
- The data is deleted from all AWS Security Agent systems within 30 days.

To delete your data

1. In the AWS console, navigate to AWS Security Agent.
2. Choose the data you want to delete (security reviews, findings, or custom requirements).
3. Choose **Delete** and confirm the deletion.

Identity and access management for AWS Security Agent

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 AWS Security Agent resources. IAM is an AWS service that you can use with no additional charge.

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in AWS Security Agent.

Service user – If you use the AWS Security Agent service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Security Agent features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator.

If you cannot access a feature in AWS Security Agent, see the section called "Troubleshooting AWS Security Agent identity and access".

Service administrator - If you're in charge of AWS Security Agent resources at your company, you probably have full access to AWS Security Agent. It's your job to determine which AWS Security Agent features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS Security Agent, see the section called "How AWS Security Agent works with IAM".

IAM administrator - If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Security Agent. To view example AWS Security Agent identity-based policies that you can use in IAM, see the section called "AWS Security Agent identity-based policy examples".

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. AWS suggests using an IAM role, and avoiding using the root user directly.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS Account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *Account Management Reference Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions.
  - **Service role** – 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 Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – 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.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an Amazon EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the Amazon EC2 instance. To assign an AWS role to an Amazon 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 Amazon EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to create an IAM role (instead of a user) 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 is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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**.

Every IAM entity (user or role) starts with no permissions. By default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. 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 Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing 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 such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access control lists (ACLs)

Access control lists (ACLs) are a type of policy that controls 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. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. 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 AWS Security Agent works with IAM

Before you use IAM to manage access to AWS Security Agent, learn what IAM features are available to use with AWS Security Agent.

IAM feature	AWS Security Agent support
the section called “Identity-based policies for AWS Security Agent”	Yes
the section called “Resource-based policies within AWS Security Agent”	No
the section called “Policy actions for AWS Security Agent”	Yes
the section called “Policy resources for AWS Security Agent”	Partial
the section called “Policy condition keys for AWS Security Agent”	Yes
the section called “Access control lists (ACLs) in AWS Security Agent”	No
the section called “Attribute-based access control (ABAC) with AWS Security Agent”	No
the section called “Using temporary credentials with AWS Security Agent”	Yes
the section called “Forward access sessions for AWS Security Agent”	Yes

IAM feature	AWS Security Agent support
the section called “Service roles for AWS Security Agent”	No
the section called “Service-linked roles for AWS Security Agent”	Yes

To get a high-level view of how AWS Security Agent and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS Security Agent

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. You can’t specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS Security Agent

To view examples of AWS Security Agent identity-based policies, see the section called “AWS Security Agent identity-based policy examples”.

Resource-based policies within AWS Security Agent

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. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS Accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the IAM User Guide.

Policy actions for AWS Security Agent

Supports 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 an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in AWS Security Agent use the following prefix before the action: `securityagent:`. For example, to grant someone permission to create an environment with the AWS Security Agent `CreateEnvironment` API operation, you include the `securityagent:CreateEnvironment` action in their policy. Policy statements must include either an `Action` or `NotAction` element. AWS Security Agent defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "securityagent:action1",
      "securityagent:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "securityagent:List*"
```

Policy resources for AWS Security Agent

Supports policy resources: Partial

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. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Some AWS Security Agent API actions support multiple resources. For example, multiple environments can be referenced when calling the `ListEnvironments` API action. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "EXAMPLE-RESOURCE-1",
      "EXAMPLE-RESOURCE-2"
```

For example, the AWS Security Agent environment resource has the following ARN:

```
arn:${Partition}:securityagent:${Region}:${Account}:environment/${EnvironmentId}
```

To specify the environments `my-environment-1` and `my-environment-2` in your statement, use the following example ARNs:

```
"Resource": [
         "arn:aws:securityagent:us-east-1:123456789012:environment/my-environment-1",
         "arn:aws:securityagent:us-east-1:123456789012:environment/my-environment-2"
```

To specify all environments that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:securityagent:us-east-1:123456789012:environment/*"
```

Policy condition keys for AWS Security Agent

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 (or `Condition` block) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. 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.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS Security Agent defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Access control lists (ACLs) in AWS Security Agent

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.

Attribute-based access control (ABAC) with AWS Security Agent

Supports ABAC (tags in policies): No

Using temporary credentials with AWS Security Agent

Supports temporary credentials: Yes

Some AWS Services don't work when you sign in using temporary credentials. For additional information, including which AWS Services work with temporary credentials, see AWS Services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Forward access sessions for AWS Security Agent

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS Service, combined with the requesting AWS Service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS Services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS Security Agent

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*.

Service-linked roles for AWS Security Agent

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.

AWS Security Agent identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS Security Agent resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies using the JSON editor in the *IAM User Guide*.

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Security Agent 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 IAM Access Analyzer policy validation in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or root users in your 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 Configuring MFA-protected API access in the *IAM User Guide*.

Using the AWS Security Agent console

To access the AWS Security Agent console, an IAM principal must have a minimum set of permissions. These permissions must allow the principal to list and view details about the AWS Security Agent 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 principals with that policy attached to them.

To ensure that your IAM principals can still use the AWS Security Agent console, create a policy with your own unique name. Attach the policy to the principals. For more information, see Adding permissions to a user in the *IAM User Guide*:

For policy details, see the section called "AWS managed policy: SecurityAgentWebAppAPIPolicy".

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 you're trying to perform.

Troubleshooting AWS Security Agent identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS Security Agent and IAM.

AccessDeniedException

If you receive an `AccessDeniedException` when calling an AWS API operation, then the IAM principal credentials that you're using don't have the required permissions to make that call.

