aws

AWS IoT Device Defender Developer Guide

AWS IoT Device Defender



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

AWS IoT Device Defender: AWS IoT Device Defender Developer Guide

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

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

Table of Contents

What is AWS IoT Device Defender? 1
Are you a first-time AWS IoT Device Defender user? 2
How AWS IoT Device Defender works 2
Features of AWS IoT Device Defender 3
How to get started with AWS IoT Device Defender 5
Related services 5
Accessing AWS IoT Device Defender 5
Pricing for AWS IoT Device Defender 6
Getting started with AWS IoT Device Defender 7
Setting up 7
Sign up for an AWS account 7
Audit guide 7
Prerequisites 8
Enable audit checks 8
View audit results 8
Creating audit mitigation actions 9
Apply mitigation actions to your audit findings 9
Creating an AWS IoT Device Defender Audit IAM role (optional) 10
Enable SNS notifications (optional) 11
Configure permissions for customer managed keys (optional) 12
Enable logging (optional) 13
ML Detect guide 13
Prerequisites 14
How to use ML Detect in the console 14
How to use ML Detect with the CLI 32
Customize when and how you view AWS IoT Device Defender audit results 46
Getting started 47
Customize your audit findings in the console 47
Customize your audit findings in the CLI 50
Audit 58
Issue severity 58
Next steps 59
Audit checks 59
Intermediate CA revoked for active device certificates check 60

Revoked CA certificate still active ... 61
Device certificate shared ... 62
Device certificate key quality ... 63
CA certificate key quality ... 65
Unauthenticated Cognito role overly permissive ... 67
Authenticated Cognito role overly permissive ... 75
AWS IoT policies overly permissive ... 85
AWS IoT policy potentially misconfigured ... 90
Role alias overly permissive ... 96
Role alias allows access to unused services ... 97
CA certificate expiring ... 98
Conflicting MQTT client IDs ... 99
Device certificate expiring ... 100
Device certificate age check ... 102
Revoked device certificate still active ... 103
Logging disabled ... 104
Audit commands ... 105
Manage audit settings ... 105
Schedule audits ... 111
Run an On-Demand audit ... 125
Manage audit instances ... 126
Check audit results ... 136
Audit finding suppressions ... 145
How audit finding suppressions work ... 145
How to use audit finding suppressions in the console ... 146
How to use audit finding suppressions in the CLI ... 153
Audit finding suppressions APIs ... 155
Detect ... 156
Monitoring the behavior of unregistered devices ... 157
Detect availability change ... 158
AWS IoT Device Defender detect alternatives ... 159
Frequently asked questions ... 160
Security use cases ... 161
Cloud-side use cases ... 161
Device-side use cases ... 164
Concepts ... 168

Behaviors ... 171
ML Detect ... 174
Use cases of ML Detect ... 175
How ML Detect works ... 175
Minimum requirements ... 175
Limitations ... 176
Marking false positives and other verification states in alarms ... 177
Supported metrics ... 177
Service quotas ... 178
ML Detect CLI commands ... 178
ML Detect APIs ... 178
Pause or delete an ML Detect Security Profile ... 179
Custom metrics ... 180
How to use custom metrics in the console ... 181
How to use custom metrics from the CLI ... 184
Custom metrics CLI commands ... 188
Custom metrics APIs ... 188
Device-side metrics ... 188
Bytes out (aws:all-bytes-out) ... 189
Bytes in (aws:all-bytes-in) ... 190
Listening TCP port count (aws:num-listening-tcp-ports) ... 192
Listening UDP port count (aws:num-listening-udp-ports) ... 193
Packets out (aws:all-packets-out) ... 195
Packets in (aws:all-packets-in) ... 196
Destination IPs (aws:destination-ip-addresses) ... 198
Listening TCP ports (aws:listening-tcp-ports) ... 198
Listening UDP ports (aws:listening-udp-ports) ... 199
Established TCP connections count (aws:num-established-tcp-connections) ... 200
Device metrics document specification ... 201
Sending metrics from devices ... 210
Cloud-side metrics ... 211
Message size (aws:message-byte-size) ... 211
Messages sent (aws:num-messages-sent) ... 213
Messages received (aws:num-messages-received) ... 214
Authorization failures (aws:num-authorization-failures) ... 216
Source IP (aws:source-ip-address) ... 218

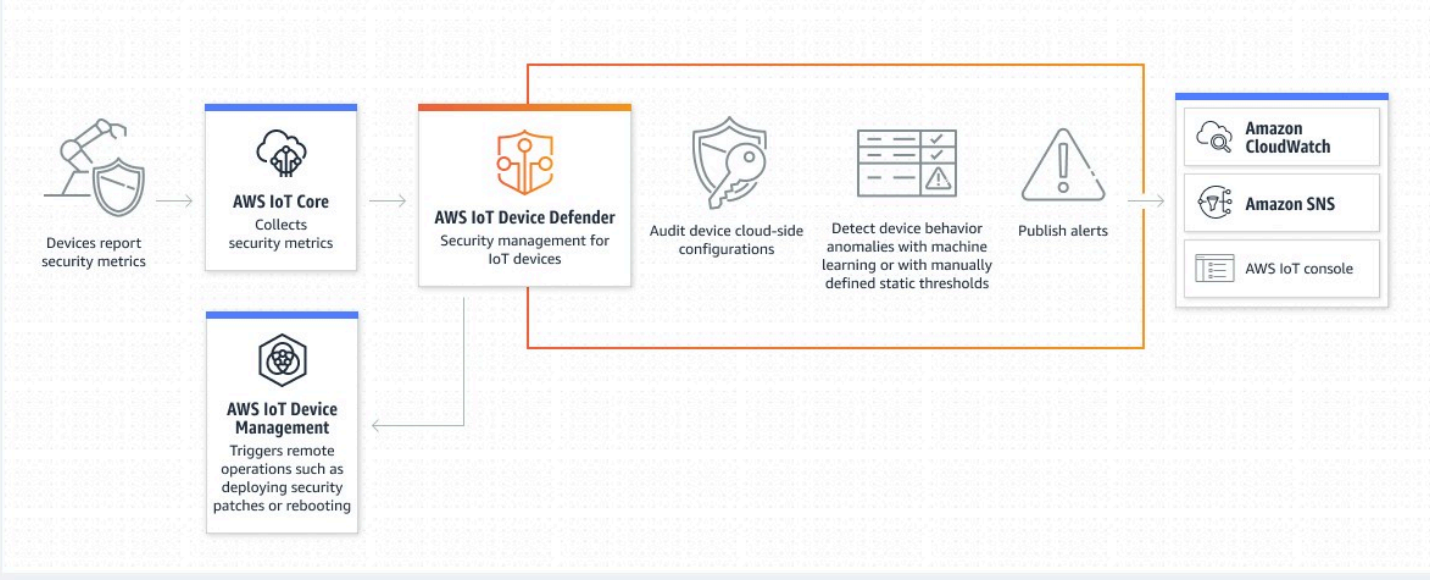
Connection attempts (aws:num-connection-attempts) ... 218
Disconnects (aws:num-disconnects) ... 220
Disconnect duration (aws:disconnect-duration) ... 221
Detect metrics export ... 222
How detect metric export works ... 224
Metrics export schema ... 224
Detect metrics export pricing ... 226
Permissions ... 226
Setting up Detect metrics export in the AWS IoT console ... 228
Creating a security profile to enable metrics export ... 229
Updating a security profile to enable metrics export (CLI) ... 230
Updating a security profile to turn off metrics export (CLI) ... 232
Metrics export CLI commands ... 233
Metrics export API operations ... 233
Scoping metrics in security profiles using dimensions ... 234
How to use dimensions in the console ... 235
How to use dimensions on the AWS CLI ... 236
Permissions ... 241
Give AWS IoT Device Defender detect permission to publish alarms to an SNS topic ... 241
Detect commands ... 243
How to use AWS IoT Device Defender detect ... 245
Mitigation actions ... 248
Audit mitigation actions ... 248
Detect mitigation actions ... 252
How to define and manage mitigation actions ... 253
Create mitigation actions ... 253
Apply mitigation actions ... 255
Permissions ... 260
Mitigation action commands ... 267
Using AWS IoT Device Defender with other AWS services ... 269
Using AWS IoT Device Defender with devices running AWS IoT Greengrass ... 269
Using AWS IoT Device Defender with FreeRTOS and embedded devices ... 269
Using AWS IoT Device Defender with AWS IoT Device Management ... 270
Security Hub CSPM integration ... 270
Enabling and configuring the integration ... 271
How AWS IoT Device Defender sends findings to Security Hub CSPM ... 271

Typical finding from AWS IoT Device Defender 273
Stopping AWS IoT Device Defender from sending findings to Security Hub CSPM 279
Cross-service confused deputy prevention 279
Security best practices for device agents 280
AWS IoT Device Defender troubleshooting guide 284
Security 290
Data protection 290
Identity and access management 291
Audience 292
Authenticating with identities 292
Managing access using policies 293
How AWS IoT Device Defender works with IAM 295
Identity-based policy examples 300
Troubleshooting 303
Compliance validation 305
Resilience 305
Document history 307

What is AWS IoT Device Defender?

Use AWS IoT Device Defender, a security and monitoring service, to audit the configuration of your devices, monitor connected devices, and mitigate security risks. With AWS IoT Device Defender, you can enforce consistent security policies across your AWS IoT device fleet and respond quickly when devices are compromised. IoT fleets can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make fleet setup complex and error-prone. Because devices are often constrained in computational power, memory, and storage capabilities, this limits the use of encryption and other forms of security on the devices themselves.

Devices often use software with known vulnerabilities. These factors make IoT fleets an attractive target for hackers and make it difficult to secure your device fleet on an ongoing basis. AWS IoT Device Defender addresses these challenges by providing tools to identify security issues and deviations from best practices. AWS IoT Device Defender can audit device fleets to confirm that they adhere to security best practices and detect abnormal behavior on devices. The following diagram shows the basic architecture of AWS IoT Device Defender and how it relates to services such as AWS IoT Core, Amazon CloudWatch, and Amazon SNS.



Topics

- Are you a first-time AWS IoT Device Defender user?
- How AWS IoT Device Defender works
- Features of AWS IoT Device Defender

- How to get started with AWS IoT Device Defender
- Related services
- Accessing AWS IoT Device Defender
- Pricing for AWS IoT Device Defender

Are you a first-time AWS IoT Device Defender user?

If you're a first-time user of AWS IoT Device Defender, we recommend that you begin by reading the following sections:

- How AWS IoT Device Defender works
- Features of AWS IoT Device Defender
- How to get started with AWS IoT Device Defender
- Related services
- Accessing AWS IoT Device Defender
- Pricing for AWS IoT Device Defender

How AWS IoT Device Defender works

AWS IoT Device Defender is a fully managed security and monitoring service that helps you secure your fleet of IoT devices. AWS IoT Device Defender audits IoT resources associated with your devices to confirm that they comply with security best practices. Audit checks send alerts if there are any detected security risks, and provide relevant information to help mitigate any issues. AWS IoT Device Defender also continuously monitors security metrics from the cloud, and device-side to detect unexpected device behaviors to identify any possible compromised devices. You can launch audit checks on-demand or on a scheduled basis to assess your IoT device configurations.

AWS IoT Device Defender works with AWS IoT Core to incorporate the context of device interactions to increase the accuracy of audit checks. AWS IoT Device Defender collects and analyzes high-value security metrics from your connected devices to detect abnormal behaviors. When you use *Rules Detect*, the metric data is continuously evaluated against user-defined behaviors. When you use *ML Detect*, the metric data is continuously evaluated by automatically built machine learning (ML) models to identify anomalies.

The results from scheduled audit tasks and any detected device activity anomalies are published to the AWS IoT Console and AWS IoT Device Defender API. They are accessible through Amazon CloudWatch. Additionally, you can configure AWS IoT Device Defender to send results toAmazon SNS topics for integration with security dashboards or starting automated remediation workflows.

AWS IoT Device Defender supports a wide range of use cases, including the following:

- **Protect your devices:** You can audit your device-related resources against AWS IoT security best practices to help you detect device vulnerabilities. AWS IoT Device Defender audits can help you identify and uncover risks to your devices, and confirm that security measures are in place.
- **Detect unusual device behavior:** You can pinpoint changes in connection patterns, reveal device communication with unauthorized endpoints, and identify changes in inbound and outbound device traffic patters.
- **Get insight to mitigate risks:** You can take actions to mitigate issues uncovered in an *Audit finding* or *Detect alarm*.
- **Uphold and maintain device security:** You can use insights from Audit and Detect checks to diagnose and remediate possible security breaches.
- **Enhance device security:** You can distinguish an incorrectly configured device, probe the health of your device fleets, and locate unexpected device behavioral metrics.

Features of AWS IoT Device Defender

The following are a few of the key features of AWS IoT Device Defender.

Key Features

Audit	AWS IoT Device Defender audits your device-related resources against AWS IoT security best practices. in the *IAM User Guide* AWS IoT Device Defender reports configurations that are out of compliance with security best practices, such as overly permissive policies that can allow one device to read and update data for many other devices.

Rules Detect	AWS IoT Device Defenderdetects unusual device behavior that can be indicative of a compromise by continuously monitoring high-value security metrics from the device and AWS IoT Core. You can specify normal device behavior for a group of devices by setting up behaviors (rules) for these metrics. AWS IoT Device Defender monitors and evaluates each datapoint reported for these metrics against user-defined behaviors (rules) and alerts you if an anomaly is detected.
ML Detect	AWS IoT Device Defender automatically sets device behaviors for you with machine learning (ML) models using device data across six cloud-side metrics and seven device-side metrics from a trailing 14-day period. It then retrains the models each day (as long as it has sufficient data to train the model) to refresh the expected device behaviors based on the latest trailing 14 days after initial models are built. AWS IoT Device Defender monitors and identifies anomalous datapoints for these metrics with the ML models and sets off an alarm if an anomaly is detected.
Alerting	AWS IoT Device Defender publishes alarms to the AWS IoT Console, Amazon CloudWatch, and Amazon SNS.

Mitigation	AWS IoT Device Defender can be used to investigate issues by providing contextual and historical information about the device such as device metadata, device statistics, and historical alerts for the device. You can also use AWS IoT Device Defender built-in mitigation actions to perform mitigation steps on Audit and Detect alarms such as adding things to a thing group, replacing default policy version, and updating device certificate.

How to get started with AWS IoT Device Defender

For help getting started with AWS IoT Device Defender, see the following tutorials.

- Setting up
- ML Detect guide
- Audit guide
- Customize when and how you view AWS IoT Device Defender audit results

Related services

- **AWS IoT Greengrass**: AWS IoT Greengrass provides pre-built integration with AWS IoT Device Defender to monitor device behaviors on an ongoing basis.
- **AWS IoT Device Management:** You can use AWS IoT Device Management fleet indexing to index, search, and aggregate your AWS IoT Device Defender detect violations.

Accessing AWS IoT Device Defender

You can use the AWS IoT Device Defender console or the API to access AWS IoT Device Defender.

Pricing for AWS IoT Device Defender

With AWS IoT Device Defender, you only pay for what you use. There is no minimum fee or mandatory service usage. However, you are billed separately for Audit and Detect features. Audit pricing is per device count, per month. When you turn on Audit, you're charged based on the number of active device principals in a month. Therefore, adding or removing audit checks would not affect your monthly bill when using this feature. You can calculate your AWS IoT Device Defender and architecture cost in a single estimate using the AWS Pricing Calculator.

- AWS Pricing Calculator

Getting started with AWS IoT Device Defender

You can use the following tutorials to work with AWS IoT Device Defender.

Topics

- Setting up
- Audit guide
- ML Detect guide
- Customize when and how you view AWS IoT Device Defender audit results

Setting up

Before you use AWS IoT Device Defender for the first time, complete the following tasks:

Topics

- Sign up for an AWS account

Sign up for an AWS account

To get started with AWS, you need an AWS account. There are two ways to sign up for AWS. If you want complete control over account configuration and where your content is stored, or if you have regulated workloads such as HIPAA or FedRAMP, use Sign up for AWS (advanced). If you are a new user and want to use a login you already own to sign in to AWS and create an AWS environment with preconfigured defaults in select AWS Regions, use Sign up for AWS.

Audit guide

This tutorial provides instructions on how to configure a recurring audit, setting up alarms, reviewing audit results and mitigating audit issues.

Topics

- Prerequisites
- Enable audit checks
- View audit results

- Creating audit mitigation actions
- Apply mitigation actions to your audit findings
- Creating an AWS IoT Device Defender Audit IAM role (optional)
- Enable SNS notifications (optional)
- Configure permissions for customer managed keys (optional)
- Enable logging (optional)

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have this, see Setting up.

Enable audit checks

In the following procedure, you enable audit checks that look at account and device settings and policies to ensure security measures are in place. In this tutorial we instruct you to enable all audit checks, but you're able to select whichever checks you wish.

Audit pricing is per device count per month (fleet devices connected to AWS IoT). Therefore, adding or removing audit checks would not affect your monthly bill when using this feature.

1. Open the AWS IoT console. In the navigation pane, expand **Security** and choose **Intro**.
2. Choose **Automate AWS IoT security audit**. Audit checks are automatically turned on.
3. Expand **Audit** and choose **Settings** to view your audit checks. Select an audit check name to learn about what the audit check does. For more information about audit checks, see Audit Checks.
4. (Optional) If you already have a role that you want to use, choose **Manage service permissions**, choose the role from the list, and then choose **Update**.

View audit results

The following procedure shows you how to view your audit results. In this tutorial, you see the audit results from the audit checks set up in Enable audit checks tutorial.

To view audit results

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Audit**, and then choose **Results**.
2. Select the **Name** of the audit schedule you'd like to investigate.
3. In **Non-compliant checks**, under **Mitigation**, select the info buttons for information about why it's non-compliant. For guidance on how to make your non-compliant checks compliant, see Audit checks.

Creating audit mitigation actions

In the following procedure, you will create an AWS IoT Device Defender Audit Mitigation Action to enable AWS IoT logging. Each audit check has mapped mitigation actions that will affect which **Action type** you choose for the audit check you want to fix. For more information, see Mitigation actions.

To use the AWS IoT console to create mitigation actions

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Detect**, and then choose **Mitigation actions**.
2. On the **Mitigation actions** page, choose **Create**.
3. On the **Create a new mitigation action** page, for **Action name**, enter a unique name for your mitigation action such as *`EnableErrorLoggingAction`*.
4. For **Action type**, choose **Enable AWS IoT logging**.
5. In **Permissions**, choose **Create role**. For **Role name**, use *`IoTMitigationActionErrorLoggingRole`*. Then, choose **Create**.
6. In **Parameters**, under **Role for logging**, choose `IoTMitigationActionErrorLoggingRole`. For **Log level**, choose `Error`.
7. Choose **Create**.

Apply mitigation actions to your audit findings

The following procedure shows you how to apply mitigation actions to your audit results.

To mitigate non-compliant audit findings

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Audit**, and then choose **Results**.
2. Choose an audit result that you want to respond to.
3. Check your results.
4. Choose **Start mitigation actions**.
5. For **Logging disabled**, choose the mitigation action that you previously created, `EnableErrorLoggingAction`. You can select the appropriate actions for each non-compliant finding to address the issues.
6. For **Select reason codes**, choose the reason code that was returned by the audit check.
7. Choose **Start task**. The mitigation action may take a few minutes to run.

To check that the mitigation action worked

1. In the AWS IoT console, in the navigation pane, choose **Settings**.
2. In **Service log**, confirm that the **Log level** is `Error (least verbosity)`.

Creating an AWS IoT Device Defender Audit IAM role (optional)

In the following procedure, you create an AWS IoT Device Defender Audit IAM role that provides AWS IoT Device Defender read access to AWS IoT.

To create the service role for AWS IoT Device Defender (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type.
4. In **Use cases for other AWS services**, choose **AWS IoT**, and then choose **IoT - Device Defender Audit**.
5. Choose **Next**.
6. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

Expand the **Permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see Creating IAM policies in the *IAM User Guide*. After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.

7. Choose **Next**.
8. Enter a role name to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because various entities might reference the role, you can't edit the name of the role after it has been created.
9. (Optional) For **Description**, enter a description for the new role.
10. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Select permissions** sections to edit the use cases and permissions for the role.
11. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources in the *IAM User Guide*.
12. Review the role and then choose **Create role**.

Enable SNS notifications (optional)

In the following procedure, you enable Amazon SNS (SNS) notifications to alert you when your audits identify any non-compliant resources. In this tutorial you will set up notifications for the audit checks enabled in the Enable audit checks tutorial.

1. If you haven't already, attach a policy that provides access to SNS via the AWS Management Console. You can do this by following the instructions in Attaching a policy to an IAM user group in the *IAM User Guide* and selecting the **AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction** policy.
2. Open the AWS IoT console. In the navigation pane, expand **Security**, **Audit**, and then choose **Settings**.
3. At the bottom of the **Device Defender audit settings** page, choose **Enable SNS alerts**.
4. Choose **Enabled**.

5. For **Topic**, choose **Create new topic**. Name the topic *IoTDDNotifications* and choose **Create**. For **Role**, choose the role that you created in Creating an AWS IoT Device Defender Audit IAM role (optional).
6. Choose **Update**.
7. If you'd like to receive email or text in your Ops platforms through Amazon SNS, see Using Amazon Simple Notification Service for user notifications.

Configure permissions for customer managed keys (optional)

> **Note**
>
> This configuration is only required if you have opted in to customer managed keys for AWS IoT Core. For more information about AWS IoT Core encryption at rest, see Data encryption at rest in AWS IoT Core.

If you have enabled customer managed keys (CMK) for AWS IoT Core encryption at rest, the IAM role used by AWS IoT Device Defender Audit requires additional permissions to decrypt data. Without these permissions, audit operations will fail.

The `AWSIoTDeviceDefenderAudit` managed policy does not include `kms:Decrypt` permissions by design, following the principle of least privilege. You must manually add these permissions to your audit role when using customer managed keys.