```
An error occurred (AccessDeniedException) when calling the CreateEnvironment operation:
```

```
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
securityagent:CreateEnvironment on resource:
 arn:aws:securityagent:region:111122223333:environment/my-env
```

In the previous example message, the user does not have permissions to call the AWS Security Agent `CreateEnvironment` API operation. To provide AWS Security Agent admin permissions to an IAM principal, see the section called "AWS Security Agent identity-based policy examples".

For more general information about IAM, see Control access to AWS resources using policies in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my AWS Security Agent 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 AWS Security Agent supports these features, see the section called "How AWS Security Agent 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 How IAM roles differ from resource-based policies in the *IAM User Guide*.

Incident response

AWS Security Agent is a proactive security testing service designed to identify and prevent vulnerabilities before they can be exploited. The service focuses on preventative security validation through design reviews, code analysis, and penetration testing rather than reactive incident detection or response.

Incident detection

AWS Security Agent does not provide incident detection capabilities. The service operates as a proactive security testing tool that validates applications for vulnerabilities during development and before deployment. For runtime security monitoring and incident detection, use services such as Amazon GuardDuty, AWS Security Hub, or Amazon CloudWatch.

Incident alerting

AWS Security Agent does not generate real-time alerts for security incidents. The service delivers security findings through the AWS Console after completing design reviews, code analyses, or penetration testing engagements. These findings represent potential vulnerabilities discovered during testing rather than active security incidents.

Incident remediation

AWS Security Agent does not provide automated incident remediation. The service identifies security vulnerabilities and provides remediation guidance, including:

- Detailed descriptions of identified vulnerabilities
- Reproducible exploit paths for validated findings
- Specific code fixes and implementation guidance
- Impact analysis for discovered issues

Development and security teams use this guidance to manually address vulnerabilities before they reach production environments.

Supporting incident response activities

While AWS Security Agent is not designed for incident response, security teams can use the service to support post-incident activities:

Vulnerability validation

After a security incident, use AWS Security Agent to test whether similar vulnerabilities exist in other applications or environments.

Security posture assessment

Conduct penetration testing to validate security improvements implemented as part of incident remediation.

Root cause analysis

Use code security review capabilities to identify how similar vulnerabilities might exist in other codebases.

Compliance validation for AWS Security Agent

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. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.

- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS Security Agent

> **Note**
>
> AWS Security Agent is available in the following Regions: US East (N. Virginia) – `us-east-1`, US East (Ohio) – `us-east-2`, US West (Oregon) – `us-west-2`, Asia Pacific (Mumbai) – `ap-south-1`, Asia Pacific (Seoul) – `ap-northeast-2`, Asia Pacific (Singapore) – `ap-southeast-1`, Asia Pacific (Sydney) – `ap-southeast-2`, Asia Pacific (Tokyo) – `ap-northeast-1`, Canada (Central) – `ca-central-1`, Europe (Frankfurt) – `eu-central-1`, Europe (Ireland) – `eu-west-1`, Europe (London) – `eu-west-2`, Europe (Paris) – `eu-west-3`, Europe (Stockholm) – `eu-north-1`, and South America (São Paulo) – `sa-east-1`. It uses cross-region inference. In Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), and South America (São Paulo), AWS Security Agent uses global cross-region inference, which routes inference requests to any commercial AWS Region. In all other Regions, it uses geographic cross-region inference, which keeps data processing within specific geographic boundaries. AWS Security Agent is a tool used during the development of your application, and should not be deployed as critical or customer-facing infrastructure.

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 with 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.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in AWS Security Agent

As a managed service, AWS Security Agent 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 Security in the *AWS Well-Architected Framework*.

Network isolation

AWS Security Agent is a fully managed service accessed through the AWS Console and AWS Security Agent web application. Access to the service is controlled through AWS Identity and Access Management (IAM) or AWS IAM Identity Center, which can integrate with your identity provider.

AWS Security Agent supports interface VPC endpoints powered by AWS PrivateLink, enabling you to privately access the service APIs from your VPC without traversing the public internet. For more information, see the section called “Interface VPC endpoints”.

AWS Security Agent requires internet access to perform penetration testing on target applications and for control plane operations. The service uses AWS-managed infrastructure for testing and does not provision EC2 instances or other compute resources in your account. If you configure VPC access for penetration testing, Security Agent creates an ENI in your subnet, but this ENI does not have a public IP address. For more information, see the section called “Connect agent to private VPC resources”.

Multi-tenancy and resource isolation

AWS Security Agent is a multi-tenant service. Security reviews, findings, and customer data are isolated to individual AWS accounts and encrypted at rest. AWS applies standard infrastructure isolation controls to ensure that one customer’s security testing activities do not impact another customer’s performance or confidentiality.

CI/CD pipeline integrations

When you run penetration tests from a CI/CD pipeline, the pipeline integration runs on your own CI/CD runner, which is infrastructure that you operate and secure. The integration connects to the AWS Security Agent service endpoint over TLS and authenticates with short-lived credentials obtained through OpenID Connect (OIDC) federation. Securing the runner, restricting which pipelines can obtain the OIDC token, and protecting the temporary credentials on the runner are your responsibility. For more information, see the section called “Run penetration tests from your CI/CD pipeline” and the section called “Security best practices”.

AWS Security Agent and interface VPC endpoints (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and AWS Security Agent. You can access Security Agent 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 Security Agent.

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 Security Agent.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for Security Agent

Before you set up an interface endpoint for Security Agent, review Considerations in the *AWS PrivateLink Guide*.

Security Agent supports making calls to all of its API actions from your VPC, including operations for managing Agent Spaces, penetration tests, and security findings.

You can select IPv4, IPv6, or dualstack when creating an endpoint.

VPC endpoint policies are supported for Security Agent. By default, full access to Security Agent is allowed through the interface endpoint. You can control access by attaching an endpoint policy to the interface endpoint or by associating a security group with the endpoint network interfaces.

All API calls to Security Agent are encrypted using TLS 1.2 or higher, including calls made through VPC endpoints. For more information about data protection, see the section called "Data protection". Security Agent API calls are logged in AWS CloudTrail. For more information, see the section called "Example: AWS Security Agent log file entries".

Create an interface endpoint for Security Agent

You can create an interface endpoint for Security Agent 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*.

Create an interface endpoint for Security Agent using the following service name:

```
com.amazonaws.<region>.securityagent
```

To create the interface endpoint using the AWS CLI, run the following command. Replace the Region, VPC ID, subnet IDs, and security group ID with your own values.

```
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-1a2b3c4d \
    --vpc-endpoint-type Interface \
    --service-name com.amazonaws.<region>.securityagent \
    --subnet-ids subnet-1a2b3c4d subnet-5e6f7a8b \
    --security-group-ids sg-1a2b3c4d \
    --private-dns-enabled
```

Important

Private DNS must be enabled for the interface endpoint. Security Agent requires you to use the default Regional DNS name (for example, `securityagent.us-east-1.api.aws`) to make API requests through the endpoint. Calling the endpoint-specific VPCE URL directly is not supported.

To use private DNS, you must set both the `enableDnsSupport` and `enableDnsHostnames` attributes to `true` for your VPC. For more information, see DNS attributes for your VPC in the *Amazon VPC User Guide*.

Configure security groups for the interface endpoint

When you create an interface endpoint, you can associate security groups with the endpoint network interfaces to control traffic to Security Agent. Create a security group that allows inbound HTTPS traffic (port 443) from the resources in your VPC that need to communicate with Security Agent.

The security group should include the following inbound rule:

- Type: HTTPS
- Protocol: TCP
- Port range: 443

- Source: Your VPC CIDR block (for example, `10.0.0.0/16`) or the security group IDs of the resources that need access to Security Agent

For more information, see Security groups in the *AWS PrivateLink Guide*.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Security Agent through the interface endpoint. To control the access allowed to Security Agent from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for AWS Security Agent actions

The following is an example of an endpoint policy for AWS Security Agent. When attached to an endpoint, this policy grants access to the listed AWS Security Agent actions for all principals on all resources.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": "*",
         "Action": [
            "securityagent:CreatePentest",
            "securityagent:ListPentests",
            "securityagent:BatchGetPentests",
            "securityagent:StartPentestJob",
            "securityagent:StopPentestJob",
            "securityagent:ListFindings",
```

```
            "securityagent:BatchGetFindings"
         ],
         "Resource": "*"
      }
   ]
}
```

Example: VPC endpoint policy that denies all access from a specified AWS account

The following VPC endpoint policy denies AWS account 123456789012 all access to resources using the endpoint. The policy allows all actions from other accounts.

```
{
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": "*",
         "Action": "securityagent:*",
         "Resource": "*"
      },
      {
         "Effect": "Deny",
         "Principal": {
            "AWS": "123456789012"
         },
         "Action": "securityagent:*",
         "Resource": "*"
      }
   ]
}
```

Troubleshooting

Connection times out when calling Security Agent API

- Verify that the VPC endpoint status is `available`:

```
aws ec2 describe-vpc-endpoints --vpc-endpoint-ids vpce-1a2b3c4d \
    --query "VpcEndpoints[].State"
```

- Confirm that the security group associated with the endpoint allows inbound TCP traffic on port 443 from your VPC CIDR or source security group.

- Check that your VPC route tables do not route Security Agent traffic to an internet gateway or NAT gateway instead of the endpoint.

DNS resolution fails for `securityagent.<region>.api.aws`

- Verify that private DNS is enabled on the endpoint:

```
aws ec2 describe-vpc-endpoints --vpc-endpoint-ids vpce-1a2b3c4d \
    --query "VpcEndpoints[].PrivateDnsEnabled"
```

- Confirm that both `enableDnsSupport` and `enableDnsHostnames` are set to `true` on your VPC:

```
aws ec2 describe-vpc-attribute --vpc-id vpc-1a2b3c4d --attribute enableDnsSupport
aws ec2 describe-vpc-attribute --vpc-id vpc-1a2b3c4d --attribute enableDnsHostnames
```

`Access denied` errors when calling Security Agent APIs

- Check the VPC endpoint policy to ensure it allows the actions you are trying to perform.
- Verify that your IAM identity policy grants the required `securityagent:` permissions.
- If using a custom endpoint policy, confirm that both the endpoint policy and the IAM policy allow the request.

Configuration and vulnerability analysis in AWS Security Agent

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your

account. For more information, see Cross-service confused deputy prevention in the AWS Identity and Access Management User Guide.

Security best practices for AWS Security Agent

AWS Security Agent provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Use non-production environments for penetration testing

AWS Security Agent uses a comprehensive suite of penetration testing tools from the Kali Linux distribution. These tools are designed to identify security vulnerabilities and may perform actions that modify application state, data, or system configurations.

Best practice: Conduct penetration tests against non-production environments that mirror your production setup. These test environments should:

- Not contain live customer data or sensitive production information
- Be isolated from production systems
- Have similar configurations and security controls as production
- Not use credentials with access to production systems

Testing in production environments may result in:

- Data modification or deletion
- Service disruptions or performance degradation
- Unintended state changes
- Triggering of security alerts or incident response procedures

Validate AI-generated security findings

AWS Security Agent performs security analysis using AI agents. Due to the non-deterministic nature of AI systems, penetration test runs may produce varying results across different executions.