To add KMS permissions to your AWS IoT Device Defender Audit IAM role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then search for the role you created in Creating an AWS IoT Device Defender Audit IAM role (optional) or the role you specified when configuring audit settings.
3. Choose the role name to open its details page.
4. In the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
5. Choose the **JSON** tab and enter the following policy. Replace the example *us-east-1* Region, *111122223333* account ID, and *KEY_ID* with your AWS KMS key details:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:111122223333:key/KEY_ID"
    }
  ]
}
```

6. Choose **Next**.
7. For **Policy name**, enter a descriptive name such as **`DeviceDefenderAuditKMSDecrypt`**.
8. Choose **Create policy**.

Enable logging (optional)

This procedure describes how to enable AWS IoT to log information to CloudWatch Logs. This will allow you to view your audit results. Enabling logging may result in incurred charges.

To enable logging

1. Open the AWS IoT console. On the navigation pane, choose **Settings**.
2. In **Logs**, choose **Manage logs**.
3. For **Select role**, choose **Create role**. Name the role *`AWSIoTLoggingRole`* and choose **Create**. A policy is automatically attached.
4. For **Log level**, choose **Debug (most verbosity)**.
5. Choose **Update**.

ML Detect guide

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender

detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

Note

ML Detect is not available in the following regions:

- Asia Pacific (Malaysia)

In this Getting Started guide, you create an ML Detect Security Profile that uses machine learning (ML) to create models of expected behavior based on historical metric data from your devices. While ML Detect is creating the ML model, you can monitor its progress. After the ML model is built, you can view and investigate alarms on an ongoing basis and mitigate identified issues.

For more information about ML Detect and its API and CLI commands, see ML Detect.

This chapter contains the following sections:

- Prerequisites
- How to use ML Detect in the console
- How to use ML Detect with the CLI

Prerequisites

- An AWS account. If you don't have this, see Setting up.

How to use ML Detect in the console

Tutorials

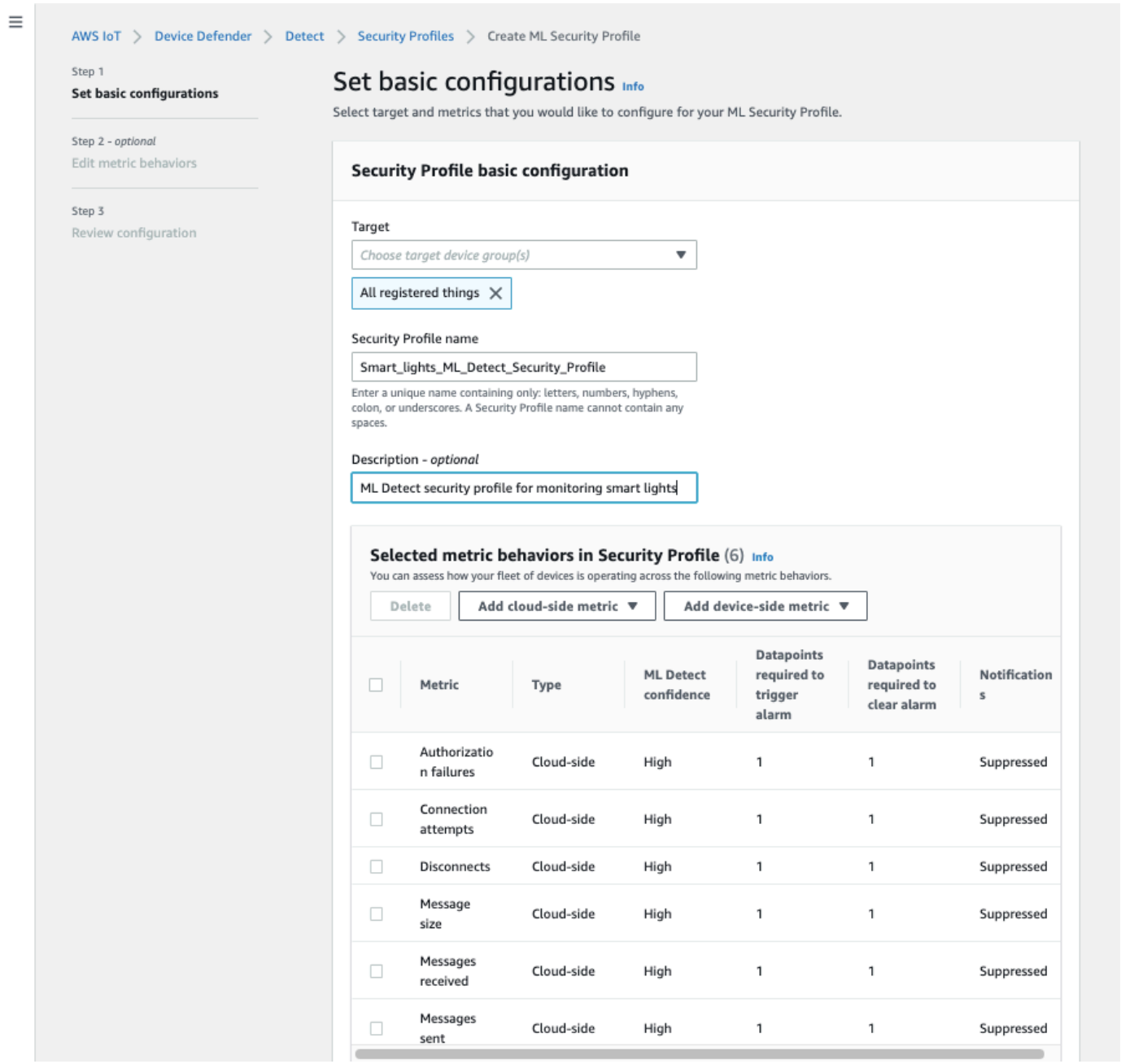
- Enable ML Detect
- Monitor your ML model status
- Review your ML Detect alarms
- Fine-tune your ML alarms
- Mark your alarm's verification state

- Mitigate identified device issues

Enable ML Detect

The following procedures detail how to set up ML Detect in the console.

1. First, make sure your devices will create the minimum datapoints required as defined in ML Detect minimum requirements for ongoing training and refreshing of the model. For data collection to progress, ensure your Security Profile is attached to a target, which can be a thing or thing group.
2. In the AWS IoT console, in the navigation pane, expand **Defend**. Choose **Detect**, **Security profiles**, **Create security profile**, and then **Create ML anomaly Detect profile**.
3. On the **Set basic configurations** page, do the following.
   - Under **Target**, choose your target device groups.
   - Under **Security profile name**, enter a name for your Security Profile.
   - (Optional) Under **Description** you can write in a short description for the ML profile.
   - Under **Selected metric behaviors in Security Profile**, choose the metrics you'd like to monitor.

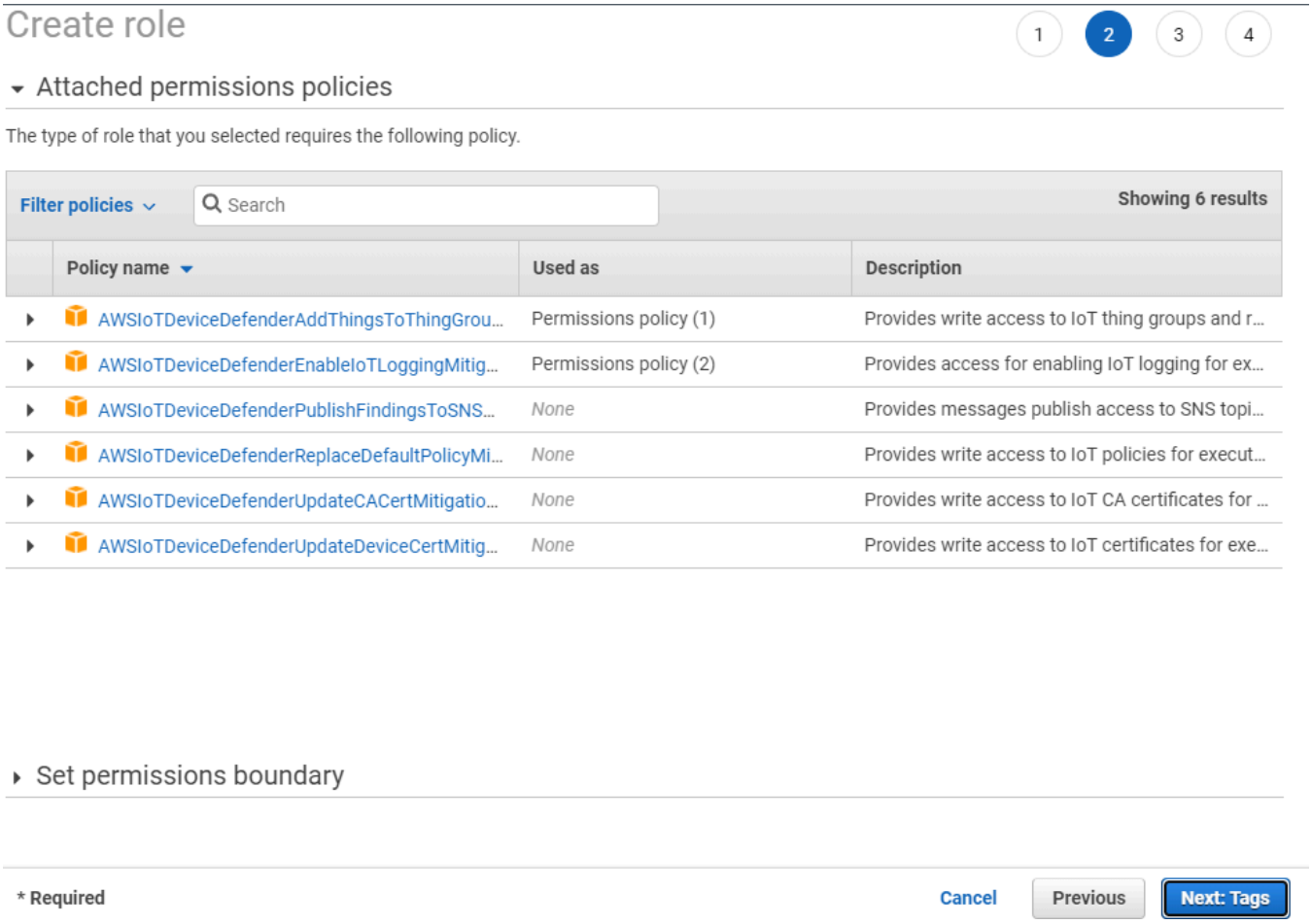


When you're done, choose **Next**.

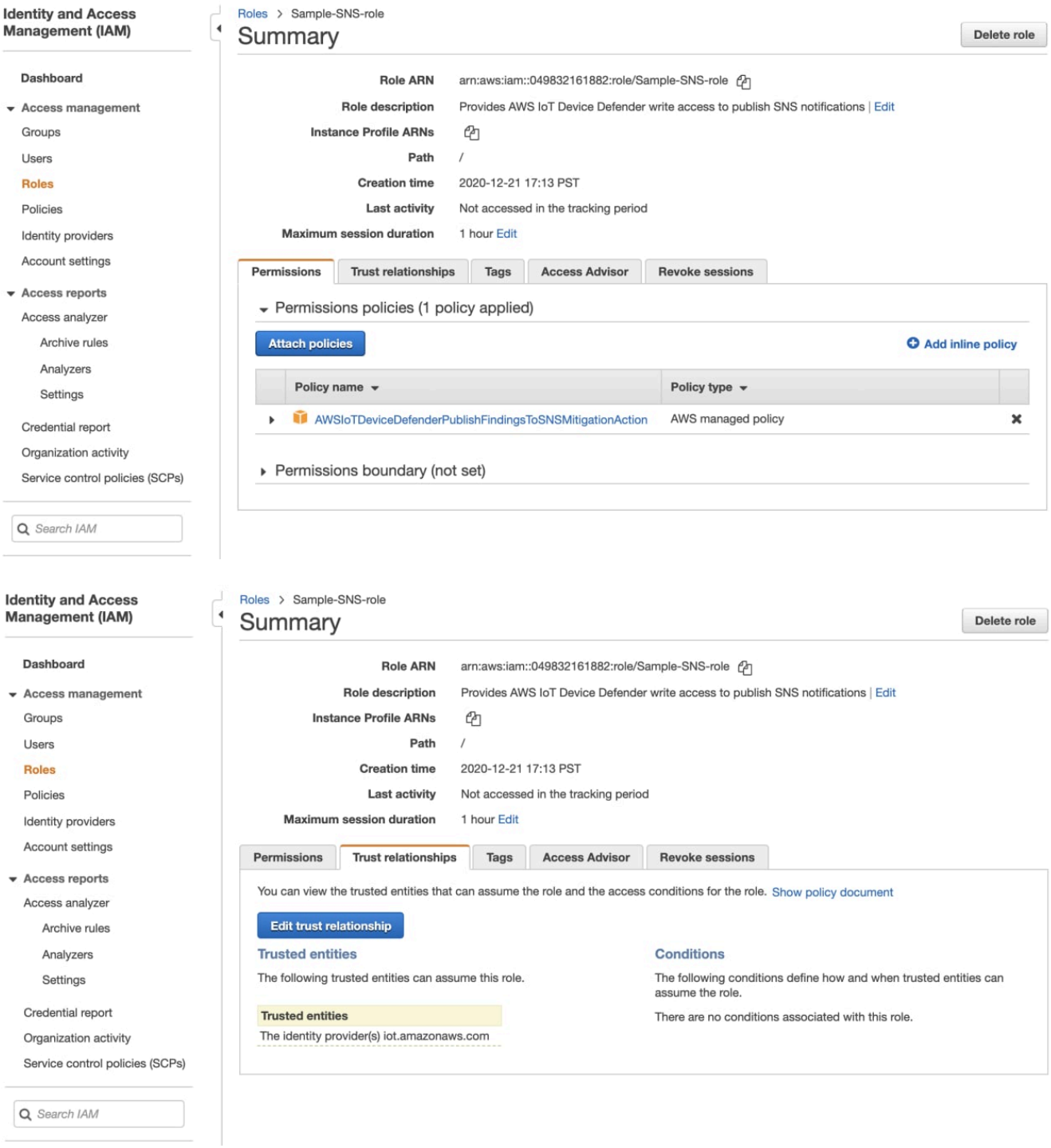
4. On the **Set SNS (optional)** page, specify an SNS topic for alarm notifications when a device violates a behavior in your profile. Choose an IAM role you will use to publish to the selected SNS topic.

   If you don't have an SNS role yet, use the following steps to create a role with the proper permissions and trust relationships required.

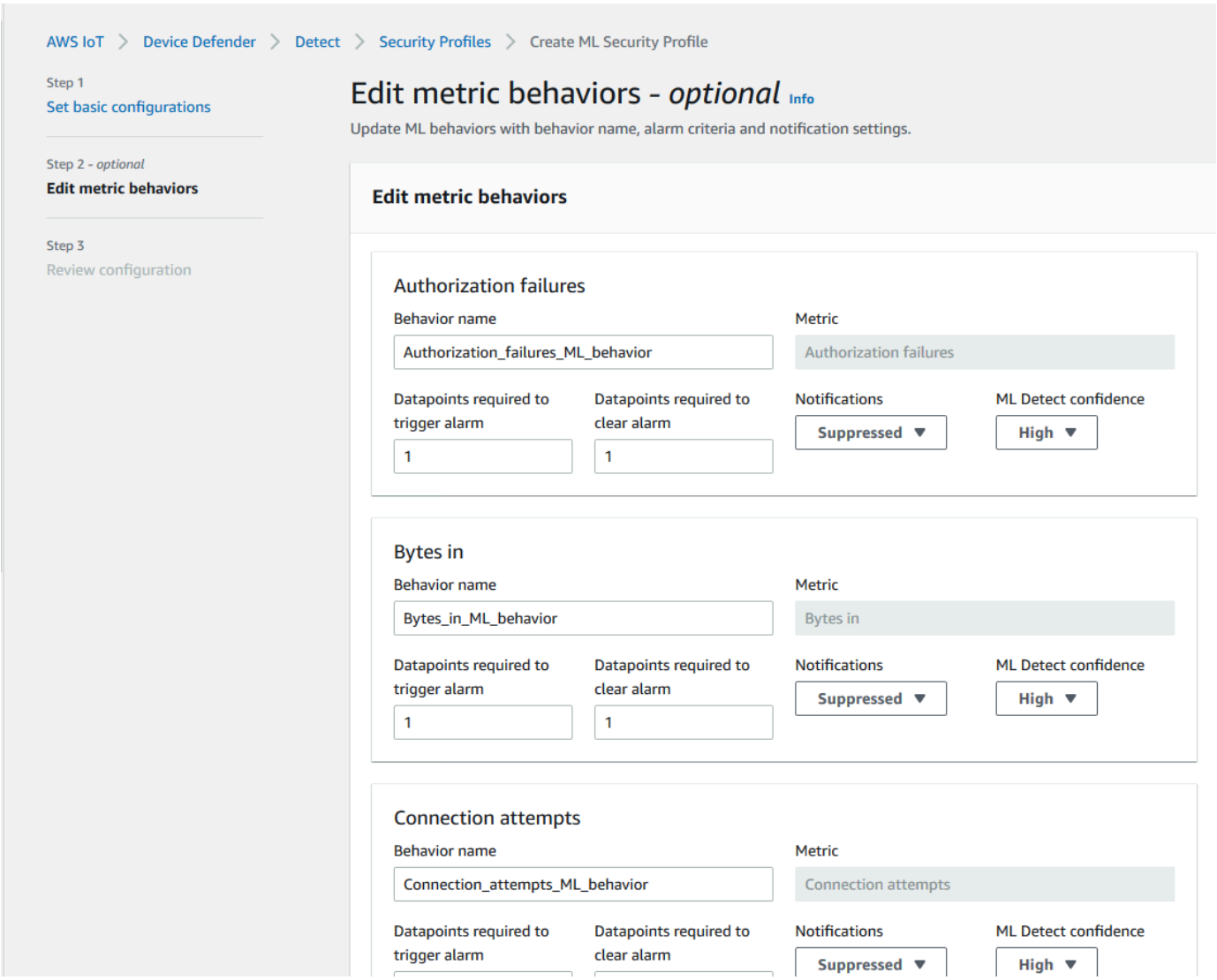
- Navigate to the IAM console. In the navigation pane, choose **Roles** and then choose **Create role**.
- Under **Select type of trusted entity**, select **AWS Service**. Then, under **Choose a use case**, choose **IoT** and under **Select your use case**, choose **IoT - Device Defender Mitigation Actions**. When you're done, choose **Next: Permissions**.
- Under **Attached permissions policies**, ensure that **AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction** is selected, and then choose **Next: Tags**.



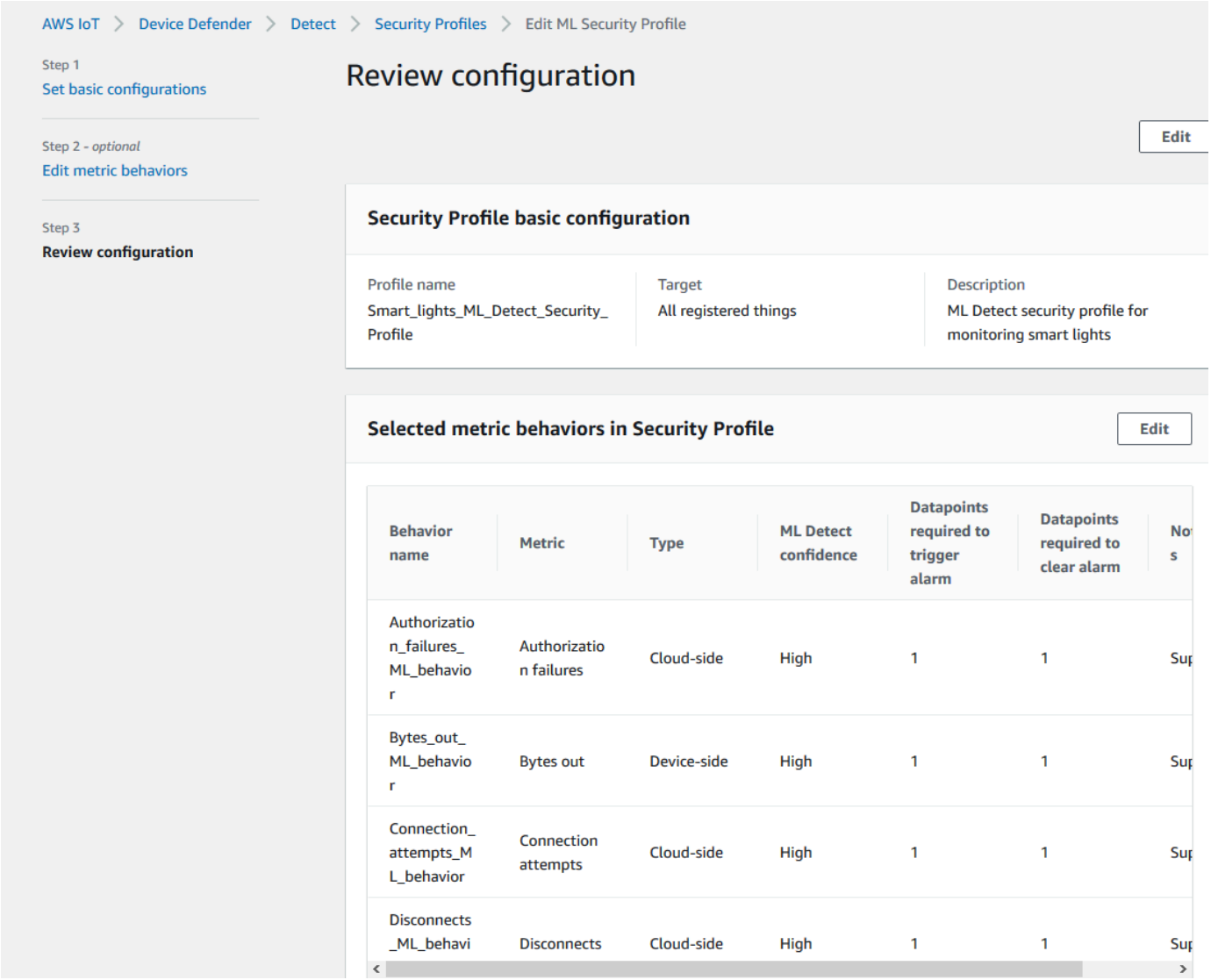
- Under **Add tags (optional)**, you can add any tags you'd like to associate with your role. When you're done, choose **Next: Review**.
- Under **Review**, give your role a name and ensure that **AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction** is listed under **Permissions** and **AWS service: iot.amazonaws.com** is listed under **Trust relationships**. When you're done, choose **Create role**.



5. On the **Edit Metric behavior** page, you can customize your ML behavior settings.



6. When you're done, choose **Next**.
7. On the **Review configuration** page, verify the behaviors you'd like machine learning to monitor, and then choose **Next**.



8. After you've created your Security Profile, you're redirected to the **Security Profiles** page, where the newly created Security Profile appears.

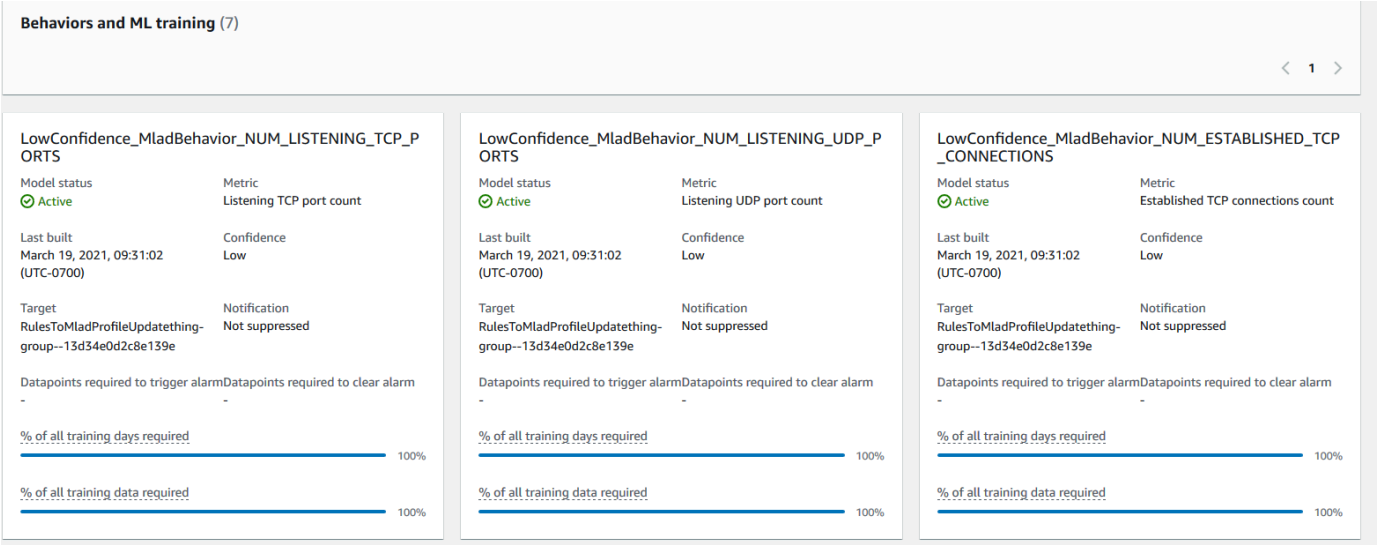
> **Note**
>
> The initial ML model training and creation takes 14 days to complete. You can expect to see alarms after it's complete, if there is any anomalous activity on your devices.

Monitor your ML model status

While your ML models are in the initial training period, you can monitor their progress at any time by taking the following steps.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Security profiles**.
2. On the **Security Profiles** page, choose the Security Profile you'd like to review. Then, choose **Behaviors and ML training**.
3. On the **Behaviors and ML training** page, check the training progress of your ML models.

   After your model status is **Active**, it'll start making Detect decisions based on your usage and update the profile every day.

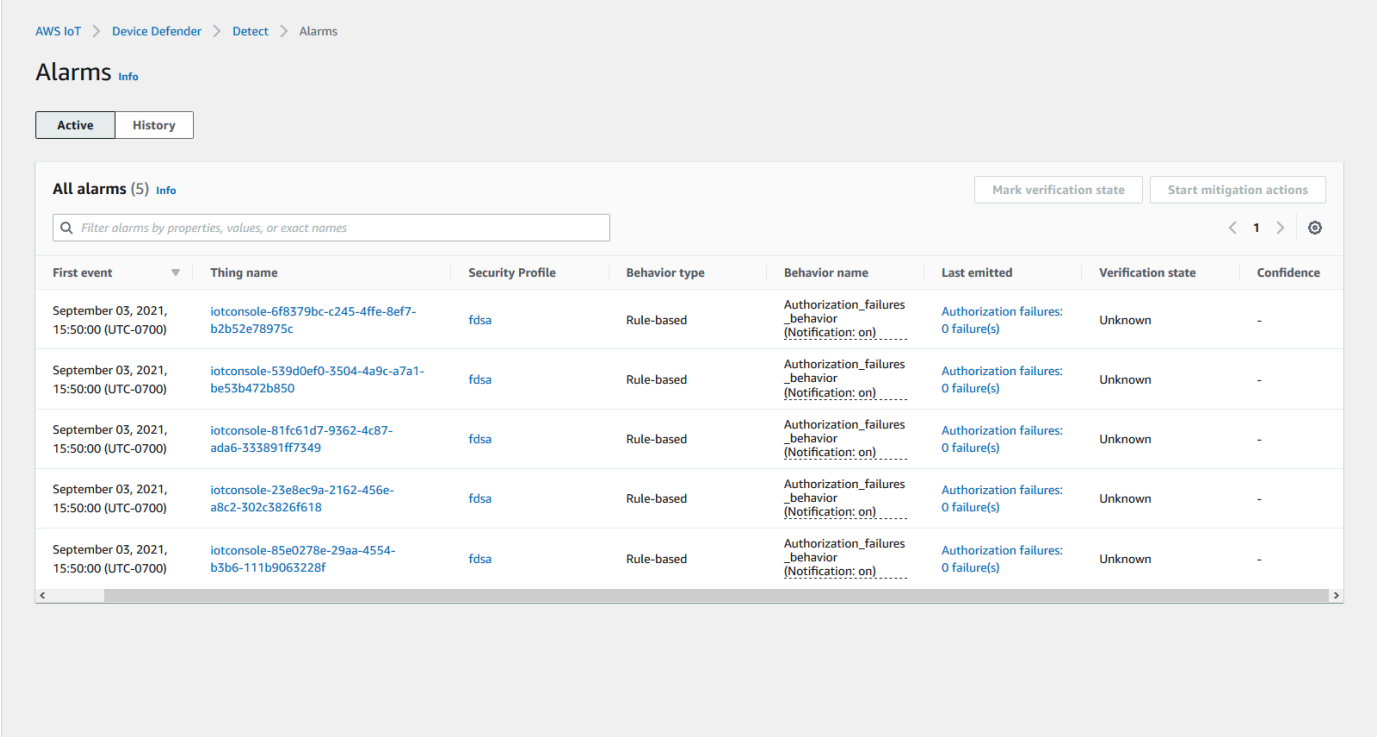


> **Note**
>
> If your model doesn't progress as expected, make sure your devices are meeting the Minimum requirements.

Review your ML Detect alarms

After your ML models are built and ready for data inference, you can regularly view and investigate alarms that are identified by the models.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Alarms**.

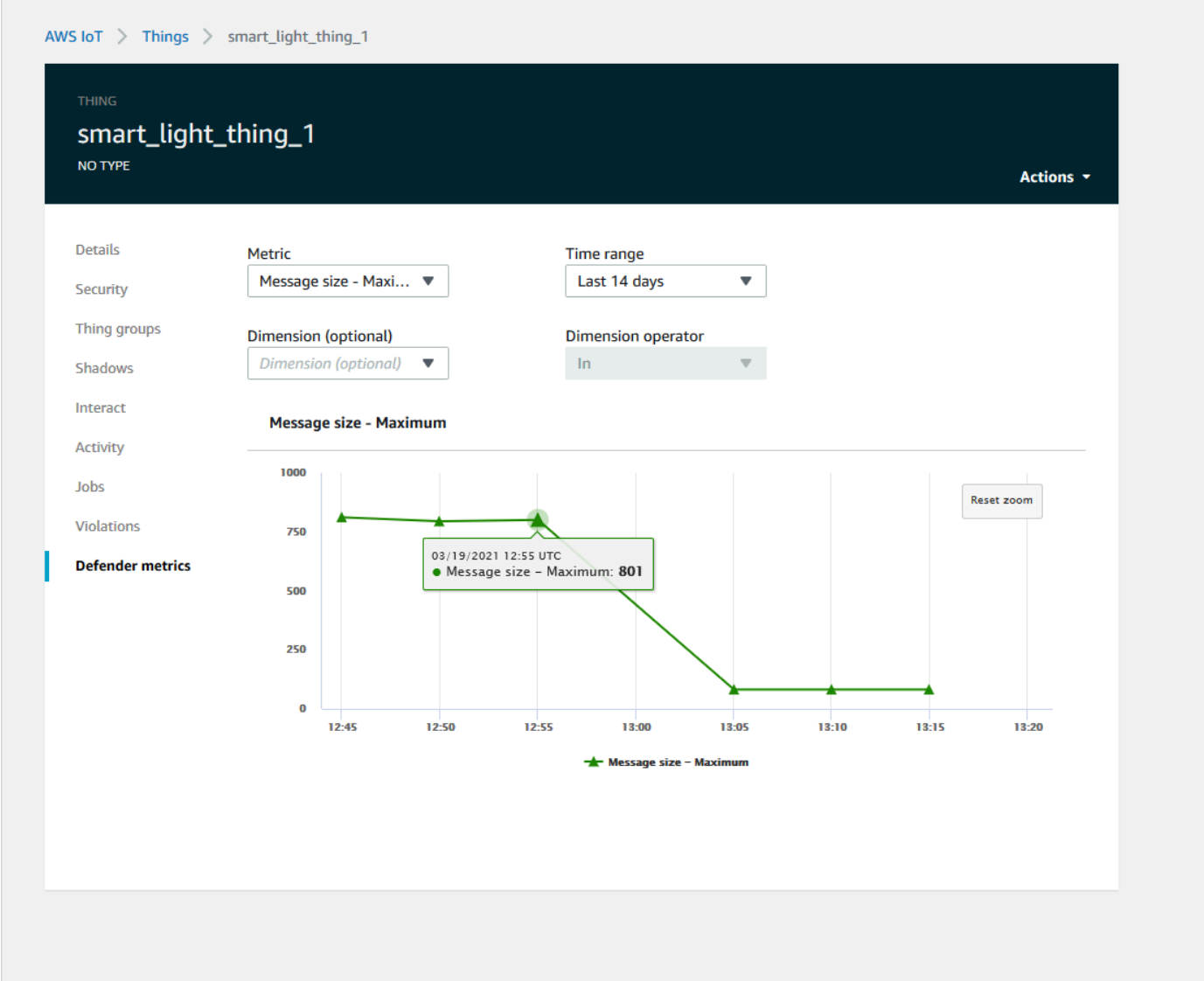


2. If you navigate to the **History** tab, you can also view details about your devices that are no longer in alarms.



To get more information, under **Manage** choose **Things**, chose the thing you'd like to see more details for, and then navigate to **Defender metrics**. You can access the **Defender metrics**

graph and perform your investigation on anything in alarm from the **Active** tab. In this case, the graph shows a spike in message size, which initiated the alarm. You can see the alarm subsequently cleared.

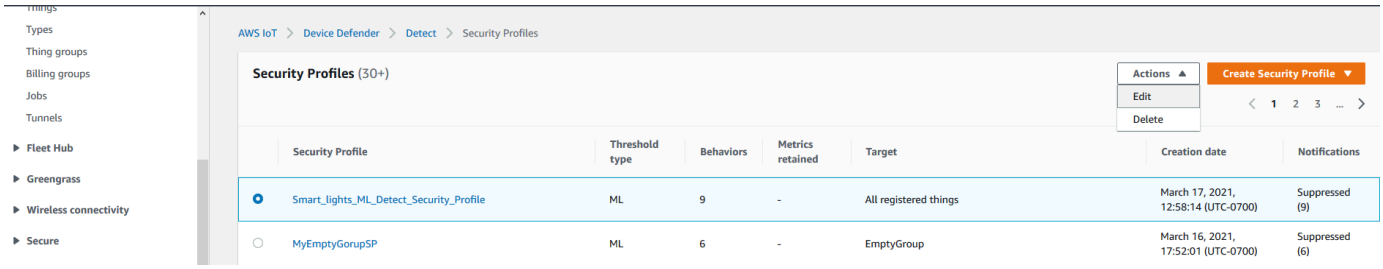


Fine-tune your ML alarms

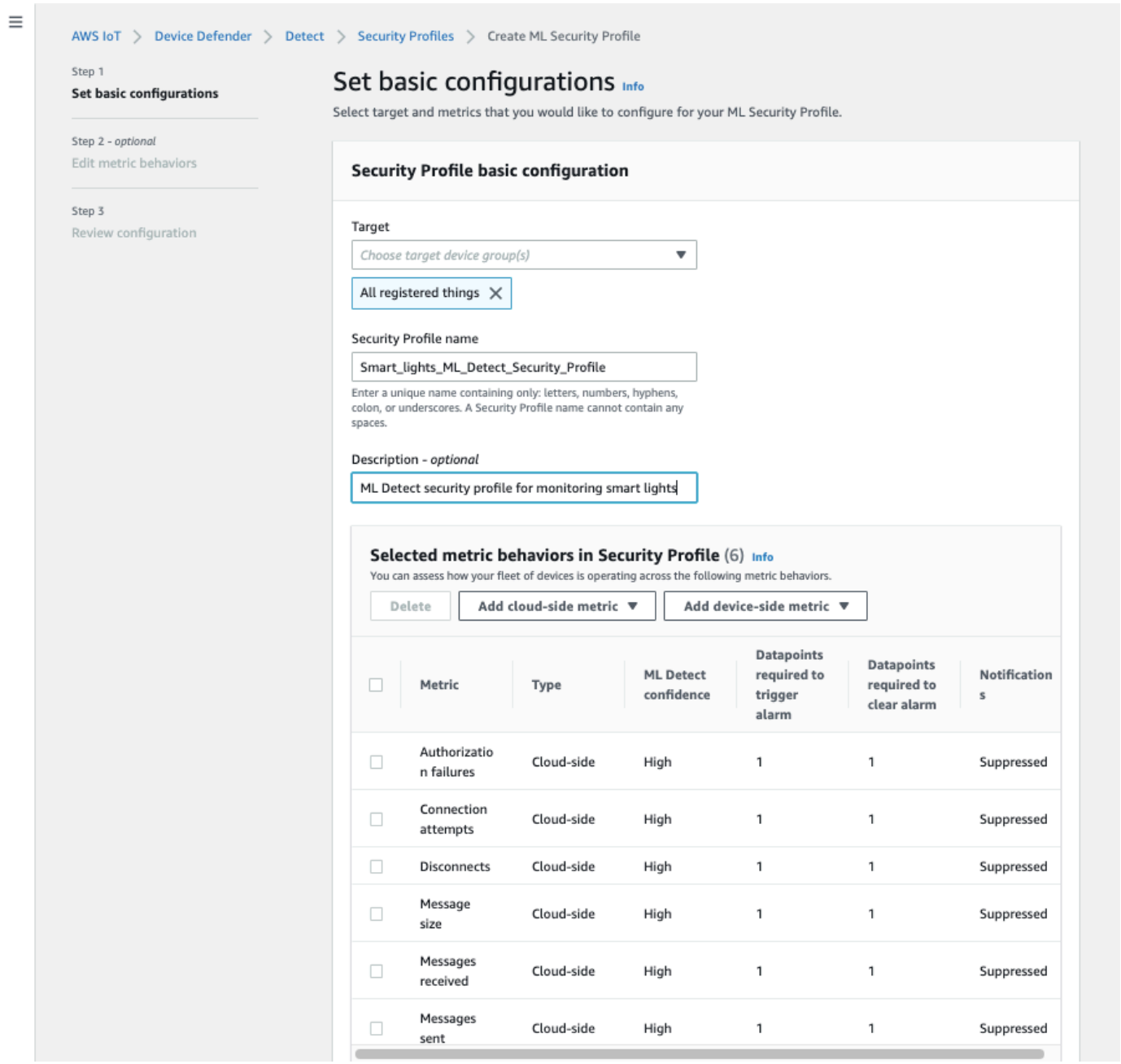
After your ML models are built and ready for data evaluations, you can update your Security Profile's ML behavior settings to change the configuration. The following procedure shows you how to update your Security Profile's ML behavior settings in the AWS CLI.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Security profiles**.

2. On the **Security Profiles** page, select the check box next to the Security Profile you'd like to review. Then, choose **Actions**, **Edit**.

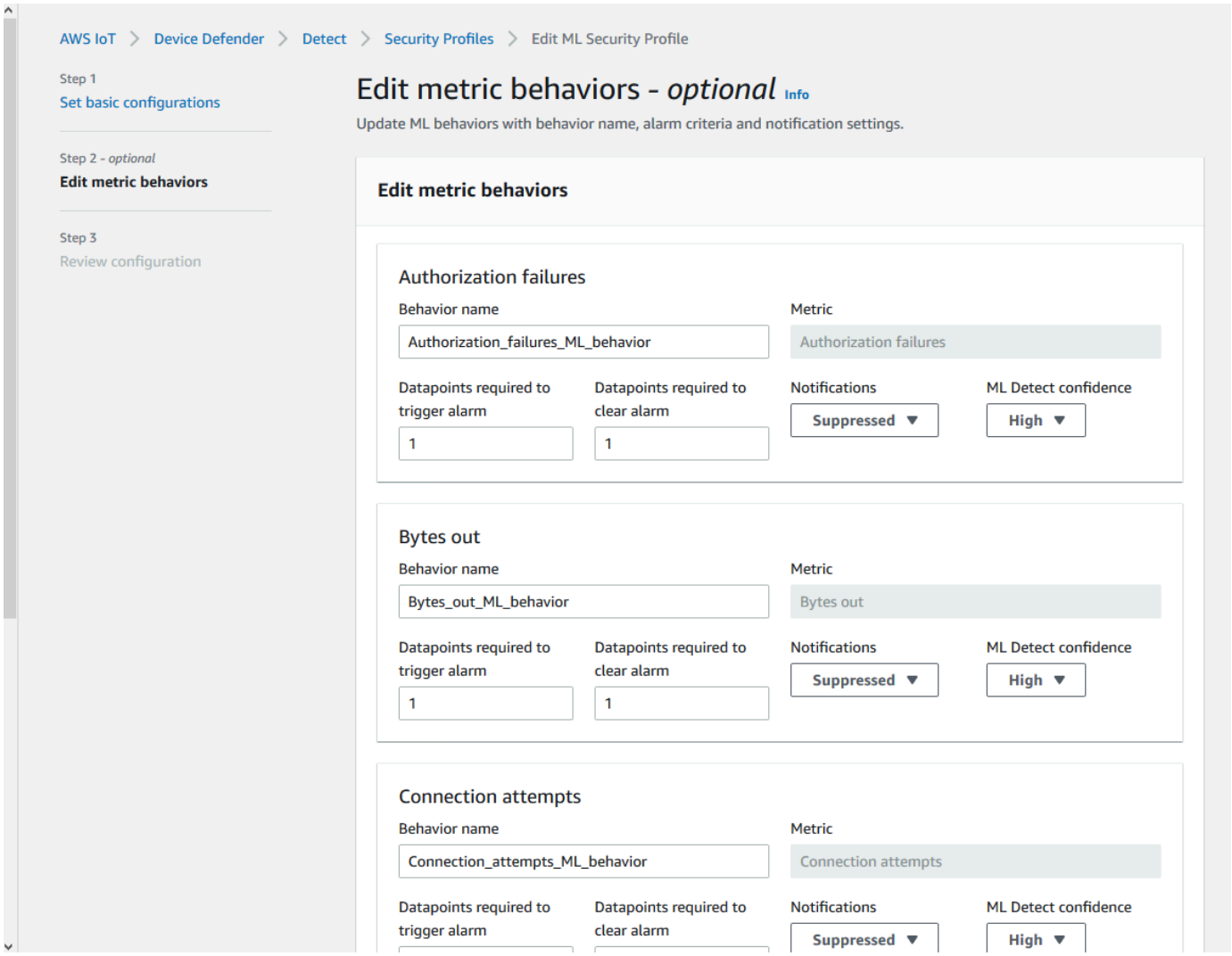


3. Under **Set basic configurations**, you can adjust Security Profile target thing groups or change what metrics you want to monitor.



4. You can update any of the following by navigating to **Edit metric behaviors**.

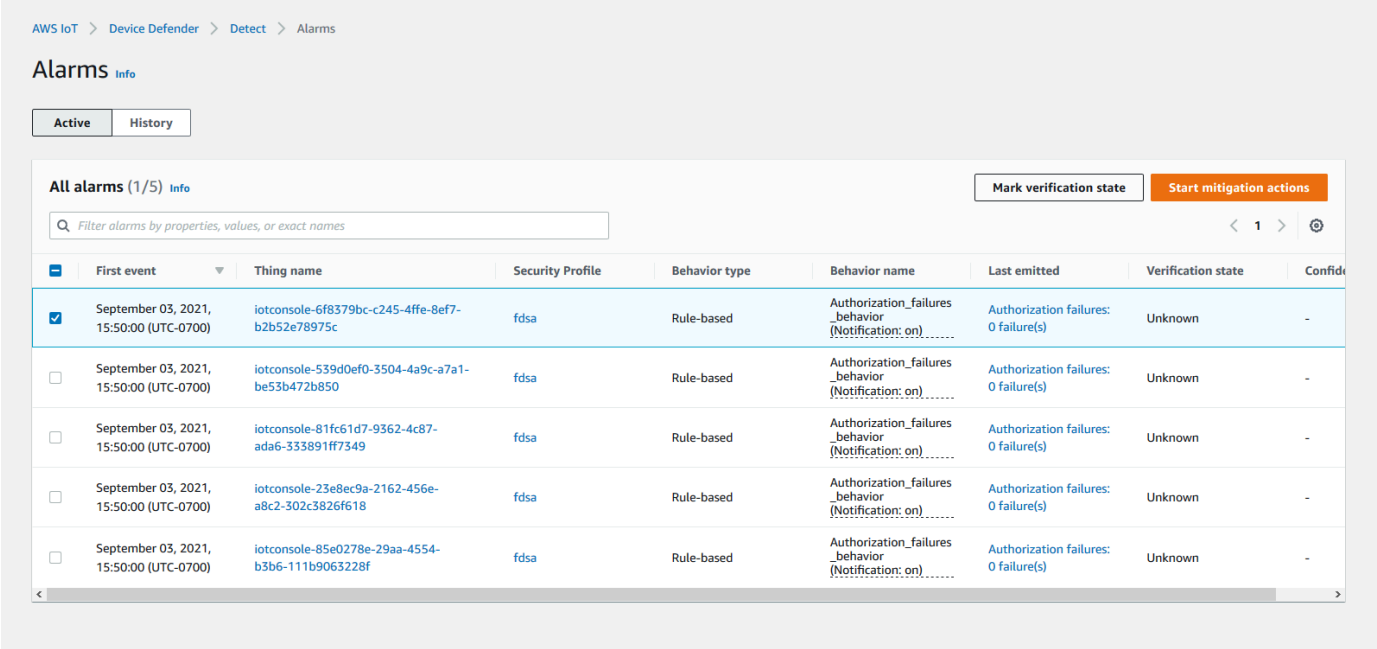
   - Your ML model datapoints required to initiate alarm
   - Your ML model datapoints required to clear alarm
   - Your ML Detect confidence level
   - Your ML Detect notifications (for example, **Not suppressed**, **Suppressed**)



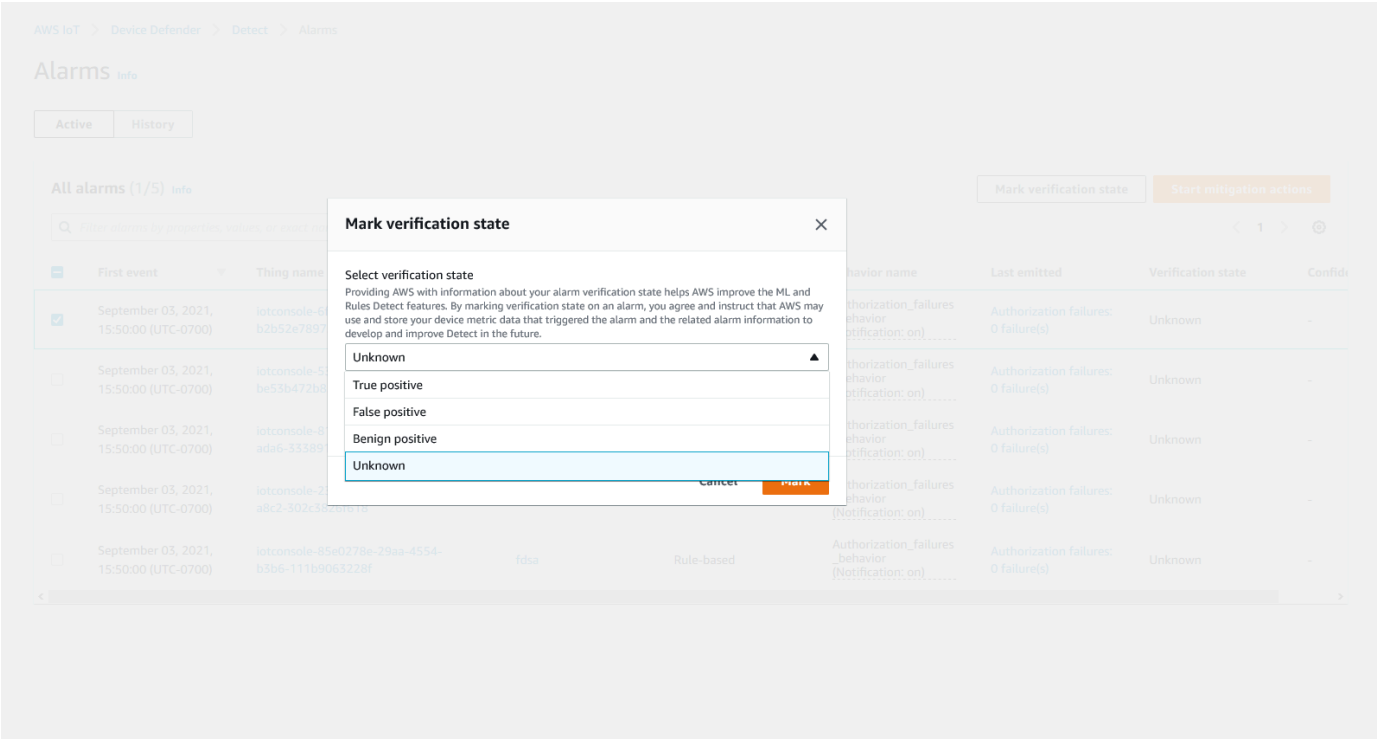
Mark your alarm's verification state

Mark your alarms by setting the verification state and providing a description of that verification state. This helps you and your team identify alarms that you don't have to respond to.

1. In the AWS IoT console, on the navigation pane, expand **Defend**, and then choose **Detect**, **Alarms**. Select an alarm to mark its verification state.



2. Choose **Mark verification state**. The verification state modal opens.
3. Choose the appropriate verification state, enter a verification description (optional), and then choose **Mark**. This action assigns a verification state and description to the chosen alarm.

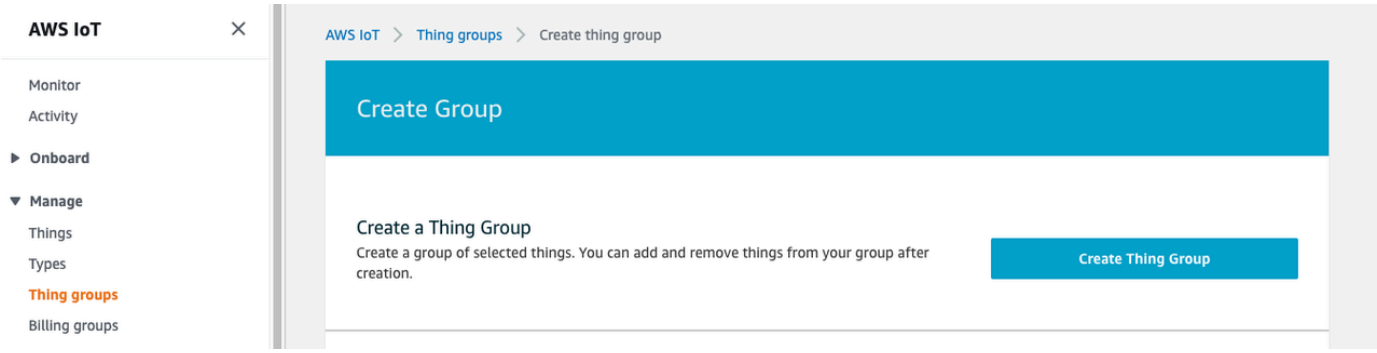


Mitigate identified device issues

1. *(Optional)* Before setting up quarantine mitigation actions, let's set up a quarantine group where we'll move the device that's in violation to. You can also use an existing group.
2. Navigate to **Manage**, **Thing groups**, and then **Create Thing Group**. Name your thing group. For this tutorial, we'll name our thing group `Quarantine_group`. Under **Thing group**, **Security**, apply the following policy to the thing group.

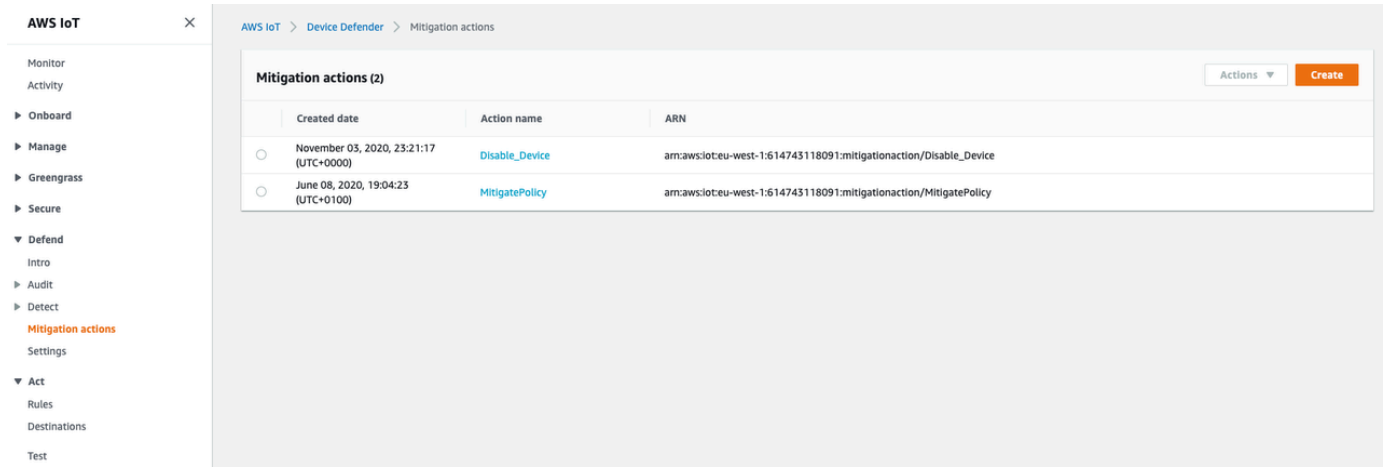
   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Deny",
               "Action": "iot:*",
               "Resource": "*"
           }
       ]
   }
   ```

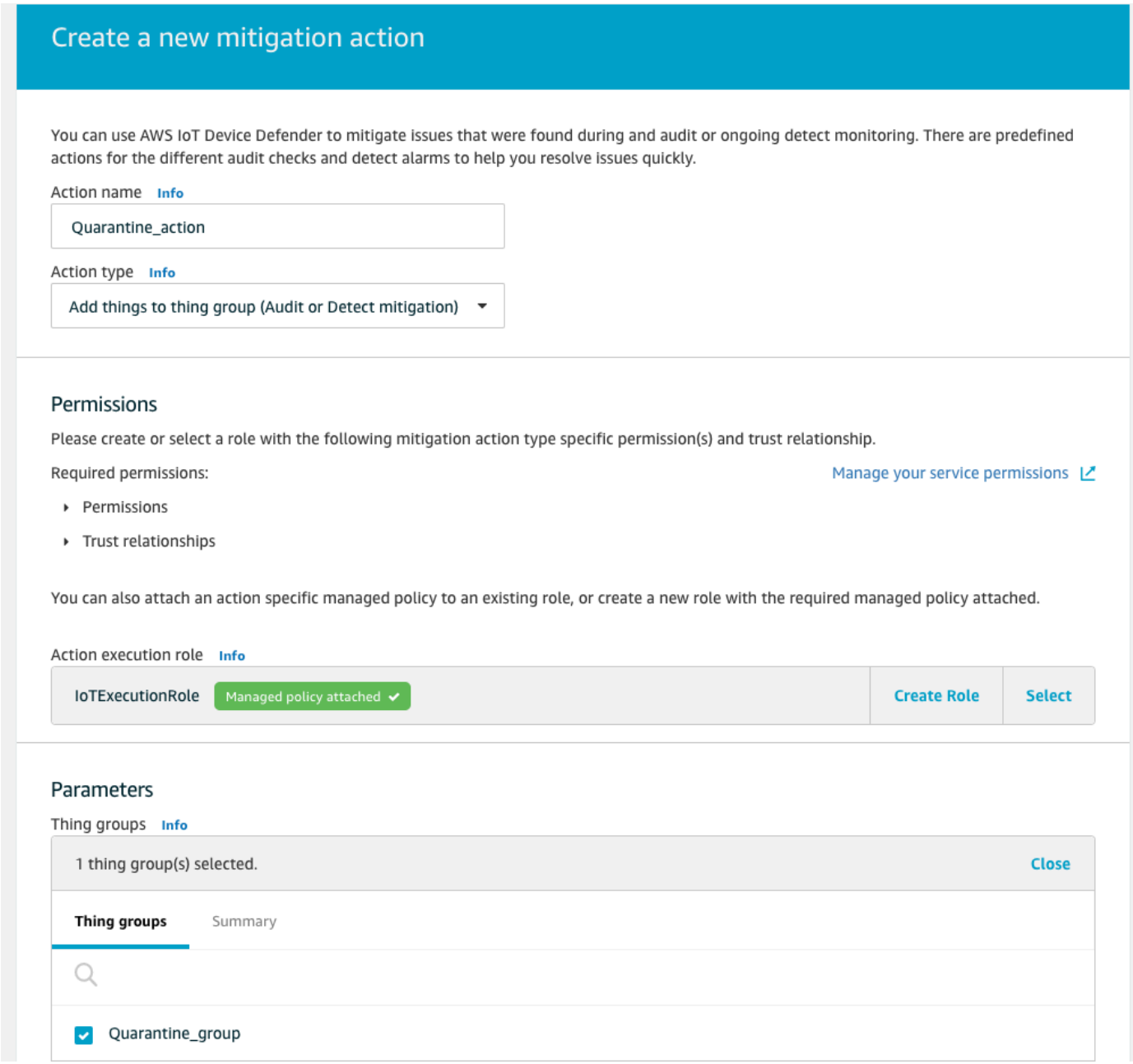
   When you're done, choose **Create thing group**.
3. Now that we've created a thing group, let's create a mitigation action that move devices that in alarm into the `Quarantine_group`.