Best practice: Validate security findings before taking remediation action:

- Review the verification scripts generated by AWS Security Agent for each finding
- Execute verification scripts in your test environment to confirm the vulnerability
- Consider running multiple penetration tests to ensure comprehensive coverage
- Apply professional security judgment to assess finding severity and exploitability

Not all identified issues may represent exploitable vulnerabilities in your specific deployment context.

Review and test generated remediation code

AWS Security Agent can generate code fixes and security improvements for identified vulnerabilities. These AI-generated fixes require verification before deployment.

Best practice: Review all generated code changes:

- Examine proposed fixes for completeness and correctness
- Test fixes thoroughly in non-production environments
- Verify that fixes don't introduce new vulnerabilities or break functionality
- Use AWS Security Agent to retest after applying fixes, or run the provided verification scripts
- Follow your organization's code review and approval processes

Code repository access

AWS Security Agent can provide security guidance on code changes through pull request comments and code review integration. To protect sensitive security information, this functionality operates under specific constraints.

Limitation: Code security guidance is restricted to private repositories only. This ensures that:

- Security findings remain confidential to your organization
- Potential vulnerabilities are not publicly disclosed before remediation
- Generated fix recommendations don't expose exploitation details

AWS Security Agent does not provide code security guidance for public repositories. It will not comment on public repositories or open-source projects where security findings would be publicly visible.

Penetration tests can review and remediate private and public repositories that you configured for the penetration test. If the repository is public, the remediation code will be provided as a downloadable diff file instead of a pull request.

Accessible URLs

Accessible URLs specify additional endpoints that the penetration testing environment can access during testing. These are necessary when your application depends on external services such as third-party authentication providers or CDNs. All network dependencies required for testing must be specified as either target URLs or accessible URLs.

During a penetration test, AWS Security Agent blocks network access to any endpoint that you do not specify. Credential tests do not apply this restriction. A credential test can reach endpoints that you do not specify. AWS Security Agent adds your custom HTTP headers to those requests. For more information about credential tests, see Test credentials.

Security implications: AWS Security Agent is not instructed to perform security testing on accessible URLs. By specifying accessible URLs, you indicate trust in these dependencies. Penetration test data, including credentials, may be transmitted to these accessible URL endpoints during testing.

Secure your CI/CD pipeline integration

When you run penetration tests from a CI/CD pipeline, the pipeline assumes an IAM role in your account through OpenID Connect (OIDC) federation. Misconfiguring this role, or leaking its credentials on the runner, would let an untrusted pipeline start penetration tests billed to your account or act with the role's permissions.

Best practice: Apply least privilege to the integration and protect credentials on the runner:

- Scope the role's trust policy to the specific repositories and branches that should run tests by matching the OIDC token's `sub` claim. Do not use a wildcard for the `sub` claim.
- Grant the role only the actions the pipeline needs, and scope them to your specific Agent Space or penetration test ARN where the action supports resource-level permissions.
- Do not store long-lived AWS access keys in your pipeline; use OIDC federation.
- Keep debug or shell tracing off for steps that handle credentials, so tokens and temporary credentials are not written to job logs.
- Disable source control credential persistence on checkout, and do not expose the OIDC token to untrusted steps such as builds from forks.

- Pin the integration to an immutable version (a commit SHA or an image digest).

For setup details, see the section called “Run penetration tests from your CI/CD pipeline”.

Cross Region Inference

AWS Security Agent automatically selects the optimal Region to process your inference requests. This maximizes available compute resources, model availability, and delivers the best customer experience. Your data remains stored only in the Region where the request originated; however, input prompts and output results might be processed outside that Region. We transmit all data encrypted across the AWS network.

AWS Security Agent uses two types of cross-region inference depending on the Region:

- **Geographic cross-region inference** – Keeps data processing within specific geographic boundaries (such as US, EU, Australia, or Japan) for most features. For Code Remediation, requests from Australia, Canada, and Japan are processed in the European Union. Requests from Canada (Central) – `ca-central-1` are processed in the United States and Canada (Central). Used in US East (N. Virginia) – `us-east-1`, US East (Ohio) – `us-east-2`, US West (Oregon) – `us-west-2`, Canada (Central) – `ca-central-1`, 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 (Paris) – `eu-west-3`, and Europe (Stockholm) – `eu-north-1`.
- **Global cross-region inference** – Routes inference requests to any commercial AWS Region, optimizing available resources and enabling higher model throughput. Used in Asia Pacific (Mumbai) – `ap-south-1`, Asia Pacific (Seoul) – `ap-northeast-2`, Asia Pacific (Singapore) – `ap-southeast-1`, and South America (São Paulo) – `sa-east-1`.

For Regions using global cross-region inference, input prompts and output results might be processed in any commercial AWS Region. All data transmitted during cross-region operations remains on the AWS network and does not traverse the public internet. We encrypt data in transit between AWS Regions.

The following table describes where your inference requests are processed based on the Region where the request originated and the feature used.

Request origin	All features except Code Remediation	Code Remediation
United States — US East (N. Virginia) – `us-east-1` , US East (Ohio) – `us-east-2` , US West (Oregon) – `us-west-2`	United States	United States
Canada — Canada (Central) – `ca-central-1`	United States and Canada (Central)	European Union
European Union — Europe (Ireland) – `eu-west-1` , Europe (Frankfurt) – `eu-central-1` , Europe (Paris) – `eu-west-3` , Europe (Stockholm) – `eu-north-1`	European Union	European Union
United Kingdom — Europe (London) – `eu-west-2`	European Union and Europe (London)	European Union
Australia — Asia Pacific (Sydney) – `ap-southeast-2`	Australia	European Union
Japan — Asia Pacific (Tokyo) – `ap-northeast-1`	Japan	European Union
South Korea — Asia Pacific (Seoul) – `ap-northeast-2`	Any commercial AWS Region	European Union
South America — South America (São Paulo) – `sa-east-1`	Any commercial AWS Region	Any commercial AWS Region
India — Asia Pacific (Mumbai) – `ap-south-1`	Any commercial AWS Region	Any commercial AWS Region

Request origin	All features except Code Remediation	Code Remediation
Southeast Asia — Asia Pacific (Singapore) – `ap-southeast-1`	Any commercial AWS Region	Any commercial AWS Region

Cross-Region inference is always enabled and cannot be opted out of. Cross-Region inference is not impacted by customer policies in Service Control Policies (SCPs) or AWS Control Tower that restrict customer content to specific Regions. For more information about how AWS Security Agent protects your data during cross-Region processing, see Cross-Region data processing.

IAM actions and resources migration

AWS Security Agent is a frontier agent that proactively secures your applications throughout the development lifecycle across all your environments. If you onboarded to AWS Security Agent prior to February 9, 2026, you will be impacted by upcoming changes on March 9, 2026 to your existing Agent Instance resources and AWS Security Agent IAM actions. In preparation for releasing public API/SDK support, the Agent Instance resource is being renamed to Agent Space, and specific IAM actions are being renamed. These changes will affect any Application or Agent Instances IAM roles you have created prior to March 9, 2026. In order to avoid seeing authentication issues after March 9, 2026, you will need to follow the steps under Preparing for Migration.

> **Note**
>
> If you create any new Agent Instances after February 9, 2026, the new Agent Instance will be created as an Agent Space and no migration steps will be required.

Planned Changes

AWS Security Agent is renaming the Agent Instance resource to Agent Space: `arn:aws:securityagent:us-east-1:{{accountId}}:agent-instance/*` renamed to `arn:aws:securityagent:us-east-1:{{accountId}}:agent-space/*`. Additionally, the following IAM actions are being renamed:

- `securityagent:ListAgentInstances` renamed to `securityagent:ListAgentSpaces`

- `securityagent:ListControls` renamed to `securityagent:ListSecurityRequirements`
- `securityagent:BatchGetAgentInstances` renamed to `securityagent:BatchGetAgentSpaces`
- `securityagent:BatchGetSecurityTestContentMetadata` renamed to `securityagent:BatchGetPentestJobContentMetadata`
- `securityagent:BatchGetTasks` renamed to `securityagent:BatchGetPentestJobTasks`
- `securityagent:CreateDocumentReview` renamed to `securityagent:CreateDesignReview`
- `securityagent:GetDocumentReview` renamed to `securityagent:GetDesignReview`
- `securityagent:GetDocumentReviewArtifact` renamed to `securityagent:GetDesignReviewArtifact`
- `securityagent:ListDocumentReviews` renamed to `securityagent:ListDesignReviews`
- `securityagent:ListDocumentReviewComments` renamed to `securityagent:ListDesignReviewComments`
- `securityagent:ListTasks` renamed to `securityagent:ListPentestJobTasks`
- `securityagent:StartPentestExecution` renamed to `securityagent:StartPentestJob`
- `securityagent:StopPentestExecution` renamed to `securityagent:StopPentestJob`
- `securityagent:DeleteDocumentReview` renamed to `securityagent:DeleteDesignReview`

Preparing for Migration

In order to avoid seeing issues after March 9, 2026 while continuing to use AWS Security Agent prior to March 9, 2026, you will need to **trust both the new and old resources and IAM actions in your IAM roles/policies** until March 9, 2026. The below instructions will provide a guide for migrating to the new resource and action formats:

1. Log in to your AWS account and navigate to the **AWS Security Agent console**
2. In the left hand panel, select **Settings** and choose the role under **Service role**

3. In the IAM console for the associated role, select **Add permissions** and **Attach policies**
4. Select **AWSSecurityAgentWebAppPolicy** and choose **Add permissions**
   a. **Important Note:** Verify that you have selected **AWSSecurityAgentWebAppPolicy** as the new policy and not **SecurityAgentWebAppAPIPolicy**
5. Verify that your IAM role now has both **AWSSecurityAgentWebAppPolicy** and **SecurityAgentWebAppAPIPolicy** under **Permissions policies**
6. In the same IAM role console, select **Trust relationships** then **Edit trust policy**
7. Update your trust policy to the following format, replacing **{{accountId}}** with your AWS account ID

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "{{accountId}}"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:securityagent:us-east-1:{{accountId}}:application/*",
            "arn:aws:securityagent:us-east-1:{{accountId}}:agent-space/*",
            "arn:aws:securityagent:us-east-1:{{accountId}}:agent-instance/*"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": "sts:SetContext",
      "Condition": {
        "StringEquals": {
```

```
          "aws:SourceAccount": "{{accountId}}"
        },
        "ForAllValues:ArnEquals": {
          "sts:RequestContextProviders": "arn:aws:iam::aws:contextProvider/
IdentityCenter"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:securityagent:us-east-1:{{accountId}}:application/*",
            "arn:aws:securityagent:us-east-1:{{accountId}}:agent-space/*",
            "arn:aws:securityagent:us-east-1:{{accountId}}:agent-instance/*"
          ]
        }
      }
    }
  ]
}
```

8. Navigate back to the **AWS Security Agent console**. From the left-hand panel, select **Agent Spaces**
9. For each Agent Space you have with penetration testing enabled, perform the following steps
   a. Navigate to the Agent Space and select **Penetration test**
   b. Under **Service access**, choose the role under **Service role name**
   c. In the IAM console for the associated role, select **Trust relationships** then **Edit trust policy**
   d. Update your trust policy to the following format, replacing **{{accountId}}** with your AWS account ID

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "securityagent.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "{{accountId}}"
        },
```

```
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:securityagent:us-east-1:{{accountId}}:agent-space/*",
            "arn:aws:securityagent:us-east-1:{{accountId}}:agent-instance/*"
          ]
        }
      }
    }
  ]
}
```

Logging and monitoring in AWS Security Agent

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Security Agent and your other AWS solutions. AWS provides the following monitoring tools to watch AWS Security Agent, report when something is wrong, and take automatic actions when appropriate:

- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- **Amazon CloudWatch** monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For more information, see the Amazon CloudWatch User Guide.