   Under **Defend**, **Mitigation actions**, choose **Create**.



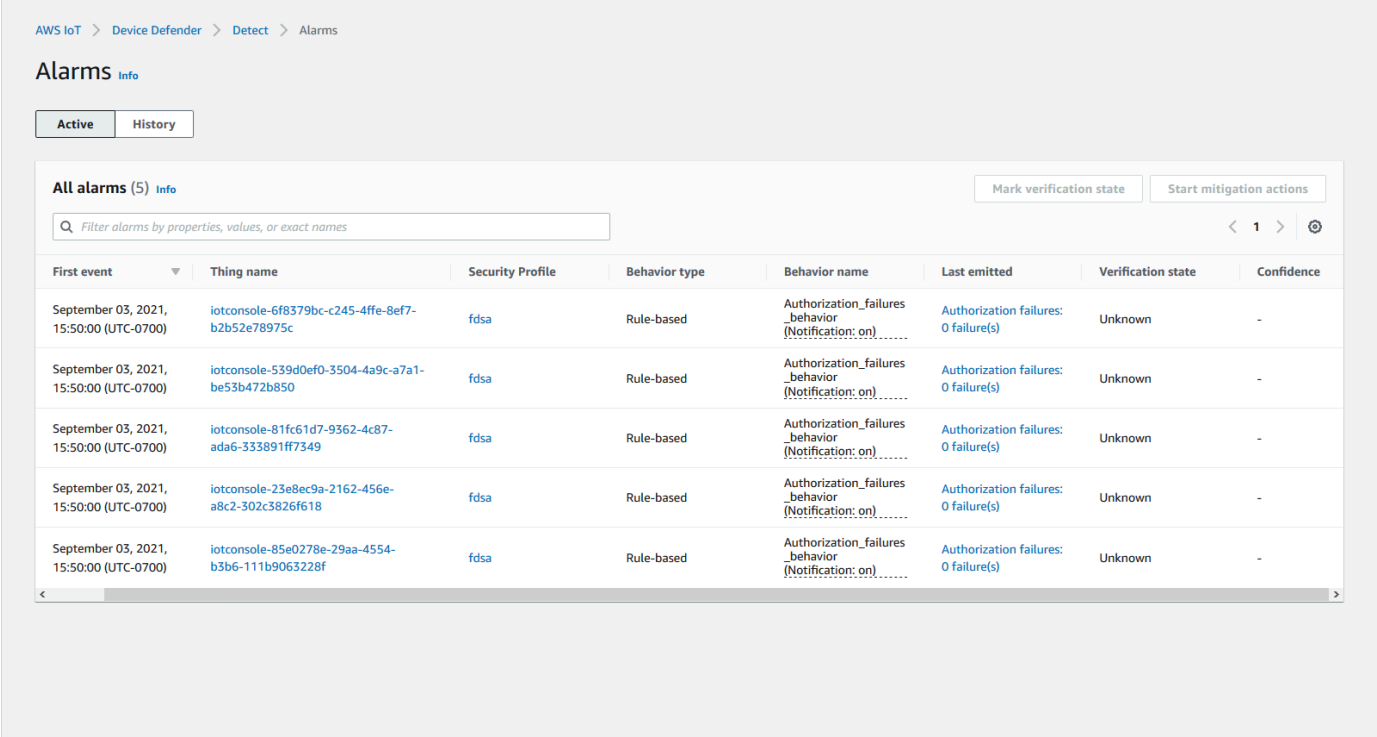
4. On the **Create a new mitigation action** page, enter the following information.

   - **Action name**: Give your mitigation action a name, such as **`Quarantine_action`**.
   - **Action type**: Choose the type of action. We'll choose **Add things to thing group (Audit or Detect mitigation)**.
   - **Action execution role**: Create a role or choose an existing role if you created one earlier.
   - **Parameters**: Choose a thing group. We can use `Quarantine_group`, which we created earlier.



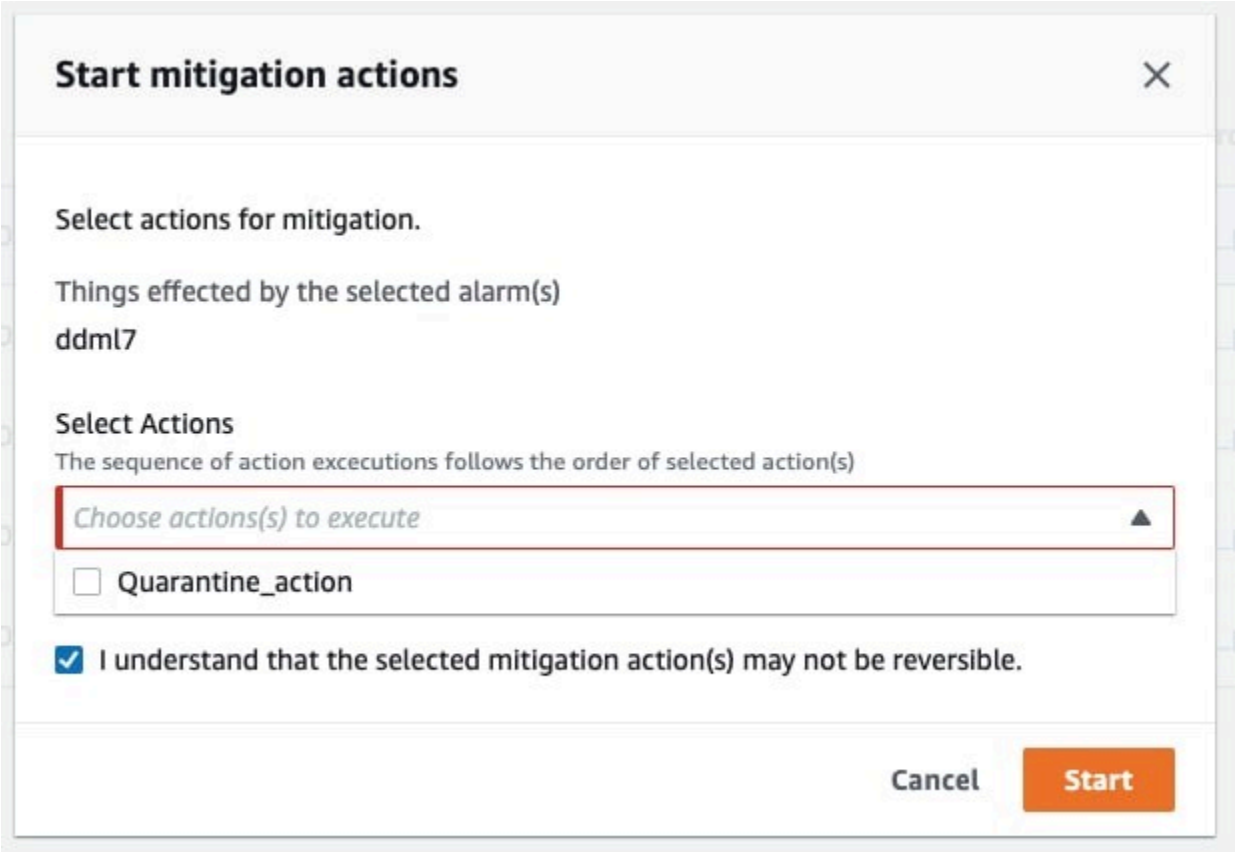
When you're done, choose **Save**. You now have a mitigation action that moves devices in alarm to a quarantine thing group, and a mitigation action to isolate the device while you investigate.

5. Navigate to **Defender**, **Detect**, **Alarms**. You can see which devices are in alarm state under **Active**.

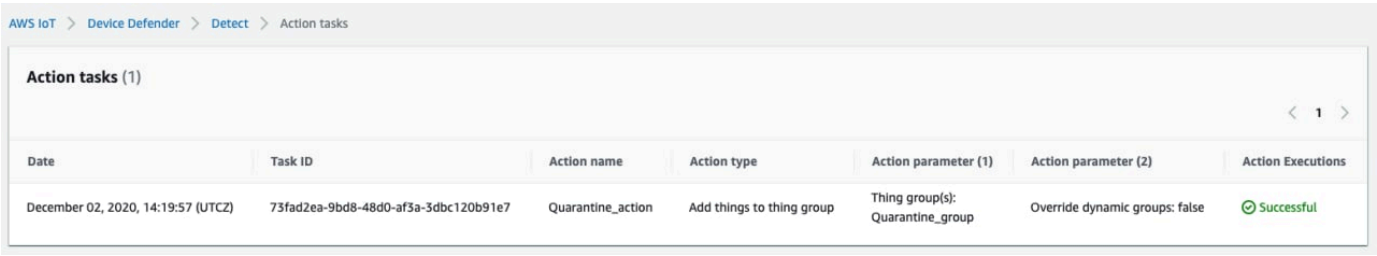


Select the device you want to move to the quarantine group and choose **Start Mitigation Actions**.

6. Under **Start mitigation actions**, **Start Actions** select the mitigation action you created earlier. For example, we'll choose **`Quarantine_action`**, then choose **Start**. The Action Tasks page opens.



7. The device is now isolated in **`Quarantine_group`** and you can investigate the root cause of the issue that set off the alarm. After you complete the investigation, you can move the device out of the thing group or take further actions.



How to use ML Detect with the CLI

The following shows you how to set up ML Detect using the CLI.

Tutorials

- Enable ML Detect
- Monitor your ML model status

- Review your ML Detect alarms
- Fine-tune your ML alarms
- Mark your alarm's verification state
- Mitigate identified device issues

Enable ML Detect

The following procedure shows you how to enable ML Detect in the AWS CLI.

1. Make sure your devices will create the minimum datapoints required as defined in ML Detect minimum requirements for ongoing training and refreshing of the model. For data collection to progress, ensure your things are in a thing group attached to a Security Profile.
2. Create an ML Detect Security Profile by using the `create-security-profile` command. The following example creates a Security Profile named *`security-profile-for-smart-lights`* that checks for number of messages sent, number of authorization failures, number of connection attempts, and number of disconnects. The example uses `mlDetectionConfig` to establish that the metric will use the ML Detect model.

```
aws iot create-security-profile \
    --security-profile-name security-profile-for-smart-lights \
    --behaviors \
     '[{
    "name": "num-messages-sent-ml-behavior",
    "metric": "aws:num-messages-sent",
    "criteria": {
      "consecutiveDatapointsToAlarm": 1,
      "consecutiveDatapointsToClear": 1,
      "mlDetectionConfig": {
        "confidenceLevel": "HIGH"
      }
    },
    "suppressAlerts": true
  },
  {
    "name": "num-authorization-failures-ml-behavior",
    "metric": "aws:num-authorization-failures",
    "criteria": {
      "consecutiveDatapointsToAlarm": 1,
      "consecutiveDatapointsToClear": 1,
```

```
      "mlDetectionConfig": {
        "confidenceLevel": "HIGH"
      }
    },
    "suppressAlerts": true
  },
  {
    "name": "num-connection-attempts-ml-behavior",
    "metric": "aws:num-connection-attempts",
    "criteria": {
      "consecutiveDatapointsToAlarm": 1,
      "consecutiveDatapointsToClear": 1,
      "mlDetectionConfig": {
        "confidenceLevel": "HIGH"
      }
    },
    "suppressAlerts": true
  },
  {
    "name": "num-disconnects-ml-behavior",
    "metric": "aws:num-disconnects",
    "criteria": {
      "consecutiveDatapointsToAlarm": 1,
      "consecutiveDatapointsToClear": 1,
      "mlDetectionConfig": {
        "confidenceLevel": "HIGH"
      }
    },
    "suppressAlerts": true
  }]'
```

Output:

```
{
    "securityProfileName": "security-profile-for-smart-lights",
    "securityProfileArn": "arn:aws:iot:eu-
west-1:123456789012:securityprofile/security-profile-for-smart-lights"
  }
```

3. Next, associate your Security Profile with one or multiple thing groups. Use the [attach-security-profile] command to attach a thing group to your Security Profile. The following

example associates a thing group named *ML_Detect_beta_static_group* with the *security-profile-for-smart-lights* Security Profile.

```
aws iot attach-security-profile \
--security-profile-name security-profile-for-smart-lights \
--security-profile-target-arn arn:aws:iot:eu-
west-1:123456789012:thinggroup/ML_Detect_beta_static_group
```

Output:

None.

4. After you've created your complete Security Profile, the ML model begins training. The initial ML model training and building takes 14 days to complete. After 14 days, if there's anomalous activity on your device, you can expect to see alarms.

Monitor your ML model status

The following procedure shows you how to monitor your ML models in-progress training.

- Use the get-behavior-model-training-summaries command to view your ML model's progress. The following example gets the ML model training progress summary for the *security-profile-for-smart-lights* Security Profile. `modelStatus` shows you if a model has completed training or is still pending build for a particular behavior.

```
aws iot get-behavior-model-training-summaries \
   --security-profile-name security-profile-for-smart-lights
```

Output:

```
{
    "summaries": [
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Messages_sent_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 29.408,
            "lastModelRefreshDate": "2020-12-07T14:35:19.237000-08:00"
        },
```

```
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Messages_received_ML_behavior",
            "modelStatus": "PENDING_BUILD",
            "datapointsCollectionPercentage": 0.0
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Authorization_failures_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 35.464,
            "lastModelRefreshDate": "2020-12-07T14:29:44.396000-08:00"
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Message_size_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 29.332,
            "lastModelRefreshDate": "2020-12-07T14:30:44.113000-08:00"
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Connection_attempts_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 32.891999999999996,
            "lastModelRefreshDate": "2020-12-07T14:29:43.121000-08:00"
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Disconnects_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 35.46,
            "lastModelRefreshDate": "2020-12-07T14:29:55.556000-08:00"
        }
    ]
}
```

> **Note**
>
> If your model doesn't progress as expected, make sure your devices are meeting the Minimum requirements.

Review your ML Detect alarms

After your ML models are built and ready for data evaluations, you can regularly view any alarms that are inferred by the models. The following procedure shows you how to view your alarms in the AWS CLI.

- To see all active alarms, use the `list-active-violations` command.

```
aws iot list-active-violations \
--max-results 2
```

Output:

```
{
    "activeViolations": []
}
```

Alternatively, you can view all violations discovered during a given time period by using the `list-violation-events` command. The following example lists violation events from September 22, 2020 5:42:13 GMT to October 26, 2020 5:42:13 GMT.

```
aws iot list-violation-events \
    --start-time 1599500533 \
    --end-time 1600796533 \
    --max-results 2
```

Output:

```
{
    "violationEvents": [
        {
            "violationId": "1448be98c09c3d4ab7cb9b6f3ece65d6",
            "thingName": "lightbulb-1",
```

```
            "securityProfileName": "security-profile-for-smart-lights",
            "behavior": {
                "name": "LowConfidence_MladBehavior_MessagesSent",
                "metric": "aws:num-messages-sent",
                "criteria": {
                    "consecutiveDatapointsToAlarm": 1,
                    "consecutiveDatapointsToClear": 1,
                    "mlDetectionConfig": {
                        "confidenceLevel": "HIGH"
                    }
                },
                "suppressAlerts": true
            },
            "violationEventType": "alarm-invalidated",
            "violationEventTime": 1600780245.29
        },
        {
            "violationId": "df4537569ef23efb1c029a433ae84b52",
            "thingName": "lightbulb-2",
            "securityProfileName": "security-profile-for-smart-lights",
            "behavior": {
                "name": "LowConfidence_MladBehavior_MessagesSent",
                "metric": "aws:num-messages-sent",
                "criteria": {
                    "consecutiveDatapointsToAlarm": 1,
                    "consecutiveDatapointsToClear": 1,
                    "mlDetectionConfig": {
                        "confidenceLevel": "HIGH"
                    }
                },
                "suppressAlerts": true
            },
            "violationEventType": "alarm-invalidated",
            "violationEventTime": 1600780245.281
        }
    ],
    "nextToken":
 "Amo6XIUrsOohsojuIG6TuwSR3X9iUvH2OCksBZg6bed2j21VSnD1uP1pflxKX1+a3cvBRSosIB0xFv40kM6RYBknZ
vxabMe/ZW31Ps/WiZHlr9Wg7R7eEGli59IJ/U0iBQ1McP/ht0E2XA2TTIvYeMmKQQPsRj/
eoV9j7P/wveu7skNGepU/mvpV0O2Ap7hnV5U+Prx/9+iJA/341va
+pQww7jpUeHmJN9Hw4MqW0ysw0Ry3w38hOQWEpz2xwFWAxAARxeIxCxt5c37RK/lRZBlhYqoB
+w2PZ74730h8pICGY4gktJxkwHyyRabpSM/G/f5DFrD9O5v8idkTZzBxW2jrbzSUIdafPtsZHL/
yAMKr3HAKtaABz2nTsOBNre7X2d/jIjjarhon0Dh9l+8I9Y5Ey
+DIFBcqFTvhibKAafQt3gs6CUiqHdWiCenfJyb8whmDE2qxvdxGElGmRb
```

```
+k6kuN5jrZxxw95gzfYDgRHv11iEn8h1qZLD0czkIFBpMppHj9cetHPvM
+qffXGAzKi8tL6eQuCdMLXmVE3jbqcJcjk9ItnaYJi5zKDz9FVbrz9qZZPtZJFHp"
}
```

Fine-tune your ML alarms

Once your ML models are built and ready for data evaluations, you can update your Security Profile's ML behavior settings to change the configuration. The following procedure shows you how to update your Security Profile's ML behavior settings in the AWS CLI.

- To change your Security Profile's ML behavior settings, use the [update-security-profile] command. The following example updates the *security-profile-for-smart-lights* Security Profile's behaviors by changing the `confidenceLevel` of a few of the behaviors and unsuppresses notifications for all behaviors.

```
aws iot update-security-profile \
    --security-profile-name security-profile-for-smart-lights \
    --behaviors \
     '[{
      "name": "num-messages-sent-ml-behavior",
      "metric": "aws:num-messages-sent",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "HIGH"
          }
      },
      "suppressAlerts": false
  },
  {
      "name": "num-authorization-failures-ml-behavior",
      "metric": "aws:num-authorization-failures",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "HIGH"
          }
      },
      "suppressAlerts": false
  },
  {
      "name": "num-connection-attempts-ml-behavior",
      "metric": "aws:num-connection-attempts",
```

```
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "HIGH"
          }
      },
      "suppressAlerts": false
  },
  {
      "name": "num-disconnects-ml-behavior",
      "metric": "aws:num-disconnects",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "LOW"
          }
      },
      "suppressAlerts": false

  }]'
```

Output:

```
 {
    "securityProfileName": "security-profile-for-smart-lights",
    "securityProfileArn": "arn:aws:iot:eu-
west-1:123456789012:securityprofile/security-profile-for-smart-lights",
    "behaviors": [
        {
            "name": "num-messages-sent-ml-behavior",
            "metric": "aws:num-messages-sent",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "HIGH"
                }
            }
        },
        {
            "name": "num-authorization-failures-ml-behavior",
            "metric": "aws:num-authorization-failures",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "HIGH"
                }
            }
```

```
        },
        {
            "name": "num-connection-attempts-ml-behavior",
            "metric": "aws:num-connection-attempts",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "HIGH"
                }
            },
            "suppressAlerts": false
        },
        {
            "name": "num-disconnects-ml-behavior",
            "metric": "aws:num-disconnects",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "LOW"
                }
            },
            "suppressAlerts": true
        }
    ],
    "version": 2,
    "creationDate": 1600799559.249,
    "lastModifiedDate": 1600800516.856
}
```

Mark your alarm's verification state

You can mark your alarms with verification states to help classify alarms and investigate anomalies.

- Mark your alarms with a verification state and a description of that state. For example to set an alarm's verification state to False positive, use the following command:

```
aws iot put-verification-state-on-violation --violation-id 12345  --verification-
state FALSE_POSITIVE --verification-state-description "This is dummy description"
 --endpoint https://us-east-1.iot.amazonaws.com --region us-east-1
```

Output:

None.

Mitigate identified device issues

1. Use the create-thing-group command to create a thing group for the mitigation action. In the following example, we create a thing group called **ThingGroupForDetectMitigationAction**.

```
aws iot create-thing-group –thing-group-name ThingGroupForDetectMitigationAction
```

Output:

```
{
 "thingGroupName": "ThingGroupForDetectMitigationAction",
 "thingGroupArn": "arn:aws:iot:us-
east-1:123456789012:thinggroup/ThingGroupForDetectMitigationAction",
 "thingGroupId": "4139cd61-10fa-4c40-b867-0fc6209dca4d"
}
```

2. Next, use the create-mitigation-action command to create a mitigation action. In the following example, we create a mitigation action called **detect_mitigation_action** with the ARN of the IAM role that is used to apply the mitigation action. We also define the type of action and the parameters for that action. In this case, our mitigation will move things to our previously created thing group called **ThingGroupForDetectMitigationAction**.

```
aws iot create-mitigation-action --action-name detect_mitigation_action \
--role-arn arn:aws:iam::123456789012:role/MitigationActionValidRole \
--action-params \
'{
     "addThingsToThingGroupParams": {
         "thingGroupNames": ["ThingGroupForDetectMitigationAction"],
         "overrideDynamicGroups": false
     }
 }'
```

Output:

```
{
 "actionArn": "arn:aws:iot:us-
east-1:123456789012:mitigationaction/detect_mitigation_action",
 "actionId": "5939e3a0-bf4c-44bb-a547-1ab59ffe67c3"
}
```

3. Use the `start-detect-mitigation-actions-task` command to start your mitigation actions task. `task-id`, `target` and `actions` are required parameters.

```
aws iot start-detect-mitigation-actions-task \
    --task-id taskIdForMitigationAction \
    --target '{ "violationIds" : [ "violationId-1", "violationId-2" ] }' \
    --actions "detect_mitigation_action" \
    --include-only-active-violations \
    --include-suppressed-alerts
```

Output:

```
{
    "taskId": "taskIdForMitigationAction"
}
```

4. (Optional)To view mitigation action executions included in a task, use the `list-detect-mitigation-actions-executions` command.

```
aws iot list-detect-mitigation-actions-executions \
    --task-id taskIdForMitigationAction \
    --max-items 5 \
    --page-size 4
```

Output:

```
{
    "actionsExecutions": [
        {
            "taskId": "e56ee95e - f4e7 - 459 c - b60a - 2701784290 af",
            "violationId": "214_fe0d92d21ee8112a6cf1724049d80",
            "actionName": "underTest_MAThingGroup71232127",
            "thingName": "cancelDetectMitigationActionsTaskd143821b",
            "executionStartDate": "Thu Jan 07 18: 35: 21 UTC 2021",
            "executionEndDate": "Thu Jan 07 18: 35: 21 UTC 2021",
            "status": "SUCCESSFUL",
        }
    ]
}
```

5. (Optional) Use the describe-detect-mitigation-actions-task command to get information about a mitigation action task.

```
aws iot describe-detect-mitigation-actions-task \
    --task-id taskIdForMitigationAction
```

Output:

```
{
    "taskSummary": {
        "taskId": "taskIdForMitigationAction",
        "taskStatus": "SUCCESSFUL",
        "taskStartTime": 1609988361.224,
        "taskEndTime": 1609988362.281,
        "target": {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "num-messages-sent-ml-behavior"
        },
        "violationEventOccurrenceRange": {
            "startTime": 1609986633.0,
            "endTime": 1609987833.0
        },
        "onlyActiveViolationsIncluded": true,
        "suppressedAlertsIncluded": true,
        "actionsDefinition": [
            {
                "name": "detect_mitigation_action",
                "id": "5939e3a0-bf4c-44bb-a547-1ab59ffe67c3",
                "roleArn":
 "arn:aws:iam::123456789012:role/MitigatioActionValidRole",
                "actionParams": {
                    "addThingsToThingGroupParams": {
                        "thingGroupNames": [
                            "ThingGroupForDetectMitigationAction"
                        ],
                        "overrideDynamicGroups": false
                    }
                }
            }
        ],
        "taskStatistics": {
            "actionsExecuted": 0,
```

```
            "actionsSkipped": 0,
            "actionsFailed": 0
        }
    }
}
```

6. (Optional) To get a list of your mitigation actions tasks, use the [list-detect-mitigation-actions-tasks](#) command.

```
aws iot list-detect-mitigation-actions-tasks \
    --start-time 1609985315 \
    --end-time 1609988915 \
    --max-items 5 \
    --page-size 4
```

Output:

```
{
    "tasks": [
        {
            "taskId": "taskIdForMitigationAction",
            "taskStatus": "SUCCESSFUL",
            "taskStartTime": 1609988361.224,
            "taskEndTime": 1609988362.281,
            "target": {
                "securityProfileName": "security-profile-for-smart-lights",
                "behaviorName": "num-messages-sent-ml-behavior"
            },
            "violationEventOccurrenceRange": {
                "startTime": 1609986633.0,
                "endTime": 1609987833.0
            },
            "onlyActiveViolationsIncluded": true,
            "suppressedAlertsIncluded": true,
            "actionsDefinition": [
                {
                    "name": "detect_mitigation_action",
                    "id": "5939e3a0-bf4c-44bb-a547-1ab59ffe67c3",
                    "roleArn": "arn:aws:iam::123456789012:role/
MitigatioActionValidRole",
                    "actionParams": {
                        "addThingsToThingGroupParams": {
                            "thingGroupNames": [
```

```
                                "ThingGroupForDetectMitigationAction"
                            ],
                            "overrideDynamicGroups": false
                        }
                    }
                }
            ],
            "taskStatistics": {
                "actionsExecuted": 0,
                "actionsSkipped": 0,
                "actionsFailed": 0
            }
        }
    ]
}
```

7. (Optional) To cancel a mitigation actions task, use the [cancel-detect-mitigation-actions-task](#) command.

```
aws iot cancel-detect-mitigation-actions-task \
    --task-id taskIdForMitigationAction
```

Output:

None.