Logging AWS Security Agent API calls with AWS CloudTrail

AWS Security Agent is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Security Agent. CloudTrail captures all API calls for AWS Security Agent as events. The calls captured include calls from the AWS Security Agent console and code calls to the AWS Security Agent API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Security Agent. 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 AWS Security Agent, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS Security Agent information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Security Agent, 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 AWS Security Agent, 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:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All AWS Security Agent actions are logged by CloudTrail and are documented in the AWS Security Agent API Reference. For example, calls to the `CreatePentest`, `StartPentestExecution`, and `ListFindings` 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 or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding AWS Security Agent 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 and includes 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.

The following example shows a CloudTrail log entry that demonstrates the `CreatePentest` action:


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
    },
    "eventTime": "2025-01-15T10:30:00Z",
    "eventSource": "securityagent.amazonaws.com",
    "eventName": "CreatePentest",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "aws-cli/2.13.0",
    "requestParameters": {
        "pentestName": "WebApp-Security-Test",
        "targetUrl": "https://example.com",
        "testScope": "OWASP-Top-10"
    },
    "responseElements": {
        "pentestId": "pt-1234567890abcdef0",
        "pentestArn": "arn:aws:securityagent:us-east-1:123456789012:pentest/
pt-1234567890abcdef0"
    },
    "requestID": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "eventID": "12345678-1234-1234-1234-123456789012",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
```