Customize when and how you view AWS IoT Device Defender audit results

AWS IoT Device Defender audit provides periodic security checks to confirm AWS IoT devices and resources are following best practices. For each check, the audit results are categorized as compliant or non-compliant, where non-compliance results in console warning icons. To reduce noise from repeating known issues, the audit finding suppression feature allows you to temporarily silence these non-compliance notifications.

You can suppress select audit checks for a specific resource or account for a predetermined time period. An audit check result that has been suppressed is categorized as a suppressed finding, separate from the compliant and non-compliant categories. This new category doesn't trigger

an alarm like a non-compliant result. This allows you to reduce non-compliance notification disturbances during known maintenance periods or until an update is scheduled to be completed.

Getting started

The following sections detail how you can use audit finding suppressions to suppress a `Device certificate expiring` check in the console and CLI. If you'd like to follow either of the demonstrations, you must first create two expiring certificates for Device Defender to detect.

Use the following to create your certificates.

- Create and register a CA certificate in the *AWS IoT Core Developer Guide*
- Create a client certificate using your CA certificate. In step 3, set your `days` parameter to **1**.

If you're using the CLI to create your certificates, enter the following command.

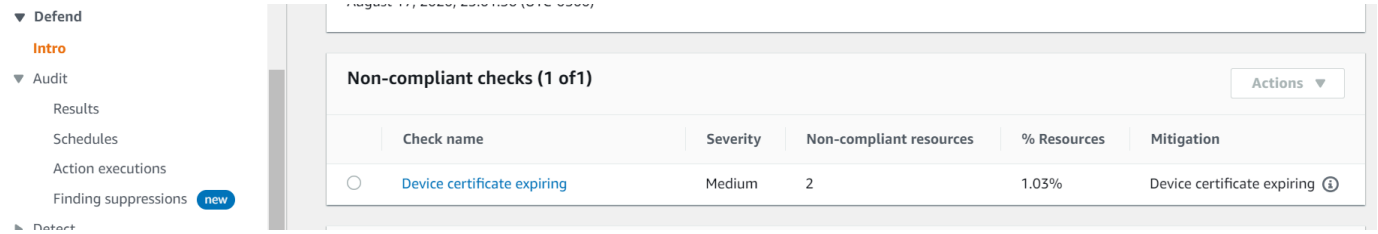
```
openssl x509 -req \
    -in device_cert_csr_filename \
    -CA root_ca_pem_filename \
    -CAkey root_ca_key_filename \
    -CAcreateserial \
    -out device_cert_pem_filename \
    -days 1 -sha256
```

Customize your audit findings in the console

The following walkthrough uses an account with two expired device certificates that trigger a non-compliant audit check. In this scenario, we want to disable the warning because our developers are testing a new feature that'll address the problem. We create an audit finding suppression for each certificate to stop the audit result from being non-compliant for the next week.

1. We will first run an on-demand audit to show that the expired device certificate check is non-compliant.

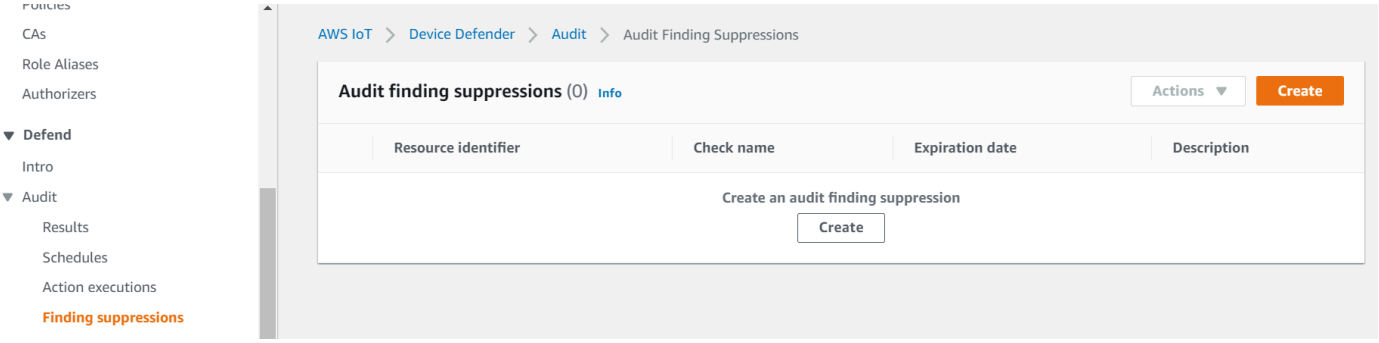
   From the AWS IoT console, choose **Defend** from the left sidebar, then **Audit**, and then **Results**. On the **Audit Results** page, choose **Create**. The **Create a new audit** window opens. Choose **Create**.



From the on-demand audit results, we can see that "Device certificate expiring" is non-compliant for two resources.

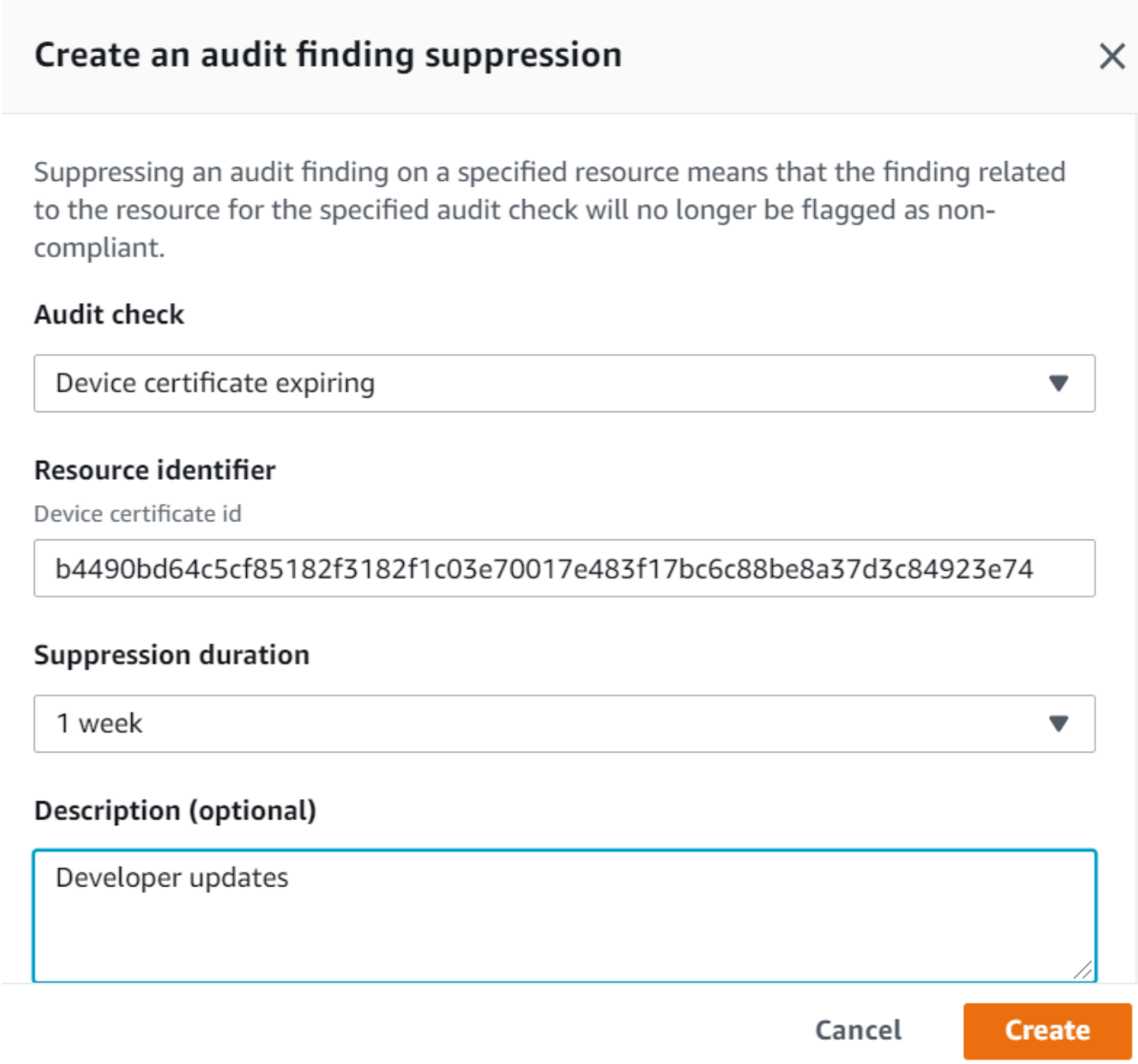
2. Now, we'd like to disable the "Device certificate expiring" non-compliant check warning because our developers are testing new features that will fix the warning.

   From the left sidebar under **Defend**, choose **Audit**, and then choose **Finding suppressions**. On the **Audit finding suppressions** page, choose **Create**.



3. On the **Create an audit finding suppression** window, we need to fill out the following.

   - **Audit check**: We select `Device certificate expiring`, because that is the audit check we'd like to suppress.
   - **Resource identifier**: We input the device certificate ID of one of the certificates we'd like to suppress audit findings for.
   - **Suppression duration**: We select `1 week`, because that's how long we'd like to suppress the `Device certificate expiring` audit check for.
   - **Description (optional)**: We add a note that describes why we're suppressing this audit finding.

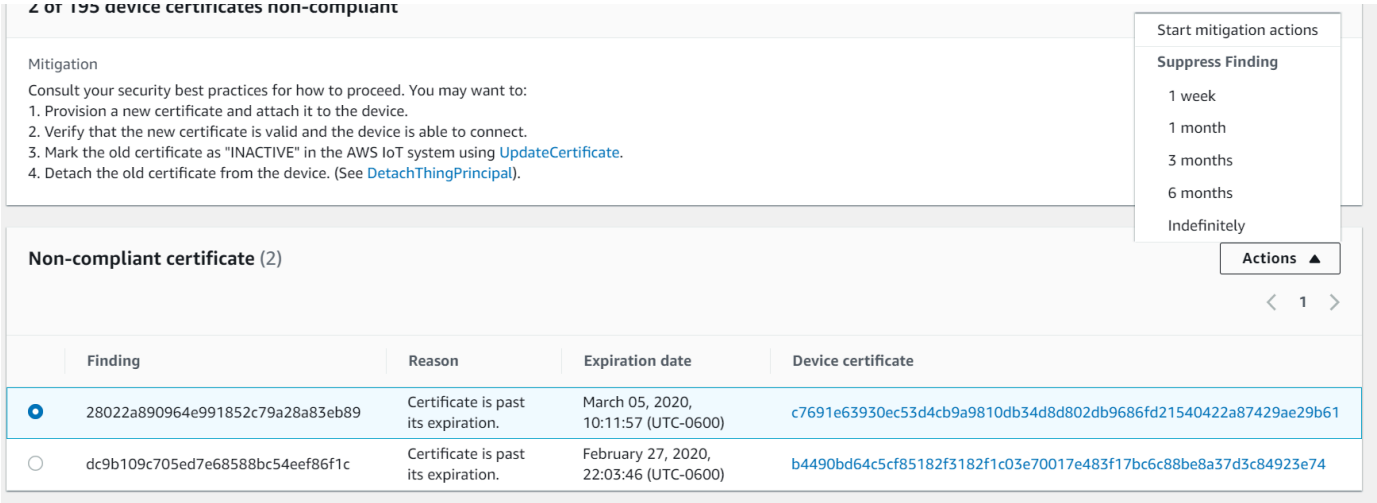


After we've filled out the fields, choose **Create**. We see a success banner after the audit finding suppression has been created.

4. We've suppressed an audit finding for one of the certificates and now we need to suppress the audit finding for the second certificate. We could use the same suppression method that we used in step 3, but we will be using a different method for demonstration purposes.

From the left sidebar under **Defend**, choose **Audit**, and then choose **Results**. On the **Audit results** page, choose the audit with the non-compliant resource. Then, select the resource under **Non-compliant checks**. In our case, we select "Device certificate expiring".

5. On the **Device certificate expiring** page, under **Non-compliant policy** choose the option button next to the finding that needs to be suppressed. Next, choose the **Actions** dropdown menu, and then choose the duration for which you'd like finding to be suppressed. In our case, we choose 1 week as we did for the other certificate. On the **Confirm suppression** window, choose **Enable suppression**.



We see a success banner after the audit finding suppression has been created. Now, both audit findings have been suppressed for 1 week while our developers work on a solution to address the warning.

Customize your audit findings in the CLI

The following walkthrough uses an account with an expired device certificate that trigger a non-compliant audit check. In this scenario, we want to disable the warning because our developers are testing a new feature that'll address the problem. We create an audit finding suppression for the certificate to stop the audit result from being non-compliant for the next week.

We use the following CLI commands.

- create-audit-suppression
- describe-audit-suppression
- update-audit-suppression

- delete-audit-suppression
- list-audit-suppressions

1. Use the following command to enable the audit.

```
aws iot update-account-audit-configuration \
   --audit-check-configurations "{\"DEVICE_CERTIFICATE_EXPIRING_CHECK\":{\"enabled
\":true}}"
```

Output:

None.

2. Use the following command to run an on-demand audit that targets the DEVICE_CERTIFICATE_EXPIRING_CHECK audit check.

```
aws iot start-on-demand-audit-task \
    --target-check-names DEVICE_CERTIFICATE_EXPIRING_CHECK
```

Output:

```
{
    "taskId": "787ed873b69cb4d6cdbae6ddd06996c5"
}
```

3. Use the describe-account-audit-configuration command to describe the audit configuration. We want to confirm that we've turned on the audit check for DEVICE_CERTIFICATE_EXPIRING_CHECK.

```
aws iot describe-account-audit-configuration
```

Output:

```
{
    "roleArn": "arn:aws:iam::<accountid>:role/service-role/project",
    "auditNotificationTargetConfigurations": {
        "SNS": {
            "targetArn": "arn:aws:sns:us-east-1:<accountid>:project_sns",
            "roleArn": "arn:aws:iam::<accountid>:role/service-role/project",
```

```
            "enabled": true
        }
    },
    "auditCheckConfigurations": {
        "AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK": {
            "enabled": false
        },
        "CA_CERTIFICATE_EXPIRING_CHECK": {
            "enabled": false
        },
        "CA_CERTIFICATE_KEY_QUALITY_CHECK": {
            "enabled": false
        },
        "CONFLICTING_CLIENT_IDS_CHECK": {
            "enabled": false
        },
        "DEVICE_CERTIFICATE_EXPIRING_CHECK": {
            "enabled": true
        },
        "DEVICE_CERTIFICATE_KEY_QUALITY_CHECK": {
            "enabled": false
        },
        "DEVICE_CERTIFICATE_SHARED_CHECK": {
            "enabled": false
        },
        "IOT_POLICY_OVERLY_PERMISSIVE_CHECK": {
            "enabled": true
        },
        "IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK": {
            "enabled": false
        },
        "IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK": {
            "enabled": false
        },
        "LOGGING_DISABLED_CHECK": {
            "enabled": false
        },
        "REVOKED_CA_CERTIFICATE_STILL_ACTIVE_CHECK": {
            "enabled": false
        },
        "REVOKED_DEVICE_CERTIFICATE_STILL_ACTIVE_CHECK": {
            "enabled": false
        },
        "UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK": {
```

```
            "enabled": false
        }
    }
}
```

`DEVICE_CERTIFICATE_EXPIRING_CHECK` should have a value of `true`.

4. Use the list-audit-task command to identify the completed audit tasks.

```
aws iot list-audit-tasks \
    --task-status "COMPLETED" \
    --start-time 2020-07-31 \
    --end-time 2020-08-01
```

Output:

```
{
    "tasks": [
        {
            "taskId": "787ed873b69cb4d6cdbae6ddd06996c5",
            "taskStatus": "COMPLETED",
            "taskType": "SCHEDULED_AUDIT_TASK"
        }
    ]
}
```

The `taskId` of the audit you ran in step 1 should have a `taskStatus` of `COMPLETED`.

5. Use the describe-audit-task command to get details about the completed audit using the `taskId` output from the previous step. This command lists details about your audit.

```
aws iot describe-audit-task \
    --task-id "787ed873b69cb4d6cdbae6ddd06996c5"
```

Output:

```
{
    "taskStatus": "COMPLETED",
    "taskType": "SCHEDULED_AUDIT_TASK",
    "taskStartTime": 1596168096.157,
    "taskStatistics": {
        "totalChecks": 1,
```

```
        "inProgressChecks": 0,
        "waitingForDataCollectionChecks": 0,
        "compliantChecks": 0,
        "nonCompliantChecks": 1,
        "failedChecks": 0,
        "canceledChecks": 0
    },
    "scheduledAuditName": "AWSIoTDeviceDefenderDailyAudit",
    "auditDetails": {
        "DEVICE_CERTIFICATE_EXPIRING_CHECK": {
            "checkRunStatus": "COMPLETED_NON_COMPLIANT",
            "checkCompliant": false,
            "totalResourcesCount": 195,
            "nonCompliantResourcesCount": 2
        }
    }
}
```

6. Use the list-audit-findings command to find the non-compliant certificate ID so that we can suspend the audit alerts for this resource.

```
aws iot list-audit-findings \
    --start-time 2020-07-31 \
    --end-time 2020-08-01
```

Output:

```
{
    "findings": [
        {
            "findingId": "296ccd39f806bf9d8f8de20d0ceb33a1",
            "taskId": "787ed873b69cb4d6cdbae6ddd06996c5",
            "checkName": "DEVICE_CERTIFICATE_EXPIRING_CHECK",
            "taskStartTime": 1596168096.157,
            "findingTime": 1596168096.651,
            "severity": "MEDIUM",
            "nonCompliantResource": {
                "resourceType": "DEVICE_CERTIFICATE",
                "resourceIdentifier": {
                    "deviceCertificateId": "b4490<shortened>"
                },
                "additionalInfo": {
                "EXPIRATION_TIME": "1582862626000"
```

```
                }
            },
            "reasonForNonCompliance": "Certificate is past its expiration.",
            "reasonForNonComplianceCode": "CERTIFICATE_PAST_EXPIRATION",
            "isSuppressed": false
        },
        {
            "findingId": "37ecb79b7afb53deb328ec78e647631c",
            "taskId": "787ed873b69cb4d6cdbae6ddd06996c5",
            "checkName": "DEVICE_CERTIFICATE_EXPIRING_CHECK",
            "taskStartTime": 1596168096.157,
            "findingTime": 1596168096.651,
            "severity": "MEDIUM",
            "nonCompliantResource": {
                "resourceType": "DEVICE_CERTIFICATE",
                "resourceIdentifier": {
                    "deviceCertificateId": "c7691<shortened>"
                },
                "additionalInfo": {
                "EXPIRATION_TIME": "1583424717000"
                }
            },
            "reasonForNonCompliance": "Certificate is past its expiration.",
            "reasonForNonComplianceCode": "CERTIFICATE_PAST_EXPIRATION",
            "isSuppressed": false
        }
    ]
}
```

7. Use the [create-audit-suppression] command to suppress notifications for the `DEVICE_CERTIFICATE_EXPIRING_CHECK` audit check for a device certificate with the id *c7691e<shortened>* until *2020-08-20*.

```
aws iot create-audit-suppression \
    --check-name DEVICE_CERTIFICATE_EXPIRING_CHECK \
    --resource-identifier deviceCertificateId="c7691e<shortened>" \
    --no-suppress-indefinitely \
    --expiration-date 2020-08-20
```

8. Use the [list-audit-suppression] command to confirm the audit suppression setting and get details about the suppression.

```
aws iot list-audit-suppressions
```

Output:

```
{
    "suppressions": [
        {
        "checkName": "DEVICE_CERTIFICATE_EXPIRING_CHECK",
            "resourceIdentifier": {
                "deviceCertificateId": "c7691e<shortened>"
            },
        "expirationDate": 1597881600.0,
        "suppressIndefinitely": false
        }
    ]
}
```

9. The update-audit-suppression command can be used to update the audit finding suppression. The example below updates the `expiration-date` to `08/21/20`.

```
aws iot update-audit-suppression \
    --check-name DEVICE_CERTIFICATE_EXPIRING_CHECK \
    --resource-identifier deviceCertificateId=c7691e<shortened> \
    --no-suppress-indefinitely \
    --expiration-date 2020-08-21
```

10. The delete-audit-suppression command can be used to remove an audit finding suppression.

```
aws iot delete-audit-suppression \
    --check-name DEVICE_CERTIFICATE_EXPIRING_CHECK \
    --resource-identifier deviceCertificateId="c7691e<shortened>"
```

To confirm deletion, use the list-audit-suppressions command.

```
aws iot list-audit-suppressions
```

Output:

```
{
 "suppressions": []
```

```
}
```

In this tutorial, we showed you how to suppress a `Device certificate expiring` check in the console and CLI. For more information about audit finding suppressions, see Audit finding suppressions

Audit

An AWS IoT Device Defender audit looks at account- and device-related settings and policies to ensure security measures are in place. An audit can help you detect any drifts from security best practices or access policies (for example, multiple devices using the same identity, or overly permissive policies that allow one device to read and update data for many other devices). You can run audits as needed (*on-demand audits*) or schedule them to be run periodically (*scheduled audits*).

An AWS IoT Device Defender audit runs a set of predefined checks for common IoT security best practices and device vulnerabilities. Examples of predefined checks include policies that grant permission to read or update data on multiple devices, devices that share an identity (X.509 certificate), or certificates that are expiring or have been revoked but are still active.

Issue severity

Issue severity indicates the level of concern associated with each identified instance of noncompliance and the recommended time to remediation.

Critical

Non compliant audit checks with this severity identify issues that require urgent attention. Critical issues often allow bad actors with little sophistication and no insider knowledge or special credentials to easily gain access to or control of your assets.

High

Non compliant audit checks with this severity require urgent investigation and remediation planning after critical issues are addressed. Like critical issues, high severity issues often provide bad actors with access to or control of your assets. However, high severity issues are often more difficult to exploit. They might require special tools, insider knowledge, or specific setups.

Medium

Non compliant audit checks with this severity present issues that need attention as part of your continuous security posture maintenance. Medium severity issues might cause negative operational impact, such as unplanned outages due to malfunction of security controls. These issues might also provide bad actors with limited access to or control of your assets, or might facilitate parts of their malicious actions.

Low

Non compliant audit checks with this severity often indicate security best practices were overlooked or bypassed. Although they might not cause an immediate security impact on their own, these lapses can be exploited by bad actors. Like medium severity issues, low severity issues require attention as part of your continuous security posture maintenance.

Next steps

To understand the types of audit checks that can be performed, see Audit checks. For information about service quotas that apply to audits, see Service Quotas.

Audit checks

> **Note**
>
> When you enable a check, data collection starts immediately. If there is a large amount of data in your account to collect, results of the check might not be available for some time after you enabled it.

The following audit checks are supported:

- Intermediate CA revoked for active device certificates check
- Revoked CA certificate still active
- Device certificate shared
- Device certificate key quality
- CA certificate key quality
- Unauthenticated Cognito role overly permissive
- Authenticated Cognito role overly permissive
- AWS IoT policies overly permissive
- AWS IoT policy potentially misconfigured
- Role alias overly permissive
- Role alias allows access to unused services

- CA certificate expiring
- Conflicting MQTT client IDs
- Device certificate expiring
- Device certificate age check
- Revoked device certificate still active
- Logging disabled

Intermediate CA revoked for active device certificates check

Use this check to identify all related device certificates that are still active despite revoking an intermediate CA.

This check appears as `INTERMEDIATE_CA_REVOKED_FOR_ACTIVE_DEVICE_CERTIFICATES_CHECK` in the CLI and API.

Severity: **Critical**

Details

The following reason codes are returned when this check finds noncompliance:

- INTERMEDIATE_CA_REVOKED_BY_ISSUER

Why it matters

The intermediate CA revoked for active device certificates check assess device identity and trust, by determining if there are active device certificates in AWS IoT Core where the intermediate issuing CAs have been revoked in the CA chain.

A revoked intermediate CA should no longer be used to sign any other CA or device certificates in CA chain. Newly added devices with certificates signed using this CA certificate after the intermediate CA is revoked will pose a security threat.

How to fix it

Review the device certificate registration activity for the time after the CA certificate was revoked. Follow your security best practices to mitigate the situation. You might want to:

1. Provision new certificates, that are signed by a different CA, for the affected devices.
2. Verify that the new certificates are valid, and that the devices can use them to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.
   - Review the device certificate registration activity for the time after the intermediate CA certificate was revoked and consider revoking any device certificates that might have been issued with it during this time. You can use ListRelatedResourcesForAuditFinding to list the device certificates signed by the CA certificate and UpdateCertificate to revoke a device certificate.
   - Detach the old certificate from the device. (See DetachThingPrincipal.)

   For more information, see Mitigation actions.

Revoked CA certificate still active

A CA certificate was revoked, but is still active in AWS IoT.

This check appears as `REVOKED_CA_CERTIFICATE_STILL_ACTIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

A CA certificate is marked as revoked in the certificate revocation list maintained by the issuing authority, but is still marked as ACTIVE or PENDING_TRANSFER in AWS IoT.

The following reason codes are returned when this check finds a noncompliant CA certificate:

- CERTIFICATE_REVOKED_BY_ISSUER

Why it matters

A revoked CA certificate should no longer be used to sign device certificates. It might have been revoked because it was compromised. Newly added devices with certificates signed using this CA certificate might pose a security threat.

How to fix it

1. Use UpdateCACertificate to mark the CA certificate as INACTIVE in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_CA_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions.
2. Review the device certificate registration activity for the time after the CA certificate was revoked and consider revoking any device certificates that might have been issued with it during this time. You can use ListCertificatesByCA to list the device certificates signed by the CA certificate and UpdateCertificate to revoke a device certificate.

Device certificate shared

Multiple, concurrent connections use the same X.509 certificate to authenticate with AWS IoT.

This check appears as `DEVICE_CERTIFICATE_SHARED_CHECK` in the CLI and API.

Severity: **Critical**

Details

When performed as part of an on-demand audit, this check looks at the certificates and client IDs that were used by devices to connect during the 31 days before the start of the audit up to 2 hours before the check is run. For scheduled audits, this check looks at data from 2 hours before the last time the audit was run to 2 hours before the time this instance of the audit started. If you have taken steps to mitigate this condition during the time checked, note when the concurrent connections were made to determine if the problem persists.

The following reason codes are returned when this check finds a noncompliant certificate:

- CERTIFICATE_SHARED_BY_MULTIPLE_DEVICES

In addition, the findings returned by this check include the ID of the shared certificate, the IDs of the clients using the certificate to connect, and the connect/disconnect times. Most recent results are listed first.

Why it matters

Each device should have a unique certificate to authenticate with AWS IoT. When multiple devices use the same certificate, this might indicate that a device has been compromised. Its identity might have been cloned to further compromise the system.

How to fix it

Verify that the device certificate has not been compromised. If it has, follow your security best practices to mitigate the situation.

If you use the same certificate on multiple devices, you might want to:

1. Provision new, unique certificates and attach them to each device.
2. Verify that the new certificates are valid and the devices can use them to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to do the following:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions.
4. Detach the old certificate from each of the devices.

Device certificate key quality

AWS IoT customers often rely on TLS mutual authentication using X.509 certificates for authenticating to AWS IoT message broker. These certificates and their certificate authority

certificates must be registered in their AWS IoT account before they are used. AWS IoT performs basic sanity checks on these certificates when they are registered. These checks include:

- They must be in a valid format.
- They must be signed by a registered certificate authority.
- They must still be within their validity period (in other words, they haven't expired).
- Their cryptographic key sizes must meet a minimum required size (for RSA keys, they must be 2048 bits or larger).

This audit check provides the following additional tests of the quality of your cryptographic key:

- CVE-2008-0166 – Check whether the key was generated using OpenSSL 0.9.8c-1 up to versions before 0.9.8g-9 on a Debian-based operating system. Those versions of OpenSSL use a random number generator that generates predictable numbers, making it easier for remote attackers to conduct brute force guessing attacks against cryptographic keys.
- CVE-2017-15361 – Check whether the key was generated by the Infineon RSA library 1.02.013 in Infineon Trusted Platform Module (TPM) firmware, such as versions before 0000000000000422 – 4.34, before 000000000000062b – 6.43, and before 0000000000008521 – 133.33. That library mishandles RSA key generation, making it easier for attackers to defeat some cryptographic protection mechanisms through targeted attacks. Examples of affected technologies include BitLocker with TPM 1.2, YubiKey 4 (before 4.3.5) PGP key generation, and the Cached User Data encryption feature in Chrome OS.

AWS IoT Device Defender reports certificates as noncompliant if they fail these tests.

This check appears as `DEVICE_CERTIFICATE_KEY_QUALITY_CHECK` in the CLI and API.

Severity: **Critical**

Details

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant certificate:

- CERTIFICATE_KEY_VULNERABILITY_CVE-2017-15361
- CERTIFICATE_KEY_VULNERABILITY_CVE-2008-0166

Why it matters

When a device uses a vulnerable certificate, attackers can more easily compromise that device.

How to fix it

Update your device certificates to replace those with known vulnerabilities.

If you are using the same certificate on multiple devices, you might want to:

1. Provision new, unique certificates and attach them to each device.
2. Verify that the new certificates are valid and the devices can use them to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions.
4. Detach the old certificate from each of the devices.

CA certificate key quality

AWS IoT customers often rely on TLS mutual authentication using X.509 certificates for authenticating to AWS IoT message broker. These certificates and their certificate authority certificates must be registered in their AWS IoT account before they are used. AWS IoT performs basic sanity checks on these certificates when they are registered, including:

- The certificates are in a valid format.
- The certificates are within their validity period (in other words, not expired).
- Their cryptographic key sizes meet a minimum required size (for RSA keys, they must be 2048 bits or larger).

This audit check provides the following additional tests of the quality of your cryptographic key:

- CVE-2008-0166 – Check whether the key was generated using OpenSSL 0.9.8c-1 up to versions before 0.9.8g-9 on a Debian-based operating system. Those versions of OpenSSL use a random number generator that generates predictable numbers, making it easier for remote attackers to conduct brute force guessing attacks against cryptographic keys.
- CVE-2017-15361 – Check whether the key was generated by the Infineon RSA library 1.02.013 in Infineon Trusted Platform Module (TPM) firmware, such as versions before 0000000000000422 – 4.34, before 000000000000062b – 6.43, and before 0000000000008521 – 133.33. That library mishandles RSA key generation, making it easier for attackers to defeat some cryptographic protection mechanisms through targeted attacks. Examples of affected technologies include BitLocker with TPM 1.2, YubiKey 4 (before 4.3.5) PGP key generation, and the Cached User Data encryption feature in Chrome OS.

AWS IoT Device Defender reports certificates as noncompliant if they fail these tests.

This check appears as `CA_CERTIFICATE_KEY_QUALITY_CHECK` in the CLI and API.

Severity: **Critical**

Details

This check applies to CA certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant certificate:

- CERTIFICATE_KEY_VULNERABILITY_CVE-2017-15361
- CERTIFICATE_KEY_VULNERABILITY_CVE-2008-0166

Why it matters

Newly added devices signed using this CA certificate might pose a security threat.

How to fix it

1. Use UpdateCACertificate to mark the CA certificate as INACTIVE in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_CA_CERTIFICATE` mitigation action on your audit findings to make this change.

- Apply the PUBLISH_FINDINGS_TO_SNS mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

2. Review the device certificate registration activity for the time after the CA certificate was revoked and consider revoking any device certificates that might have been issued with it during this time. (Use ListCertificatesByCA to list the device certificates signed by the CA certificate and UpdateCertificate to revoke a device certificate.)

Unauthenticated Cognito role overly permissive

A policy attached to an unauthenticated Amazon Cognito identity pool role is considered too permissive because it grants permission to perform any of the following AWS IoT actions:

- Manage or modify things.
- Read thing administrative data.
- Manage non-thing related data or resources.

Or, because it grants permission to perform the following AWS IoT actions on a broad set of devices:

- Use MQTT to connect, publish, or subscribe to reserved topics (including shadow or job execution data).
- Use API commands to read or modify shadow or job execution data.

In general, devices that connect using an unauthenticated Amazon Cognito identity pool role should have only limited permission to publish and subscribe to thing-specific MQTT topics or use the API commands to read and modify thing-specific data related to shadow or job execution data.

This check appears as UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK in the CLI and API.

Severity: **Critical**

Details

For this check, AWS IoT Device Defender audits all Amazon Cognito identity pools that have been used to connect to the AWS IoT message broker during the 31 days before the audit execution. All Amazon Cognito identity pools from which either an authenticated or unauthenticated Amazon Cognito identity connected are included in the audit.

The following reason codes are returned when this check finds a noncompliant unauthenticated Amazon Cognito identity pool role:

- ALLOWS_ACCESS_TO_IOT_ADMIN_ACTIONS
- ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS

Why it matters

Because unauthenticated identities are never authenticated by the user, they pose a much greater risk than authenticated Amazon Cognito identities. If an unauthenticated identity is compromised, it can use administrative actions to modify account settings, delete resources, or gain access to sensitive data. Or, with broad access to device settings, it can access or modify shadows and jobs for all devices in your account. A guest user might use the permissions to compromise your entire fleet or launch a DDOS attack with messages.

How to fix it

A policy attached to an unauthenticated Amazon Cognito identity pool role should grant only those permissions required for a device to do its job. We recommend the following steps:

1. Create a new compliant role.
2. Create a Amazon Cognito identity pool and attach the compliant role to it.
3. Verify that your identities can access AWS IoT using the new pool.
4. After verification is complete, attach the compliant role to the Amazon Cognito identity pool that was flagged as noncompliant.

You can also use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

Manage or modify things

The following AWS IoT API actions are used to manage or modify things. Permission to perform these actions should not be granted to devices that connect through an unauthenticated Amazon Cognito identity pool.

- `AddThingToThingGroup`
- `AttachThingPrincipal`
- `CreateThing`
- `DeleteThing`
- `DetachThingPrincipal`
- `ListThings`
- `ListThingsInThingGroup`
- `RegisterThing`
- `RemoveThingFromThingGroup`
- `UpdateThing`
- `UpdateThingGroupsForThing`

Any role that grants permission to perform these actions on even a single resource is considered noncompliant.

Read thing administrative data

The following AWS IoT API actions are used to read or modify thing data. Devices that connect through an unauthenticated Amazon Cognito identity pool should not be given permission to perform these actions.

- `DescribeThing`
- `ListJobExecutionsForThing`
- `ListThingGroupsForThing`
- `ListThingPrincipals`

Example

- noncompliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowIoTThingOperations",
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/name-of-thing"
      ]
    }
  ]
}
```

  This allows the device to perform the specified actions even though it is granted for one thing only.

Manage non-things

Devices that connect through an unauthenticated Amazon Cognito identity pool should not be given permission to perform AWS IoT API actions other than those discussed in these sections. You can manage your account with an application that connects through an unauthenticated Amazon Cognito identity pool by creating a separate identity pool not used by devices.

Subscribe/publish to MQTT topics

MQTT messages are sent through the AWS IoT message broker and are used by devices to perform many actions, including accessing and modifying shadow state and job execution state. A policy that grants permission to a device to connect, publish, or subscribe to MQTT messages should restrict these actions to specific resources as follows:

Connect

- noncompliant:

```
arn:aws:iot:region:account-id:client/*
```

  The wildcard * allows any device to connect to AWS IoT.

```
arn:aws:iot:region:account-id:client/${iot:ClientId}
```

  Unless `iot:Connection.Thing.IsAttached` is set to true in the condition keys, this is equivalent to the wildcard * in the previous example.

- compliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
      ]
    }
  ]
}
```

  The resource specification contains a variable that matches the device name used to connect. The condition statement further restricts the permission by checking that the certificate used by the MQTT client matches that attached to the thing with the name used.

Publish

- noncompliant:

```
arn:aws:iot:region:account-id:topic/$aws/things/*/shadow/update
```

This allows the device to update the shadow of any device (* = all devices).

```
arn:aws:iot:region:account-id:topic/$aws/things/*
```

This allows the device to read, update, or delete the shadow of any device.

- compliant:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
            ]
        }
    ]
}
```

  The resource specification contains a wildcard, but it only matches any shadow-related topic for the device whose thing name is used to connect.

Subscribe

- noncompliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

  This allows the device to subscribe to reserved shadow or job topics for all devices.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

  The same as the previous example, but using the # wildcard.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/+/shadow/update
```

This allows the device to see shadow updates on any device (+ = all devices).

- compliant:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

  The resource specifications contain wildcards, but they only match any shadow-related topic and any job-related topic for the device whose thing name is used to connect.

Receive

- compliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

  This is allowed because the device can receive messages only from topics on which it has permission to subscribe.

Read/modify shadow or job data

A policy that grants permission to a device to perform an API action to access or modify device shadows or job execution data should restrict these actions to specific resources. The following are the API actions:

- `DeleteThingShadow`
- `GetThingShadow`
- `UpdateThingShadow`
- `DescribeJobExecution`
- `GetPendingJobExecutions`
- `StartNextPendingJobExecution`
- `UpdateJobExecution`

Example

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

  This allows the device to perform the specified action on any thing.

- compliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DeleteThingShadow",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iotjobsdata:DescribeJobExecution",
          "iotjobsdata:GetPendingJobExecutions",
          "iotjobsdata:StartNextPendingJobExecution",
          "iotjobsdata:UpdateJobExecution"
```

```
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing1",
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing2"
      ]
    }
  ]
}
```

This allows the device to perform the specified actions on two things only.

Authenticated Cognito role overly permissive

A policy attached to an authenticated Amazon Cognito identity pool role is considered too permissive because it grants permission to perform the following AWS IoT actions:

- Manage or modify things.
- Manage non-thing related data or resources.

Or, because it grants permission to perform the following AWS IoT actions on a broad set of devices:

- Read thing administrative data.
- Use MQTT to connect/publish/subscribe to reserved topics (including shadow or job execution data).
- Use API commands to read or modify shadow or job execution data.

In general, devices that connect using an authenticated Amazon Cognito identity pool role should have only limited permission to read thing-specific administrative data, publish and subscribe to thing-specific MQTT topics, or use the API commands to read and modify thing-specific data related to shadow or job execution data.

This check appears as `AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

For this check, AWS IoT Device Defender audits all Amazon Cognito identity pools that have been used to connect to the AWS IoT message broker during the 31 days before the audit execution. All Amazon Cognito identity pools from which either an authenticated or unauthenticated Amazon Cognito identity connected are included in the audit.

The following reason codes are returned when this check finds a noncompliant authenticated Amazon Cognito identity pool role:

- ALLOWS_BROAD_ACCESS_TO_IOT_THING_ADMIN_READ_ACTIONS
- ALLOWS_ACCESS_TO_IOT_NON_THING_ADMIN_ACTIONS
- ALLOWS_ACCESS_TO_IOT_THING_ADMIN_WRITE_ACTIONS

Why it matters

If an authenticated identity is compromised, it can use administrative actions to modify account settings, delete resources, or gain access to sensitive data.

How to fix it

A policy attached to an authenticated Amazon Cognito identity pool role should grant only those permissions required for a device to do its job. We recommend the following steps:

1. Create a new compliant role.
2. Create a Amazon Cognito identity pool and attach the compliant role to it.
3. Verify that your identities can access AWS IoT using the new pool.
4. After verification is complete, attach the role to the Amazon Cognito identity pool that was flagged as noncompliant.

You can also use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

Manage or modify things

The following AWS IoT API actions are used to manage or modify things so permission to perform these should not be granted to devices connecting through an authenticated Amazon Cognito identity pool:

- `AddThingToThingGroup`
- `AttachThingPrincipal`
- `CreateThing`
- `DeleteThing`
- `DetachThingPrincipal`
- `ListThings`
- `ListThingsInThingGroup`
- `RegisterThing`
- `RemoveThingFromThingGroup`
- `UpdateThing`
- `UpdateThingGroupsForThing`

Any role that grants permission to perform these actions on even a single resource is considered noncompliant.

Manage non-things

Devices that connect through an authenticated Amazon Cognito identity pool should not be given permission to perform AWS IoT API actions other than those discussed in these sections. To manage your account with an application that connects through an authenticated Amazon Cognito identity pool, create a separate identity pool not used by devices.

Read thing administrative data

The following AWS IoT API actions are used to read thing data, so devices that connect through an authenticated Amazon Cognito identity pool should be given permission to perform these on a limited set of things only:

- `DescribeThing`

- `ListJobExecutionsForThing`
- `ListThingGroupsForThing`
- `ListThingPrincipals`

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

  This allows the device to perform the specified action on any thing.

- compliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing"
      ]
    }
  ]
}
```

  This allows the device to perform the specified actions on only one thing.

- compliant:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:DescribeThing",
                "iot:ListJobExecutionsForThing",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/MyThing*"
            ]
        }
    ]
}
```

This is compliant because, although the resource is specified with a wildcard (*), it is preceded by a specific string, and that limits the set of things accessed to those with names that have the given prefix.

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

This allows the device to perform the specified action on any thing.

- compliant:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing"
      ]
```

```
    }
  ]
}
```

This allows the device to perform the specified actions on only one thing.

- compliant:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeThing",
                "iot:ListJobExecutionsForThing",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/MyThing*"
            ]
        }
    ]
}
```

This is compliant because, although the resource is specified with a wildcard (*), it is preceded by a specific string, and that limits the set of things accessed to those with names that have the given prefix.

Subscribe/publish to MQTT topics

MQTT messages are sent through the AWS IoT message broker and are used by devices to perform many different actions, including accessing and modifying shadow state and job execution state. A policy that grants permission to a device to connect, publish, or subscribe to MQTT messages should restrict these actions to specific resources as follows:

Connect

- noncompliant:

```
arn:aws:iot:region:account-id:client/*
```

The wildcard * allows any device to connect to AWS IoT.

```
arn:aws:iot:region:account-id:client/${iot:ClientId}
```

Unless `iot:Connection.Thing.IsAttached` is set to true in the condition keys, this is equivalent to the wildcard * in the previous example.

- compliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
      ]
    }
  ]
}
```

  The resource specification contains a variable that matches the device name used to connect, and the condition statement further restricts the permission by checking that the certificate used by the MQTT client matches that attached to the thing with the name used.

Publish

- noncompliant:

```
arn:aws:iot:region:account-id:topic/$aws/things/*/shadow/update
```

  This allows the device to update the shadow of any device (* = all devices).

```
arn:aws:iot:region:account-id:topic/$aws/things/*
```

This allows the device to read/update/delete the shadow of any device.

- compliant:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
            ]
        }
    ]
}
```

The resource specification contains a wildcard, but it only matches any shadow-related topic for the device whose thing name is used to connect.

Subscribe

- noncompliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

This allows the device to subscribe to reserved shadow or job topics for all devices.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/#
```

The same as the previous example, but using the # wildcard.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/+/shadow/update
```

This allows the device to see shadow updates on any device (+ = all devices).

- compliant:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

  The resource specifications contain wildcards, but they only match any shadow-related topic and any job-related topic for the device whose thing name is used to connect.

Receive

- compliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

  This is compliant because the device can receive messages only from topics on which it has permission to subscribe.

Read or modify shadow or job data

A policy that grants permission to a device to perform an API action to access or modify device shadows or job execution data should restrict these actions to specific resources. The following are the API actions:

- `DeleteThingShadow`
- `GetThingShadow`
- `UpdateThingShadow`
- `DescribeJobExecution`
- `GetPendingJobExecutions`
- `StartNextPendingJobExecution`
- `UpdateJobExecution`

Examples

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

  This allows the device to perform the specified action on any thing.

- compliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DeleteThingShadow",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DescribeJobExecution",
        "iotjobsdata:DescribeJobExecution",
        "iotjobsdata:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing1",
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing2"
      ]
    }
  ]
```

```
}
```

This allows the device to perform the specified actions on only two things.

AWS IoT policies overly permissive

An AWS IoT policy gives permissions that are too broad or unrestricted. It grants permission to send or receive MQTT messages for a broad set of devices, or grants permission to access or modify shadow and job execution data for a broad set of devices.

In general, a policy for a device should grant access to resources associated with just that device and no or very few other devices. With some exceptions, using a wildcard (for example, "*") to specify resources in such a policy is considered too broad or unrestricted.

This check appears as `IOT_POLICY_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

The following reason code is returned when this check finds a noncompliant AWS IoT policy:

- ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS

Why it matters

A certificate, Amazon Cognito identity, or thing group with an overly permissive policy can, if compromised, impact the security of your entire account. An attacker could use such broad access to read or modify shadows, jobs, or job executions for all your devices. Or an attacker could use a compromised certificate to connect malicious devices or launch a DDOS attack on your network.

How to fix it

Follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Use CreatePolicyVersion to create a new, compliant version of the policy. Set the `setAsDefault` flag to true. (This makes this new version operative for all entities that use the policy.)

2. Use ListTargetsForPolicy to get a list of targets (certificates, thing groups) that the policy is attached to and determine which devices are included in the groups or which use the certificates to connect.
3. Verify that all associated devices are able to connect to AWS IoT. If a device is unable to connect, use SetPolicyVersion to roll back the default policy to the previous version, revise the policy, and try again.

You can use mitigation actions to:

- Apply the `REPLACE_DEFAULT_POLICY_VERSION` mitigation action on your audit findings to make this change.
- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

Use AWS IoT Core policy variables to dynamically reference AWS IoT resources in your policies.

MQTT permissions

MQTT messages are sent through the AWS IoT message broker and are used by devices to perform many actions, including accessing and modifying shadow state and job execution state. A policy that grants permission to a device to connect, publish, or subscribe to MQTT messages should restrict these actions to specific resources as follows:

Connect

- noncompliant:

  ```
  arn:aws:iot:region:account-id:client/*
  ```

  The wildcard * allows any device to connect to AWS IoT.

  ```
  arn:aws:iot:region:account-id:client/${iot:ClientId}
  ```

  Unless `iot:Connection.Thing.IsAttached` is set to true in the condition keys, this is equivalent to the wildcard * as in the previous example.
- compliant:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
      ]
    }
  ]
}
```

The resource specification contains a variable that matches the device name used to connect. The condition statement further restricts the permission by checking that the certificate used by the MQTT client matches that attached to the thing with the name used.

Publish

- noncompliant:

  ```
  arn:aws:iot:region:account-id:topic/$aws/things/*/shadow/update
  ```

  This allows the device to update the shadow of any device (* = all devices).

  ```
  arn:aws:iot:region:account-id:topic/$aws/things/*
  ```

  This allows the device to read, update, or delete the shadow of any device.

- compliant:

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
  ```

```
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
            ]
        }
    ]
}
```

The resource specification contains a wildcard, but it only matches any shadow-related topic for the device whose thing name is used to connect.

Subscribe

- noncompliant:

  ```
  arn:aws:iot:region:account-id:topicfilter/$aws/things/*
  ```

  This allows the device to subscribe to reserved shadow or job topics for all devices.

  ```
  arn:aws:iot:region:account-id:topicfilter/$aws/things/*
  ```

  The same as the previous example, but using the # wildcard.

  ```
  arn:aws:iot:region:account-id:topicfilter/$aws/things/+/shadow/update
  ```

  This allows the device to see shadow updates on any device (+ = all devices).

- compliant:

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "iot:Subscribe"
              ],
  ```

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

The resource specifications contain wildcards, but they only match any shadow-related topic and any job-related topic for the device whose thing name is used to connect.

Receive

- compliant:

  ```
  arn:aws:iot:region:account-id:topic/$aws/things/*
  ```

  This is compliant because the device can only receive messages from topics on which it has permission to subscribe.

Shadow and job permissions

A policy that grants permission to a device to perform an API action to access or modify device shadows or job execution data should restrict these actions to specific resources. The following are the API actions:

- `DeleteThingShadow`
- `GetThingShadow`
- `UpdateThingShadow`
- `DescribeJobExecution`
- `GetPendingJobExecutions`
- `StartNextPendingJobExecution`
- `UpdateJobExecution`

Examples

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

  This allows the device to perform the specified action on any thing.

- compliant:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DeleteThingShadow",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iotjobsdata:DescribeJobExecution",
          "iotjobsdata:GetPendingJobExecutions",
          "iotjobsdata:StartNextPendingJobExecution",
          "iotjobsdata:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing1",
        "arn:aws:iot:us-east-1:123456789012:thing/MyThing2"
      ]
    }
  ]
}
```

  This allows the device to perform the specified actions on only two things.

AWS IoT policy potentially misconfigured

An AWS IoT policy was identified as potentially misconfigured. Misconfigured policies, including overly permissive policies, can cause security incidents like allowing devices access to unintended resources.

The **AWS IoT policy potentially misconfigured** check is a warning for you to make sure that only intended actions are allowed before updating the policy.

In the CLI and API, this check appears as `IOT_POLICY_POTENTIAL_MISCONFIGURATION_CHECK`.

Severity: **Medium**

Details

AWS IoT returns the following reason code when this check finds a potentially misconfigured AWS IoT policy:

- POLICY_CONTAINS_MQTT_WILDCARDS_IN_DENY_STATEMENT
- TOPIC_FILTERS_INTENDED_TO_DENY_ALLOWED_USING_WILDCARDS

Why it matters

Misconfigured policies can lead to unintended consequences by providing more permissions to devices than required. We recommend careful consideration of the policy to limit access to resources and prevent security threats.

Policy contains MQTT wildcards in deny statement example

The **AWS IoT policy potentially misconfigured** check inspects for MQTT wildcard characters (+ or #) in deny statements. Wildcards are treated as literal strings by AWS IoT policies and can make the policy overly permissive.

The following example is intended to deny subscribing to topics related to `building/control_room` by using the MQTT wildcard # in policies. However, MQTT wildcards don't have a wildcard meaning in AWS IoT policies and devices can subscribe to `building/control_room/data1`.