```
    "eventCategory": "Management"
}
```

The following example shows a CloudTrail log entry that demonstrates the `StartPentestExecution` action:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAEXAMPLEID:session-name",
        "arn": "arn:aws:sts::123456789012:assumed-role/SecurityTeamRole/session-name",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAEXAMPLEID",
                "arn": "arn:aws:iam::123456789012:role/SecurityTeamRole",
                "accountId": "123456789012",
                "userName": "SecurityTeamRole"
            },
            "attributes": {
                "creationDate": "2025-01-15T09:00:00Z",
                "mfaAuthenticated": "true"
            }
        }
    },
    "eventTime": "2025-01-15T11:00:00Z",
    "eventSource": "securityagent.amazonaws.com",
    "eventName": "StartPentestExecution",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.45",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "pentestId": "pt-1234567890abcdef0"
    },
    "responseElements": {
        "executionId": "exec-abcdef1234567890",
        "status": "RUNNING"
    },
    "requestID": "b2c3d4e5-f6a7-8901-bcde-f12345678901",
    "eventID": "23456789-2345-2345-2345-234567890123",
```

```
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

Monitoring AWS Security Agent with Amazon CloudWatch

You can monitor AWS Security Agent using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your security testing is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met.

AWS Security Agent metrics

AWS Security Agent publishes metrics to the `AWS/SecurityAgent` namespace. It sends the following metrics to CloudWatch, grouped by capability:

Capability	Metric	Description
Design review	`DesignReviewsCompleted`	The number of design reviews completed
Code review	`CodeReviewsCompleted`	The number of code reviews completed
Code review	`SucceededCodeReviewJobs`	The number of code review jobs that completed successfully
Code review	`FailedCodeReviewJobs`	The number of code review jobs that failed
Code review	`CodeReviewJobDuration`	The duration of a code review job, in seconds
Code review	`ConcurrentCodeReviewJobs`	The number of code review jobs running concurrently

Capability	Metric	Description
Penetration testing	`SucceededPentestJobs`	The number of penetration test jobs that completed successfully
Penetration testing	`FailedPentestJobs`	The number of penetration test jobs that failed
Penetration testing	`PentestJobDuration`	The duration of a penetration test job, in seconds
Penetration testing	`ConcurrentPentestJobs`	The number of penetration test jobs running concurrently
Threat modeling	`SucceededThreatModelJobs`	The number of threat model jobs that completed successfully
Threat modeling	`FailedThreatModelJobs`	The number of threat model jobs that failed
Threat modeling	`ThreatModelJobDuration`	The duration of a threat model job, in seconds
Threat modeling	`ConcurrentThreatModelJobs`	The number of threat model jobs running concurrently

Dimensions for AWS Security Agent metrics

AWS Security Agent metrics use the following dimension:

Dimension	Description
`AgentSpaceId`	Filters metrics by the Agent Space that produced them

Service Quotas usage metrics

For metrics that back a service quota, AWS Security Agent also publishes usage data to the `AWS/Usage` namespace under the metric name `ResourceCount`. You can use these metrics to monitor your usage against your quotas and to create alarms as you approach a limit. Each datapoint carries the following dimensions:

Dimension	Value
`Service`	`SecurityAgent`
`Type`	`Resource`
`Class`	`None`
`Resource`	One of `CodeReview` , `DesignReview` , `ConcurrentPentestJob` , `ConcurrentCodeReviewJob` , or `ConcurrentThreatModelJob`

Creating CloudWatch alarms for AWS Security Agent

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods.

For example, you can create an alarm that monitors the number of failed penetration test jobs and sends a notification when the number exceeds a threshold:

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. Choose the **AWS/SecurityAgent** namespace, then choose the dimension grouping you want to monitor.
6. Select the metric you want to monitor, then choose **Select metric**.
7. Configure the alarm conditions, notification, and other settings.

8. Choose **Create alarm**.

Logging AWS Security Agent API calls with AWS CloudTrail

AWS Security Agent is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Security Agent. CloudTrail captures all API calls for AWS Security Agent as events. The calls captured include calls from the AWS Security Agent console and code calls to the AWS Security Agent API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Security Agent. 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 AWS Security Agent, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS Security Agent information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Security Agent, 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 AWS Security Agent, 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:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All AWS Security Agent actions are logged by CloudTrail. For example, calls to the `CreatePentest`, `BatchGetPentestJobs`, and `ListFindings` 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 or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Example: AWS Security Agent log file entries

Understanding AWS Security Agent 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 and includes 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.

The following example shows a CloudTrail log entry that demonstrates the `CreatePentest` action:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
    },
    "eventTime": "2025-01-15T10:30:00Z",
    "eventSource": "securityagent.amazonaws.com",
```

```
    "eventName": "CreatePentest",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "aws-cli/2.13.0",
    "requestParameters": {
        "pentestName": "WebApp-Security-Test",
        "targetUrl": "https://example.com",
        "testScope": "OWASP-Top-10"
    },
    "responseElements": {
        "pentestId": "pt-1234567890abcdef0",
        "pentestArn": "arn:aws:securityagent:us-east-1:123456789012:pentest/
pt-1234567890abcdef0"
    },
    "requestID": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "eventID": "12345678-1234-1234-1234-123456789012",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


Service Quotas

AWS Security Agent has quotas that limit the number of resources you can create and the rate at which you can perform operations. Quotas marked as adjustable can be increased by submitting a request through AWS Support. For more information, see Creating support cases and case management.

Quotas limit capacity but don't limit spending. The monthly quotas on this page cap how many reviews and penetration tests you can run. Reaching a quota blocks additional reviews or penetration tests until the next month. They don't cap what you spend, and a quota increase doesn't change your rate. For more information, see the section called "Pricing and billing".

Operations Quotas

Operations quotas limit the monthly usage of security testing and review features to help manage service capacity.

Resource	Scope	Quota	Adjustable
Design reviews	Per month per account per region	200	Yes
PR code reviews	Per month per account per region	1,000	Yes
Continuous pentest runs	Per month per account per region	500	Yes

Configuration Quotas

Configuration quotas limit the number of resources and settings you can configure in your AWS Security Agent environment.

Resource	Scope	Quota	Adjustable
Agent Spaces	Per account per region	100	Yes
Integrations	Per account per region	20	No
Integrated resources per integration	Per integration	50	No
Azure DevOps projects with code review comments or automated remediation enabled	Per Azure DevOps integration	50	No
Security requirement packs	Per account per region	20	No
Security requirements per pack	Per pack	30	No
Enabled security requirements	Per account per region	150	No
Pentest projects	Per account per region	1,000	Yes
Concurrent pentest runs	Per account per region	5	Yes
Code review projects	Per account per region	1,000	Yes

Resource	Scope	Quota	Adjustable
Concurrent code review runs	Per account per region	5	Yes

Document history

The following table describes some of the major updates and new features for the AWS Security Agents User Guide.

Change	Description	Date
Retest of previous findings in a penetration test run	When you run a penetrati on test again, AWS Security Agent now retests the findings from the most recent completed full run before it tests for new findings. For more information, see Retest of previous findings.	November 5, 2026
Assign IAM Identity Center groups to an Agent Space	You can now assign IAM Identity Center groups to an Agent Space, in addition to individual users. All members of an assigned group can sign in to the web applicati on and use the Agent Space. For more information about assigning groups, see Assign groups through the AWS Security Agent console.	November 3, 2026