The **AWS IoT policy potentially misconfigured** check will flag this policy with reason code `POLICY_CONTAINS_MQTT_WILDCARDS_IN_DENY_STATEMENT`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
```

```
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
*"
        },
        {
            "Effect": "Deny",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
control_room/#"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/building/*"
        }
    ]
}
```

The following is an example of a properly configured policy. Devices don't have permission to subscribe to subtopics of `building/control_room/` and don't have permissions to receive messages from subtopics of `building/control_room/`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/*"
    },
    {
      "Effect": "Deny",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
control_room/*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/building/*"
```

```
    },
    {
      "Effect": "Deny",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/building/
control_room/*"
    }
  ]
}
```

Topic filters intended to deny allowed using wildcards example

The following example policy is intended to deny subscribing to topics related to `building/control_room` by denying the resource `building/control_room/*`. However, devices can send requests to subscribe to `building/#` and receive messages from all topics related to `building`, including `building/control_room/data1`.

The **AWS IoT policy potentially misconfigured** check will flag this policy with reason code `TOPIC_FILTERS_INTENDED_TO_DENY_ALLOWED_USING_WILDCARDS`.

The following example policy has permissions to receive message on `building/control_room topics`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
*"
        },
        {
            "Effect": "Deny",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
control_room/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/building/*"
        }
    ]
}
```

The following is an example of a properly configured policy. Devices don't have permission to subscribe to subtopics of `building/control_room/` and don't have permissions to receive messages from subtopics of `building/control_room/`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
*"
        },
        {
            "Effect": "Deny",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/building/
control_room/*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/building/*"
        },
        {
            "Effect": "Deny",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/building/
control_room/*"
        }
    ]
}
```

> **Note**
>
> This check might report false positives. We recommend that you evaluate any flagged policies and mark false positives resources using audit suppressions.

How to fix it

This check flags potentially misconfigured policies so there might be false positives. Mark any false positives using audit suppressions so they aren't flagged in the future.

You can also follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Use CreatePolicyVersion to create a new, compliant version of the policy. Set the `setAsDefault` flag to true. (This makes this new version operative for all entities that use the policy.)

   For examples of creating AWS IoT policies for common use cases, see Publish/Subscribe policy examples in the *AWS IoT Core Developer Guide*.
2. Verify that all associated devices are able to connect to AWS IoT. If a device is unable to connect, use SetPolicyVersion to roll back the default policy to the previous version, revise the policy, and try again.

You can use mitigation actions to:

- Apply the `REPLACE_DEFAULT_POLICY_VERSION` mitigation action on your audit findings to make this change.
- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

Use IoT Core policy variables in the *AWS IoT Core Developer Guide* to dynamically reference AWS IoT resources in your policies.

Role alias overly permissive

AWS IoT role alias provides a mechanism for connected devices to authenticate to AWS IoT using X.509 certificates and then obtain short-lived AWS credentials from an IAM role that is associated with an AWS IoT role alias. The permissions for these credentials must be scoped down using access policies with authentication context variables. If your policies are not configured correctly, you could leave yourself exposed to an escalation of privilege attack. This audit check ensures that the temporary credentials provided by AWS IoT role aliases are not overly permissive.

This check is triggered if one of the following conditions are found:

- The policy provides administrative permissions to any services used in the past year by this role alias (for example, "iot:*", "dynamodb:*", "iam:*", and so on).
- The policy provides broad access to thing metadata actions, access to restricted AWS IoT actions, or broad access to AWS IoT data plane actions.
- The policy provides access to security auditing services such as "iam", "cloudtrail", "guardduty", "inspector", or "trustedadvisor".

This check appears as `IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

The following reason codes are returned when this check finds a noncompliant IoT policy:

- ALLOWS_BROAD_ACCESS_TO_USED_SERVICES
- ALLOWS_ACCESS_TO_SECURITY_AUDITING_SERVICES
- ALLOWS_BROAD_ACCESS_TO_IOT_THING_ADMIN_READ_ACTIONS
- ALLOWS_ACCESS_TO_IOT_NON_THING_ADMIN_ACTIONS
- ALLOWS_ACCESS_TO_IOT_THING_ADMIN_WRITE_ACTIONS
- ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS

Why it matters

By limiting permissions to those that are required for a device to perform its normal operations, you reduce the risks to your account if a device is compromised.

How to fix it

Follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Follow the steps in Authorizing direct calls to AWS services using AWS IoT Core credential provider to apply a more restrictive policy to your role alias.

You can use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom action in response to the Amazon SNS message.

For more information, see Mitigation actions.

Role alias allows access to unused services

AWS IoT role alias provides a mechanism for connected devices to authenticate to AWS IoT using X.509 certificates and then obtain short-lived AWS credentials from an IAM role that is associated with an AWS IoT role alias. The permissions for these credentials must be scoped down using access policies with authentication context variables. If your policies are not configured correctly, you could leave yourself exposed to an escalation of privilege attack. This audit check ensures that the temporary credentials provided by AWS IoT role aliases are not overly permissive.

This check is triggered if the role alias has access to services that haven't been used for the AWS IoT device in the last year. For example, the audit reports if you have an IAM role linked to the role alias that has only used AWS IoT in the past year but the policy attached to the role also grants permission to `"iam:getRole"` and `"dynamodb:PutItem"`.

This check appears as `IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK` in the CLI and API.

Severity: **Medium**

Details

The following reason codes are returned when this check finds a noncompliant AWS IoT policy:

- ALLOWS_ACCESS_TO_UNUSED_SERVICES

Why it matters

By limiting permissions to those services that are required for a device to perform its normal operations, you reduce the risks to your account if a device is compromised.

How to fix it

Follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Follow the steps in Authorizing direct calls to AWS services using AWS IoT Core credential provider to apply a more restrictive policy to your role alias.

You can use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom action in response to the Amazon SNS message.

For more information, see Mitigation actions.

CA certificate expiring

A CA certificate is expiring within 30 days or has expired.

This check appears as `CA_CERTIFICATE_EXPIRING_CHECK` in the CLI and API.

Severity: **Medium**

Details

This check applies to CA certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant CA certificate:

- CERTIFICATE_APPROACHING_EXPIRATION
- CERTIFICATE_PAST_EXPIRATION

Why it matters

An expired CA certificate should not be used to sign new device certificates.

How to fix it

Consult your security best practices for how to proceed. You might want to:

1. Register a new CA certificate with AWS IoT.
2. Verify that you are able to sign device certificates using the new CA certificate.
3. Use UpdateCACertificate to mark the old CA certificate as INACTIVE in AWS IoT. You can also use mitigation actions to do the following:
   - Apply the `UPDATE_CA_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions.

Conflicting MQTT client IDs

Multiple devices connect using the same client ID.

This check appears as `CONFLICTING_CLIENT_IDS_CHECK` in the CLI and API.

Severity: **High**

Details

Multiple connections were made using the same client ID, causing an already connected device to be disconnected. The MQTT specification allows only one active connection per client ID, so when another device connects using the same client ID, it knocks the previous one off the connection.

When performed as part of an on-demand audit, this check looks at how client IDs were used to connect during the 31 days prior to the start of the audit. For scheduled audits, this check looks at data from the last time the audit was run to the time this instance of the audit started. If you have taken steps to mitigate this condition during the time checked, note when the connections/disconnections were made to determine if the problem persists.

The following reason codes are returned when this check finds noncompliance:

- DUPLICATE_CLIENT_ID_ACROSS_CONNECTIONS

The findings returned by this check also include the client ID used to connect, principal IDs, and disconnect times. The most recent results are listed first.

Why it matters

Devices with conflicting IDs are forced to constantly reconnect, which might result in lost messages or leave a device unable to connect.

This might indicate that a device or a device's credentials have been compromised, and might be part of a DDoS attack. It is also possible that devices are not configured correctly in the account or a device has a bad connection and is forced to reconnect several times per minute.

How to fix it

Register each device as a unique thing in AWS IoT, and use the thing name as the client ID to connect. Or use a UUID as the client ID when connecting the device over MQTT. You can also use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

Device certificate expiring

A device certificate is expiring within the configured threshold period or has expired. The certificate expiration check threshold can be configured between 30 days (minimum) and 3652 days (10 years, maximum) with a default value of 30 days.

This check appears as `DEVICE_CERTIFICATE_EXPIRING_CHECK` in the CLI and API.

Severity: **Medium**

Details

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant device certificate:

- CERTIFICATE_APPROACHING_EXPIRATION
- CERTIFICATE_PAST_EXPIRATION

Why it matters

A device certificate should not be used after it expires.

Configuring the Device certificate expiring check

This configuration enables you to monitor and receive alerts for certificates that are approaching their expiration date across your device fleet. For example, if you want to be notified when certificates are within 30 days of expiration, you can configure the check as follows:

```
{
    "roleArn": "your-audit-role-arn",
    "auditCheckConfigurations": {
        "DEVICE_CERTIFICATE_EXPIRING_CHECK": {
            "enabled": true,
            "configuration": {
                "CERT_EXPIRATION_THRESHOLD_IN_DAYS": "30"
            }
        }
    }
}
```

How to fix it

Consult your security best practices for how to proceed. You might want to:

1. Provision a new certificate and attach it to the device.
2. Verify that the new certificate is valid and the device is able to use it to connect.
3. Use UpdateCertificate to mark the old certificate as INACTIVE in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions.

4. Detach the old certificate from the device. (See DetachThingPrincipal.)

Device certificate age check

This audit check alerts you when a device certificate has been active for a number of days greater than or equal to the number you specify. This check helps you stay informed about your certificates' status, enabling timely action on a periodic basis, regardless of when the certificate reaches the end of its lifespan, improving security by reducing the risk of certificate compromise.

The certificate age check threshold can be configured between 30 days (minimum) and 3652 days (10 years, maximum), with a default value of 365 days.

This check appears as `DEVICE_CERTIFICATE_AGE_CHECK` in the CLI and API. This check is disabled by default Severity: **Low**

Details

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER. The following reason codes are returned when this check finds a noncompliant device certificate:

- CERTIFICATE_PAST_AGE_THRESHOLD

Configuring the device certificate age check

This configuration allows you to tailor certificate rotation alerts to the specific needs of your fleet, helping you maintain a strong security posture across all devices. You can configure this check using the `UpdateAccountAuditConfiguration` API. For example, if you want to be alerted when certificates have been active for more than 365 days, you can configure the check as follows:

```
{
    "roleArn": "your-audit-role-arn",
    "auditCheckConfigurations": {
        "DEVICE_CERTIFICATE_AGE_CHECK": {
            "enabled": true,
            "configuration": {
                "CERT_AGE_THRESHOLD_IN_DAYS": "365"
            }
        }
```

```
    }
}
```

Revoked device certificate still active

A revoked device certificate is still active.

This check appears as `REVOKED_DEVICE_CERTIFICATE_STILL_ACTIVE_CHECK` in the CLI and API.

Severity: **Medium**

Details

A device certificate is in its CA's certificate revocation list, but it is still active in AWS IoT.

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds noncompliance:

- CERTIFICATE_REVOKED_BY_ISSUER

Why it matters

A device certificate is usually revoked because it has been compromised. It is possible that it has not yet been revoked in AWS IoT due to an error or oversight.

How to fix it

Verify that the device certificate has not been compromised. If it has, follow your security best practices to mitigate the situation. You might want to:

1. Provision a new certificate for the device.
2. Verify that the new certificate is valid and the device is able to use it to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.

- Apply the ADD_THINGS_TO_THING_GROUP mitigation action to add the device to a group where you can take action on it.
- Apply the PUBLISH_FINDINGS_TO_SNS mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

4. Detach the old certificate from the device. (See DetachThingPrincipal.)

Logging disabled

AWS IoT logs are not enabled in Amazon CloudWatch. Verifies both V1 and V2 logging.

This check appears as LOGGING_DISABLED_CHECK in the CLI and API.

Severity: **Low**

Details

The following reason codes are returned when this check finds noncompliance:

- LOGGING_DISABLED

Why it matters

AWS IoT logs in CloudWatch provide visibility into behaviors in AWS IoT, including authentication failures and unexpected connects and disconnects that might indicate that a device has been compromised.

How to fix it

Enable AWS IoT logs in CloudWatch. See Logging and Monitoring in the *AWS IoT Core Developer Guide*. You can also use mitigation actions to:

- Apply the ENABLE_IOT_LOGGING mitigation action on your audit findings to make this change.
- Apply the PUBLISH_FINDINGS_TO_SNS mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions.

Audit commands

Manage audit settings

Use `UpdateAccountAuditConfiguration` to configure audit settings for your account. This command allows you to enable those checks you want to be available for audits, set up optional notifications, and configure permissions.

Check these settings with `DescribeAccountAuditConfiguration`.

Use `DeleteAccountAuditConfiguration` to delete your audit settings. This restores all default values, and effectively disables audits because all checks are disabled by default.

UpdateAccountAuditConfiguration

Configures or reconfigures the Device Defender audit settings for this account. Settings include how audit notifications are sent and which audit checks are enabled or disabled.

Synopsis

```
aws iot  update-account-audit-configuration \
    [--role-arn <value>] \
    [--audit-notification-target-configurations <value>] \
    [--audit-check-configurations <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "roleArn": "string",
  "auditNotificationTargetConfigurations": {
    "string": {
      "targetArn": "string",
      "roleArn": "string",
      "enabled": "boolean"
    }
  },
  "auditCheckConfigurations": {
    "string": {
      "enabled": "boolean"
    }
```

```
  }
}
```

`cli-input-json` Fields

Name	Type	Description
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to AWS IoT to access information about your devices, policies, certifica tes, and other items when performing an audit.
auditNotificationTargetConf igurations	map	Information about the targets to which audit notifications are sent.
targetArn	string	The ARN of the target (SNS topic) to which audit notificat ions are sent.
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to send notifications to the target.
enabled	boolean	True if notifications to the target are enabled.
auditCheckConfigurations	map	Specifies which audit checks are enabled and disabled for this account. Use `DescribeA ccountAuditConfigu ration` to see the list of all checks, including those that are currently enabled. Some data collection might start immediately when

Name	Type	Description
		certain checks are enabled. When a check is disabled, any data collected so far in relation to the check is deleted. You cannot disable a check if it is used by any scheduled audit. You must first delete the check from the scheduled audit or delete the scheduled audit itself. On the first call to `UpdateAccountAuditConfiguration`, this parameter is required and must specify at least one enabled check.
enabled	boolean	True if this audit check is enabled for this account.
configuration	map	(Optional) custom configurations for specific audit checks, such as the `CERT_AGE_THRESHOLD_IN_DAYS` and `CERT_EXPIRATION_THRESHOLD_IN_DAYS`, allowing you to define when you want to be alerted about certificate age and impending expiration.

Output

None

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DescribeAccountAuditConfiguration

Gets information about the Device Defender audit settings for this account. Settings include how audit notifications are sent and which audit checks are enabled or disabled.

Synopsis

```
aws iot  describe-account-audit-configuration  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
}
```

Output

```
{
  "roleArn": "string",
  "auditNotificationTargetConfigurations": {
    "string": {
      "targetArn": "string",
      "roleArn": "string",
      "enabled": "boolean"
```

```
    }
  },
  "auditCheckConfigurations": {
    "string": {
      "enabled": "boolean"
    }
  }
}
```

CLI output fields

Name	Type	Description
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to AWS IoT to access information about your devices, policies, certificates, and other items when performing an audit. On the first call to `UpdateAccountAuditConfiguration`, this parameter is required.
auditNotificationTargetConfigurations	map	Information about the targets to which audit notifications are sent for this account.
targetArn	string	The ARN of the target (SNS topic) to which audit notifications are sent.
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to send notifications to the target.
enabled	boolean	True if notifications to the target are enabled.

Name	Type	Description
auditCheckConfigurations	map	Which audit checks are enabled and disabled for this account.
enabled	boolean	True if this audit check is enabled for this account.
configuration	map	(Optional) provides specific configurations for certain audit checks, such as the maximum allowed age for certificates or the number of days before expiration when an alert should be triggered.

Errors

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DeleteAccountAuditConfiguration

Restores the default settings for Device Defender audits for this account. Any configuration data you entered is deleted and all audit checks are reset to disabled.

Synopsis

```
aws iot  delete-account-audit-configuration \
    [--delete-scheduled-audits | --no-delete-scheduled-audits]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "deleteScheduledAudits": "boolean"
}
```

`cli-input-json` Fields

Name	Type	Description
deleteScheduledAudits	boolean	If true, all scheduled audits are deleted.

Output

None

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Schedule audits

Use `CreateScheduledAudit` to create one or more scheduled audits. This command allows you to specify the checks you want to perform during an audit and how often the audit should be run.

Keep track of your scheduled audits with `ListScheduledAudits` and `DescribeScheduledAudit`.

Change an existing scheduled audit with `UpdateScheduledAudit` or delete it with `DeleteScheduledAudit`.

CreateScheduledAudit

Creates a scheduled audit that is run at a specified time interval.

Synopsis

```
aws iot  create-scheduled-audit \
    --frequency <value> \
    [--day-of-month <value>] \
    [--day-of-week <value>] \
    --target-check-names <value> \
    [--tags <value>] \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "frequency": "string",
  "dayOfMonth": "string",
  "dayOfWeek": "string",
  "targetCheckNames": [
    "string"
  ],
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
frequency	string	How often the scheduled audit takes place. Can be one of DAILY, WEEKLY, BIWEEKLY, or MONTHLY. The actual

Name	Type	Description
		start time of each audit is determined by the system. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit takes place. Can be 1 through 31 or LAST. This field is required if the `frequency` parameter is set to MONTHLY. If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit takes place. Can be one of SUN, MON, TUE,WED, THU, FRI, or SAT. This field is required if the `frequency` parameter is set to WEEKLY or BIWEEKLY. enum: SUN | MON | TUE | WED | THU | FRI | SAT

Name	Type	Description
targetCheckNames	list member: AuditCheckName	Which checks are performed during the scheduled audit. Checks must be enabled for your account. (Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or `UpdateAccountAuditConfiguration` to select which checks are enabled.)
tags	list member: Tag java class: java.util.List	Metadata that can be used to manage the scheduled audit.
Key	string	The tag's key.
Value	string	The tag's value.
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name you want to give to the scheduled audit. (Maximum of 128 characters)

Output

```
{
  "scheduledAuditArn": "string"
}
```

CLI output fields

Name	Type	Description
scheduledAuditArn	string	The ARN of the scheduled audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

`LimitExceededException`

A limit has been exceeded.

ListScheduledAudits

Lists all of your scheduled audits.

Synopsis

```
aws iot  list-scheduled-audits \
    [--next-token <value>] \
    [--max-results <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "nextToken": "string",
  "maxResults": "integer"
}
```

`cli-input-json` Fields

Name	Type	Description
nextToken	string	The token for the next set of results.
maxResults	integer range- max:250 min:1	The maximum number of results to return at one time. The default is 25.

Output

```
{
  "scheduledAudits": [
    {
      "scheduledAuditName": "string",
      "scheduledAuditArn": "string",
      "frequency": "string",
      "dayOfMonth": "string",
      "dayOfWeek": "string"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Type	Description
scheduledAudits	list member: Scheduled AuditMetadata java class: java.util.List	The list of scheduled audits.
scheduledAuditName	string length- max:128 min:1	The name of the scheduled audit.

Name	Type	Description
	pattern: [a-zA-Z0-9_-]+	
scheduledAuditArn	string	The ARN of the scheduled audit.
frequency	string	How often the scheduled audit takes place. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit is run (if the `frequency` is MONTHLY). If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit is run (if the `frequency` is WEEKLY or BIWEEKLY). enum: SUN | MON | TUE | WED | THU | FRI | SAT
nextToken	string	A token that can be used to retrieve the next set of results, or `null` if there are no more results.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DescribeScheduledAudit

Gets information about a scheduled audit.

Synopsis

```
aws iot  describe-scheduled-audit \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit whose information you want to get.

Output

```
{
```

```
  "frequency": "string",
  "dayOfMonth": "string",
  "dayOfWeek": "string",
  "targetCheckNames": [
    "string"
  ],
  "scheduledAuditName": "string",
  "scheduledAuditArn": "string"
}
```

CLI output fields

Name	Type	Description
frequency	string	How often the scheduled audit takes place. One of DAILY, WEEKLY, BIWEEKLY, or MONTHLY. The actual start time of each audit is determined by the system. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit takes place. Can be 1 through 31 or LAST. If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit takes place. One of SUN, MON, TUE, WED, THU, FRI, or SAT. enum: SUN | MON | TUE | WED | THU | FRI | SAT

Name	Type	Description
targetCheckNames	list member: AuditCheckName	Which checks are performed during the scheduled audit. Checks must be enabled for your account. (Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or use `UpdateAccountAuditConfiguration` to select which checks are enabled.)
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit.
scheduledAuditArn	string	The ARN of the scheduled audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

UpdateScheduledAudit

Updates a scheduled audit, including which checks are performed and how often the audit takes place.

Synopsis

```
aws iot  update-scheduled-audit \
    [--frequency <value>] \
    [--day-of-month <value>] \
    [--day-of-week <value>] \
    [--target-check-names <value>] \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "frequency": "string",
  "dayOfMonth": "string",
  "dayOfWeek": "string",
  "targetCheckNames": [
    "string"
  ],
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
frequency	string	How often the scheduled audit takes place. Can be one of DAILY, WEEKLY, BIWEEKLY, or MONTHLY. The actual start time of each audit is determined by the system. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY

Name	Type	Description
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit takes place. Can be 1 through 31 or LAST. This field is required if the `frequency` parameter is set to MONTHLY. If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit takes place. Can be one of SUN, MON, TUE, WED, THU, FRI, or SAT. This field is required if the `frequency` parameter is set to WEEKLY or BIWEEKLY. enum: SUN | MON | TUE | WED | THU | FRI | SAT
targetCheckNames	list member: AuditCheckName	Which checks are performed during the scheduled audit. Checks must be enabled for your account. (Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or use `UpdateAccountAuditConfiguration` to select which checks are enabled.)

Name	Type	Description
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit. (Maximum of 128 characters)

Output

```
{
  "scheduledAuditArn": "string"
}
```

CLI output fields

Name	Type	Description
scheduledAuditArn	string	The ARN of the scheduled audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DeleteScheduledAudit

Deletes a scheduled audit.

Synopsis

```
aws iot  delete-scheduled-audit \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit you want to delete.

Output

None

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Run an On-Demand audit

Use `StartOnDemandAuditTask` to specify the checks you want to perform and start an audit running right away.

StartOnDemandAuditTask

Starts an on-demand Device Defender audit.

Synopsis

```
aws iot  start-on-demand-audit-task \
    --target-check-names <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "targetCheckNames": [
    "string"
  ]
}
```

`cli-input-json` Fields

Name	Type	Description
targetCheckNames	list member: AuditCheckName	Which checks are performed during the audit. The checks you specify must be enabled for your account or an exception occurs. Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or use `UpdateAccountAuditConfiguration` to select which checks are enabled.

Output

```
{
  "taskId": "string"
}
```

CLI output fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the on-demand audit you started.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

`LimitExceededException`

A limit has been exceeded.

Manage audit instances

Use `DescribeAuditTask` to get information about a specific audit instance. If it has already run, the results include which checks failed and which passed, those that the system was unable to complete, and if the audit is still in progress, those it is still working on.

Use `ListAuditTasks` to find the audits that were run during a specified time interval.

Use `CancelAuditTask` to halt an audit in progress.

DescribeAuditTask

Gets information about a Device Defender audit.

Synopsis

```
aws iot  describe-audit-task \
    --task-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "taskId": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the audit whose information you want to get.

Output

```
{
  "taskStatus": "string",
  "taskType": "string",
  "taskStartTime": "timestamp",
  "taskStatistics": {
    "totalChecks": "integer",
    "inProgressChecks": "integer",
    "waitingForDataCollectionChecks": "integer",
    "compliantChecks": "integer",
    "nonCompliantChecks": "integer",
    "failedChecks": "integer",
    "canceledChecks": "integer"
```

```
  },
  "scheduledAuditName": "string",
  "auditDetails": {
    "string": {
      "checkRunStatus": "string",
      "checkCompliant": "boolean",
      "totalResourcesCount": "long",
      "nonCompliantResourcesCount": "long",
      "errorCode": "string",
      "message": "string"
    }
  }
}
```

CLI output fields

Name	Type	Description
taskStatus	string	The status of the audit: one of IN_PROGRESS, COMPLETED, FAILED, or CANCELED. enum: IN_PROGRESS | COMPLETED | FAILED | CANCELED
taskType	string	The type of audit: ON_DEMAND_AUDIT_TASK or SCHEDULED_AUDIT_TASK. enum: ON_DEMAND _AUDIT_TASK | SCHEDULED _AUDIT_TASK
taskStartTime	timestamp	The time the audit started.
taskStatistics	TaskStatistics	Statistical information about the audit.
totalChecks	integer	The number of checks in this audit.

Name	Type	Description
inProgressChecks	integer	The number of checks in progress.
waitingForDataCollectionChecks	integer	The number of checks waiting for data collection.
compliantChecks	integer	The number of checks that found compliant resources.
nonCompliantChecks	integer	The number of checks that found noncompliant resources.
failedChecks	integer	The number of checks.
canceledChecks	integer	The number of checks that did not run because the audit was canceled.
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit (only if the audit was a scheduled audit).
auditDetails	map	Detailed information about each check performed during this audit.

Name	Type	Description
checkRunStatus	string	The completion status of this check, one of IN_PROGRESS, WAITING_FOR_DATA_COLLECTION, CANCELED, COMPLETED_COMPLIANT, COMPLETED_NON_COMPLIANT, or FAILED. enum: IN_PROGRESS | WAITING_FOR_DATA_COLLECTION | CANCELED | COMPLETED_COMPLIANT | COMPLETED_NON_COMPLIANT | FAILED
checkCompliant	boolean	True if the check completed and found all resources compliant.
totalResourcesCount	long	The number of resources on which the check was performed.
nonCompliantResourcesCount	long	The number of resources that the check found noncompliant.
errorCode	string	The code of any error encountered when performing this check during this audit. One of INSUFFICIENT_PERMISSIONS or AUDIT_CHECK_DISABLED.