On-demand pull request code reviews	To review a pull request at any time, including one that doesn't match your trigger filters, comment `@AWS-Cont`	October 13, 2026

	`inuum-Pentesting review` on it.	
Connect AWS Security Agent to Jira	You can now connect AWS Security Agent to Jira Cloud and create a Jira issue for a security finding. After you connect a Jira project to an Agent Space, you can create issues for findings from a penetration test or code review, open a linked issue in Jira, and unlink an issue from a finding. AWS Security Agent can also comment on a linked issue with revalidation results and link a remediati on pull request to it. For more information, see Connect AWS Security Agent to Jira.	October 13, 2026
Choose a test scope for a penetration test	You can now choose a test scope for a penetration test. Choose **Gen AI applicati on** to test a generative AI application for large language model (LLM) risks, such as prompt injection, instead of the vulnerability classes of a traditional website. For more information, see Choose a test scope.	October 9, 2026

Configurable pull request code review triggers	You can now limit code reviews by target branch or label, review draft pull requests, and start a review when a label is added in GitHub or GitLab.	October 1, 2026
Tutorials: Gate a CI/CD deployment with a scoped penetration test	Added step-by-step tutorials for running a penetration test as a post-deployment gate in GitHub Actions, GitLab CI/CD, Bitbucket Pipelines, and Azure DevOps Pipelines . Each tutorial deploys an intentionally vulnerable sample application, gates promotion on the finding, and then shows the gate passing after you fix the vulnerability. For more information, see Run penetration tests from your CI/CD pipeline.	September 28, 2026

View email MFA messages for a penetration test credential	You can now view the messages that arrive at a credential's email MFA address, in the console or with the `list-actor-messages` API. This means you can use the address directly as the email address for a test user in your application. You can open the verification message yourself, instead of forwarding messages from a mailbox of your own. For more information, see View email MFA messages.	September 25, 2026

Run penetration tests from your CI/CD pipeline	Added documentation for running penetration tests from a CI/CD pipeline. You can add a post-deployment gate for GitHub Actions, GitLab CI/CD, Bitbucket Pipelines, or Azure DevOps Pipelines that tests the deployed change and blocks promotion on findings above a severity threshold. The topic covers how the test is scoped to the deployed change, OpenID Connect (OIDC) authentication, least-privilege IAM trust and permissions policies, dry-run validation, gating and scope conflicts , cleanup, and protecting credentials on the runner. For more information, see Run penetration tests from your CI/CD pipeline.	September 25, 2026
Test penetration test credentials before a run	Added documentation for the updated penetration test setup workflow. You can now test each credential before a run to confirm that AWS Security Agent can sign in. This testing also discovers the domains your application reaches while signed in. Use the new **Network configura tion** step to classify every domain the test may reach.	September 18, 2026

Penetration testing for non-standard ports	AWS Security Agent now tests target endpoints that serve traffic on non-standard ports. Include the port in the target URL, for example `https://example.com:8443` . Accessible URLs also accept a port.	September 16, 2026
Exposed network ports findings	During a penetration test, AWS Security Agent now reports open network ports on your target hosts as a new informational finding. When a host listens on TCP ports beyond the standard web ports your application uses, AWS Security Agent adds a single Exposed Network Ports finding. The finding lists each port with its detected service and version. AWS Security Agent identifies these ports for your awareness. It does not attempt to exploit the services on them. For more information, see Does AWS Security Agent report open network ports?.	August 31, 2026

Code remediation for Amazon S3 sources	AWS Security Agent now supports code remediation for findings from Amazon S3 sources, for both penetrati on tests and code reviews, and both automatically and on demand. Because there's no connected repository to open a pull request against, AWS Security Agent attaches a downloadable code diff to the finding that you can apply locally with `git apply`.	August 18, 2026
AWS Security Agent now supports 20 target domains per Agent Space	You can now configure up to 20 target domains per Agent Space for penetration testing. This limit was previously 5.	August 18, 2026
Revalidate penetration test findings	Added documentation for revalidating penetrati on test findings. You can select existing findings and re-test whether they are still exploitable, without running a full penetration test. Revalidation runs as a separate job and reports whether each finding is still active or resolved, without changing the original finding.	August 13, 2026

Maximum task hours for penetration tests and code reviews	Added documentation for setting a maximum task-hours limit when you create a penetration test or code review. When a run reaches the limit, AWS Security Agent stops it gracefully and keeps the findings discovered so far.	August 13, 2026
Pricing and billing	Added documentation for pricing and billing. This topic explains what accrues task hours, the unit AWS Security Agent bills for penetration testing. It covers why task hours differ from the duration of a run and where to find task hours. It also explains why service quotas are not spending limits. For more information, see Pricing and billing.	August 11, 2026

Email MFA for penetration testing credentials	Added documentation for email MFA. You can now enable email-based multi-factor authentication (MFA) for penetration testing credentials. When enabled, AWS Security Agent can complete logins for applications that send a one-time code or verification link by email. AWS Security Agent generates a unique forwarding address that you configure in your email provider.	August 6, 2026
Private connections general availability	Private connections are now generally available. This capability allows AWS Security Agent to connect to source control systems running in private networks using Amazon VPC Lattice, without exposing your systems to the public internet.	July 28, 2026
Private connections (preview)	Added documentation for private connections. This new capability allows AWS Security Agent to connect to source control systems running in private networks using Amazon VPC Lattice, without exposing your systems to the public internet.	June 16, 2026

Threat modeling (preview)	Added documentation for threat modeling. This new capability builds a threat model of your application from design documents (scope docs), source code (sources), or both. Scope docs are feature design documents that define the focus of the analysis, while source code provides context about your existing system. If you don't provide scope docs, the agent generates a threat model from the source code alone. Each run produces a system overview and a set of threats classified by STRIDE category with severity ratings and recommendations.	June 8, 2026
Full repository code review (preview)	Added documentation for full repository code review. This new capability performs context-aware security analysis of your entire codebase and generates code remediation for findings.	May 12, 2026
Updated Region availability for finding remediation	Region availability for penetration test finding remediation in the AWS Security Agent web applicati on has been updated.	April 16, 2026

Updated Region availability for enabling finding remediation	Region availability for enabling penetration test finding remediation in the AWS Management Console has been updated.	April 16, 2026
Customer managed key support	Added documentation for customer managed key (CMK) support. You can now specify a customer managed KMS key when creating Agent Spaces and integrations to encrypt your data with keys you control.	March 31, 2026
AWS managed policy updates	Added AWSSecurityAgentWebAppPolicy managed policy for the new TargetDomain and DesignReviewFeedback resource types.	March 31, 2026
AWS managed policy updates	Added AWSSecurityAgentWebAppPolicy managed policy for the new AgentSpace resource type and IAM action name changes.	February 9, 2026
AWS managed policy updates	Updated SecurityAgentWebAppAPIPolicy to allow customers to delete design reviews.	January 28, 2026
AWS managed policy updates	Updated SecurityAgentWebAppAPIPolicy to allow customers to start automated code remediation for security findings.	January 20, 2026

AWS managed policy updates	Updated to SecurityAgentWebAppAPIPolicy to allow customers to view images in the console.	December 5, 2025
AWS Security Agents initial release	Initial documentation for service launch	December 2, 2025