Name	Type	Description
message	string length- max:2048	The message associated with any error encountered when performing this check during this audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

ListAuditTasks

Lists the Device Defender audits that have been performed during a given time period.

Synopsis

```
aws iot  list-audit-tasks \
    --start-time <value> \
    --end-time <value> \
    [--task-type <value>] \
    [--task-status <value>] \
    [--next-token <value>] \
    [--max-results <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "startTime": "timestamp",
  "endTime": "timestamp",
  "taskType": "string",
  "taskStatus": "string",
  "nextToken": "string",
  "maxResults": "integer"
}
```

`cli-input-json` Fields

Name	Type	Description
startTime	timestamp	The beginning of the time period. Audit information is retained for a limited time (180 days). Requesting a start time prior to what is retained results in an `InvalidRequestException` .
endTime	timestamp	The end of the time period.
taskType	string	A filter to limit the output to the specified type of audit: can be one of ON_DEMAND_AUDIT_TASK or SCHEDULED__AUDIT_TASK. enum: ON_DEMAND_AUDIT_TASK | SCHEDULED_AUDIT_TASK
taskStatus	string	A filter to limit the output to audits with the specified completion status: can be one of IN_PROGRESS, COMPLETED, FAILED, or CANCELED.

Name	Type	Description
		enum: IN_PROGRESS | COMPLETED | FAILED | CANCELED
nextToken	string	The token for the next set of results.
maxResults	integer range- max:250 min:1	The maximum number of results to return at one time. The default is 25.

Output

```
{
  "tasks": [
    {
      "taskId": "string",
      "taskStatus": "string",
      "taskType": "string"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Type	Description
tasks	list member: AuditTaskMetadata java class: java.util.List	The audits that were performed during the specified time period.
taskId	string length- max:40 min:1	The ID of this audit.

Name	Type	Description
	pattern: [a-zA-Z0-9-]+	
taskStatus	string	The status of this audit: one of IN_PROGRESS, COMPLETED, FAILED, or CANCELED. enum: IN_PROGRESS | COMPLETED | FAILED | CANCELED
taskType	string	The type of this audit: one of ON_DEMAND_AUDIT_TASK or SCHEDULED_AUDIT_TASK. enum: ON_DEMAND_AUDIT_TASK | SCHEDULED_AUDIT_TASK
nextToken	string	A token that can be used to retrieve the next set of results, or `null` if there are no additional results.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

CancelAuditTask

Cancels an audit that is in progress. The audit can be either scheduled or on-demand. If the audit is not in progress, an `InvalidRequestException` occurs.

Synopsis

```
aws iot  cancel-audit-task \
    --task-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "taskId": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the audit you want to cancel. You can only cancel an audit that is IN_PROGRESS.

Output

None

Errors

`ResourceNotFoundException`

The specified resource does not exist.

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Check audit results

Use `ListAuditFindings` to see the results of an audit. You can filter the results by the type of check, a specific resource, or the time of the audit. You can use this information to mitigate any problems that were found.

You can define mitigation actions and apply them to the findings from your audit. For more information, see Mitigation actions.

ListAuditFindings

Lists the findings (results) of a Device Defender audit or of the audits performed during a specified time period. (Findings are retained for 180 days.)

Synopsis

```
aws iot  list-audit-findings \
    [--task-id <value>] \
    [--check-name <value>] \
    [--resource-identifier <value>] \
    [--max-results <value>] \
    [--next-token <value>] \
    [--start-time <value>] \
    [--end-time <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "taskId": "string",
  "checkName": "string",
  "resourceIdentifier": {
    "deviceCertificateId": "string",
    "caCertificateId": "string",
```

```
    "cognitoIdentityPoolId": "string",
    "clientId": "string",
    "policyVersionIdentifier": {
      "policyName": "string",
      "policyVersionId": "string"
    },

    "roleAliasArn": "string",
    "account": "string"
  },
  "maxResults": "integer",
  "nextToken": "string",
  "startTime": "timestamp",
  "endTime": "timestamp"
}
```

`cli-input-json` Fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	A filter to limit results to the audit with the specified ID. You must specify either the taskId or the startTime and endTime, but not both.
checkName	string	A filter to limit results to the findings for the specified audit check.
resourceIdentifier	ResourceIdentifier	Information that identifies the noncompliant resource.
deviceCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the certificate attached to the resource.
caCertificateId	string length- max:64 min:64	The ID of the CA certificate used to authorize the certifica te.

Name	Type	Description
	pattern: (0x)?[a-fA-F0-9]+	
cognitoIdentityPoolId	string	The ID of the Amazon Cognito identity pool.
clientId	string	The client ID.
policyVersionIdentifier	PolicyVersionIdentifier	The version of the policy associated with the resource.
policyName	string length- max:128 min:1 pattern: [w+=,.@-]+	The name of the policy.
policyVersionId	string pattern: [0-9]+	The ID of the version of the policy associated with the resource.
roleAliasArn	string	The ARN of the role alias that has overly permissive actions. length- max:2048 min:1
account	string length- max:12 min:12 pattern: [0-9]+	The account with which the resource is associated.
maxResults	integer range- max:250 min:1	The maximum number of results to return at one time. The default is 25.
nextToken	string	The token for the next set of results.

Name	Type	Description
startTime	timestamp	A filter to limit results to those found after the specified time. You must specify either the startTime and endTime or the taskId, but not both.
endTime	timestamp	A filter to limit results to those found before the specified time. You must specify either the startTime and endTime or the taskId, but not both.

Output

```
{
  "findings": [
    {
      "taskId": "string",
      "checkName": "string",
      "taskStartTime": "timestamp",
      "findingTime": "timestamp",
      "severity": "string",
      "nonCompliantResource": {
        "resourceType": "string",
        "resourceIdentifier": {
          "deviceCertificateId": "string",
          "caCertificateId": "string",
          "cognitoIdentityPoolId": "string",
          "clientId": "string",
          "policyVersionIdentifier": {
            "policyName": "string",
            "policyVersionId": "string"
          },
          "account": "string"
        },
        "additionalInfo": {
```

```
          "string": "string"
        }
      },
      "relatedResources": [
        {
          "resourceType": "string",
          "resourceIdentifier": {
            "deviceCertificateId": "string",
            "caCertificateId": "string",
            "cognitoIdentityPoolId": "string",
            "clientId": "string",

            "iamRoleArn": "string",

            "policyVersionIdentifier": {
              "policyName": "string",
              "policyVersionId": "string"
            },
            "account": "string"
          },

          "roleAliasArn": "string",

          "additionalInfo": {
            "string": "string"
          }
        }
      ],
      "reasonForNonCompliance": "string",
      "reasonForNonComplianceCode": "string"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Type	Description
findings	list member: AuditFinding	The findings (results) of the audit.

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the audit that generated this result (finding) .
checkName	string	The audit check that generated this result.
taskStartTime	timestamp	The time the audit started.
findingTime	timestamp	The time the result (finding) was discovered.
severity	string	The severity of the result (finding). enum: CRITICAL | HIGH | MEDIUM | LOW
nonCompliantResource	NonCompliantResource	The resource that was found to be noncompliant with the audit check.
resourceType	string	The type of the noncompliant resource. enum: DEVICE_CERTIFICATE | CA_CERTIFICATE | IOT_POLICY | COGNITO_I DENTITY_POOL | CLIENT_ID | ACCOUNT_SETTINGS
resourceIdentifier	ResourceIdentifier	Information that identifies the noncompliant resource.

Name	Type	Description
deviceCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the certificate attached to the resource.
caCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the CA certificate used to authorize the certifica te.
cognitoIdentityPoolId	string	The ID of the Amazon Cognito identity pool.
clientId	string	The client ID.
policyVersionIdentifier	PolicyVersionIdentifier	The version of the policy associated with the resource.
policyName	string length- max:128 min:1 pattern: [w+=,.@-]+	The name of the policy.
policyVersionId	string pattern: [0-9]+	The ID of the version of the policy associated with the resource.
account	string length- max:12 min:12 pattern: [0-9]+	The account with which the resource is associated.
additionalInfo	map	Other information about the noncompliant resource.

Name	Type	Description
relatedResources	list member: RelatedResource	The list of related resources.
resourceType	string	The type of resource. enum: DEVICE_CERTIFICATE | CA_CERTIFICATE | IOT_POLICY | COGNITO_IDENTITY_POOL | CLIENT_ID | ACCOUNT_SETTINGS
resourceIdentifier	ResourceIdentifier	Information that identifies the resource.
deviceCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the certificate attached to the resource.
caCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the CA certificate used to authorize the certificate.
cognitoIdentityPoolId	string	The ID of the Amazon Cognito identity pool.
clientId	string	The client ID.
policyVersionIdentifier	PolicyVersionIdentifier	The version of the policy associated with the resource.
iamRoleArn	string length- max:2048 min:20	The ARN of the IAM role that has overly permissive actions.

Name	Type	Description
policyName	string length- max:128 min:1 pattern: [w+=,.@-]+	The name of the policy.
policyVersionId	string pattern: [0-9]+	The ID of the version of the policy associated with the resource.
roleAliasArn	string length- max:2048 min:1	The ARN of the role alias that has overly permissive actions.
account	string length- max:12 min:12 pattern: [0-9]+	The account with which the resource is associated.
additionalInfo	map	Other information about the resource.
reasonForNonCompliance	string	The reason the resource was noncompliant.
reasonForNonCompliance Code	string	A code that indicates the reason that the resource was noncompliant.
nextToken	string	A token that can be used to retrieve the next set of results, or `null` if there are no additional results.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Audit finding suppressions

When you run an audit, it reports findings for all non-compliant resources. This means your audit reports include findings for resources where you're working toward mitigating issues and also for resources that are known to be non-compliant, such as test or broken devices. The audit continues to report findings for resources that remain non-compliant in successive audit runs, which may add unwanted information to your reports. Audit finding suppressions enable you to suppress or filter out findings for a defined period of time until the resource is fixed, or indefinitely for a resource associated with a test or broken device.

> **Note**
>
> Mitigation actions won't be available for suppressed audit findings. For more information about mitigation actions, see Mitigation actions.

For information about audit finding suppression quotas, see AWS IoT Device Defender endpoints and quotas.

How audit finding suppressions work

When you create an audit finding suppression for a non-compliant resource, your audit reports and notifications behave differently.

Your audit reports will include a new section that lists all the suppressed findings associated with the report. Suppressed findings won't be considered when we evaluate whether an audit check is compliant or not. A suppressed resource count is also returned for each audit check when you use the describe-audit-task command in the command line interface (CLI).

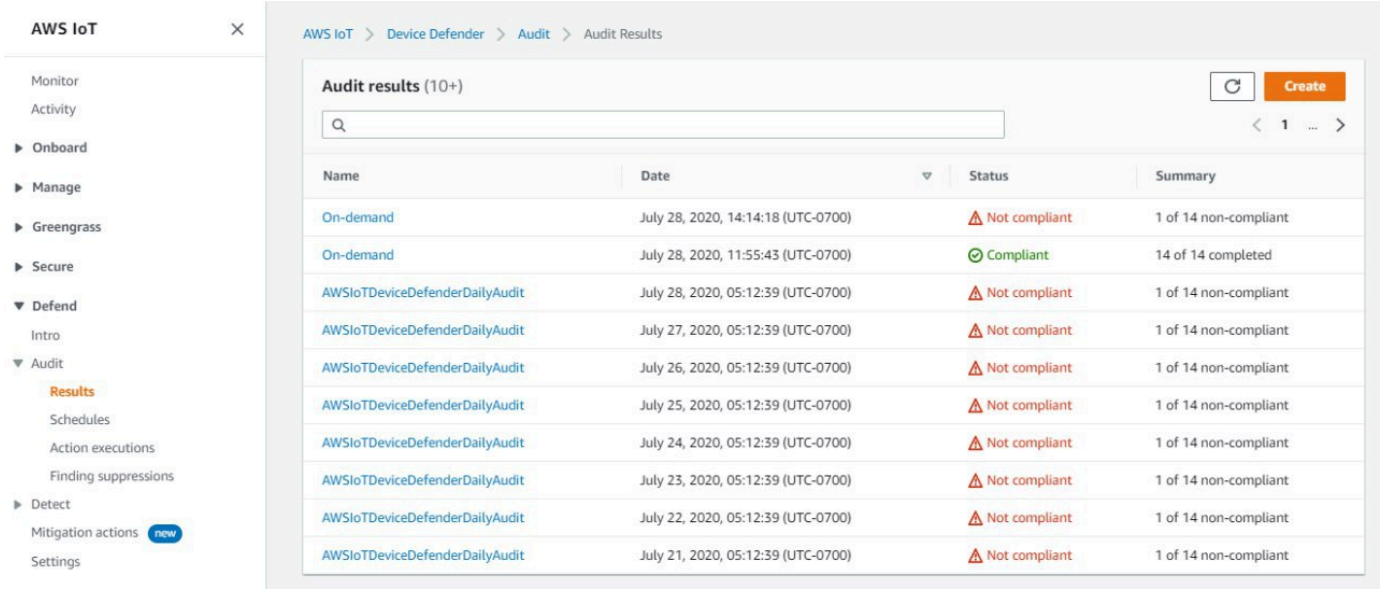
For audit notifications, suppressed findings aren't considered when we evaluate whether an audit check is compliant or not. A suppressed resource count is also included in each audit check notification AWS IoT Device Defender publishes to Amazon CloudWatch and Amazon Simple Notification Service (Amazon SNS).

How to use audit finding suppressions in the console

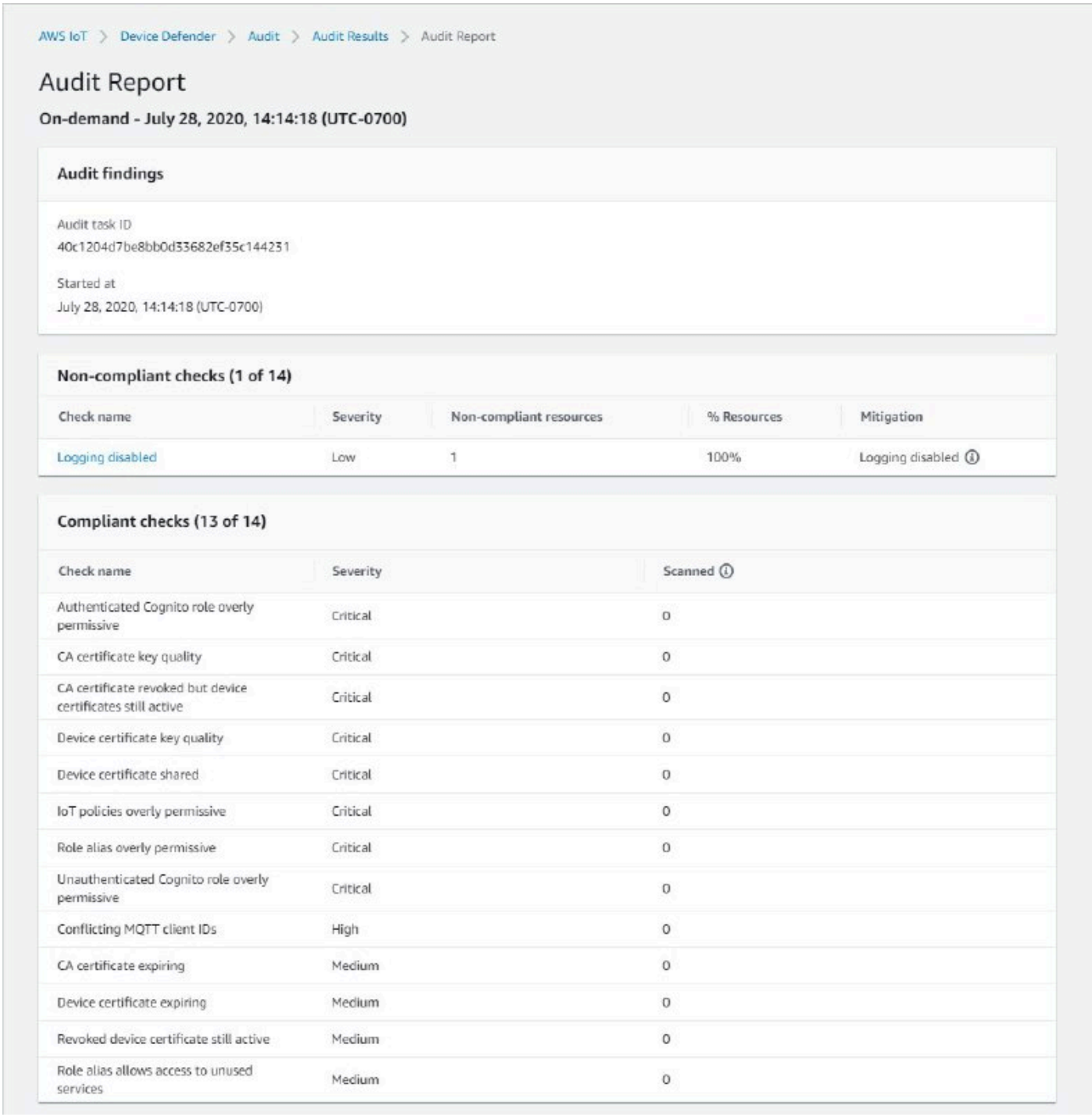
To suppress a finding from an audit report

The following procedure shows you how to create an audit finding suppression in the AWS IoT console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Results**.
2. Select an audit report you'd like to review.

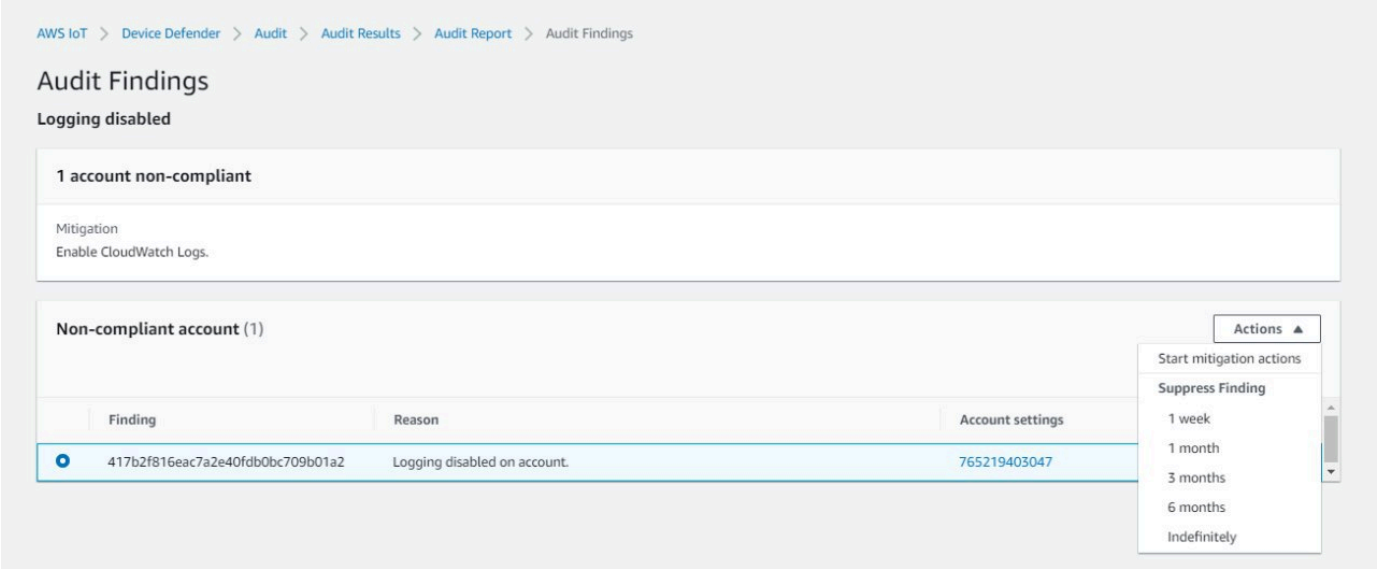


3. In the **Non-compliant checks** section, under **Check name**, choose the audit check that you're interested in.

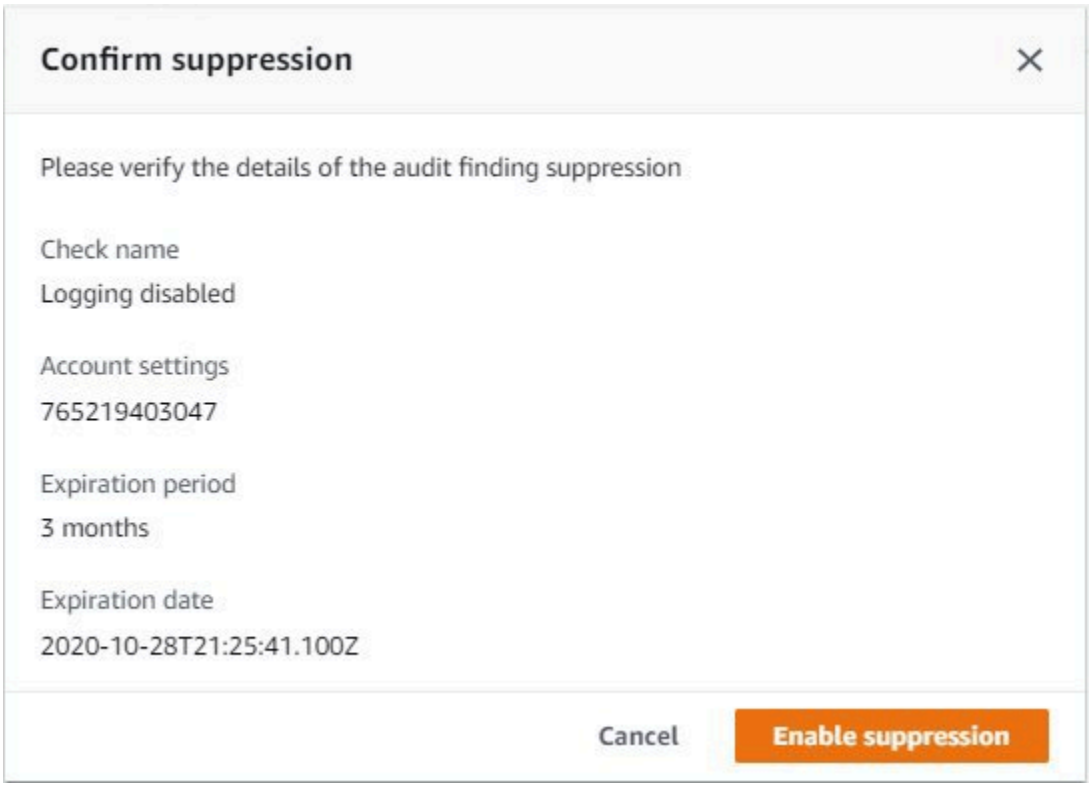


4. On the audit check details screen, if there are findings you don't want to see, select the option button next to the finding. Next, choose **Actions**, and then choose the amount of time you'd like your audit finding suppression to persist.

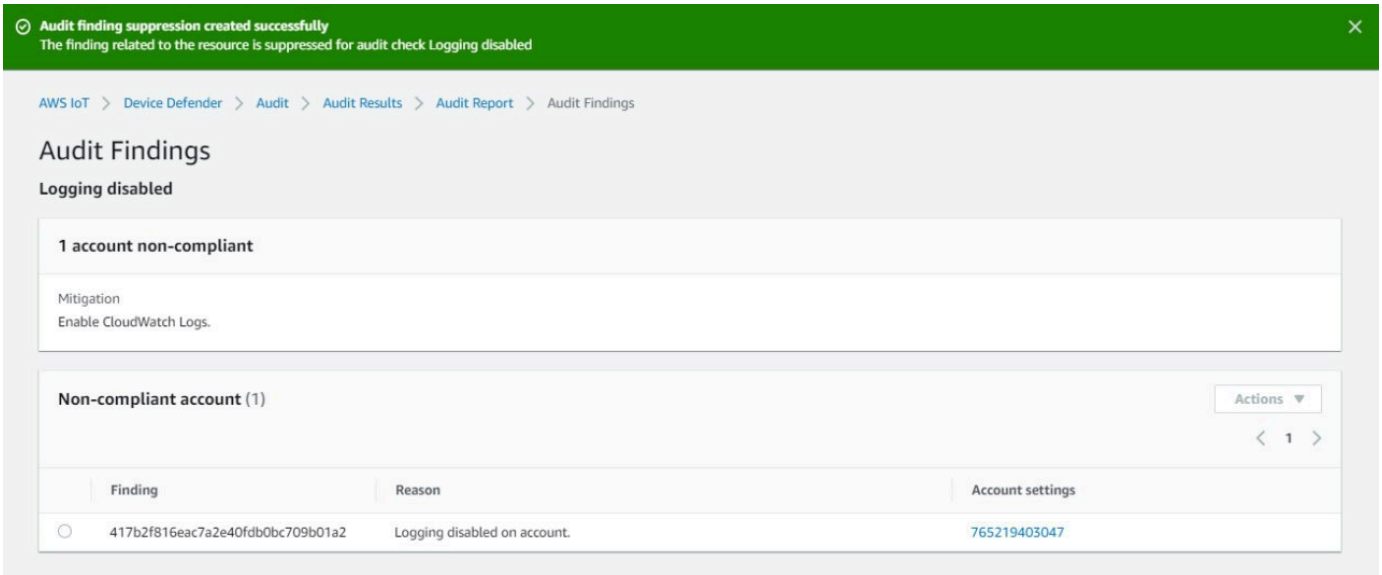
> **Note**
>
> In the console, you can select *1 week*, *1 month*, *3 months*, *6 months*, or *Indefinitely* as expiration dates for your audit finding suppression. If you want to set a specific expiration date, you can do so only in the CLI or API. Audit finding suppressions can also be canceled anytime regardless of expiration date.



5. Confirm the suppression details, and then choose **Enable suppression**.



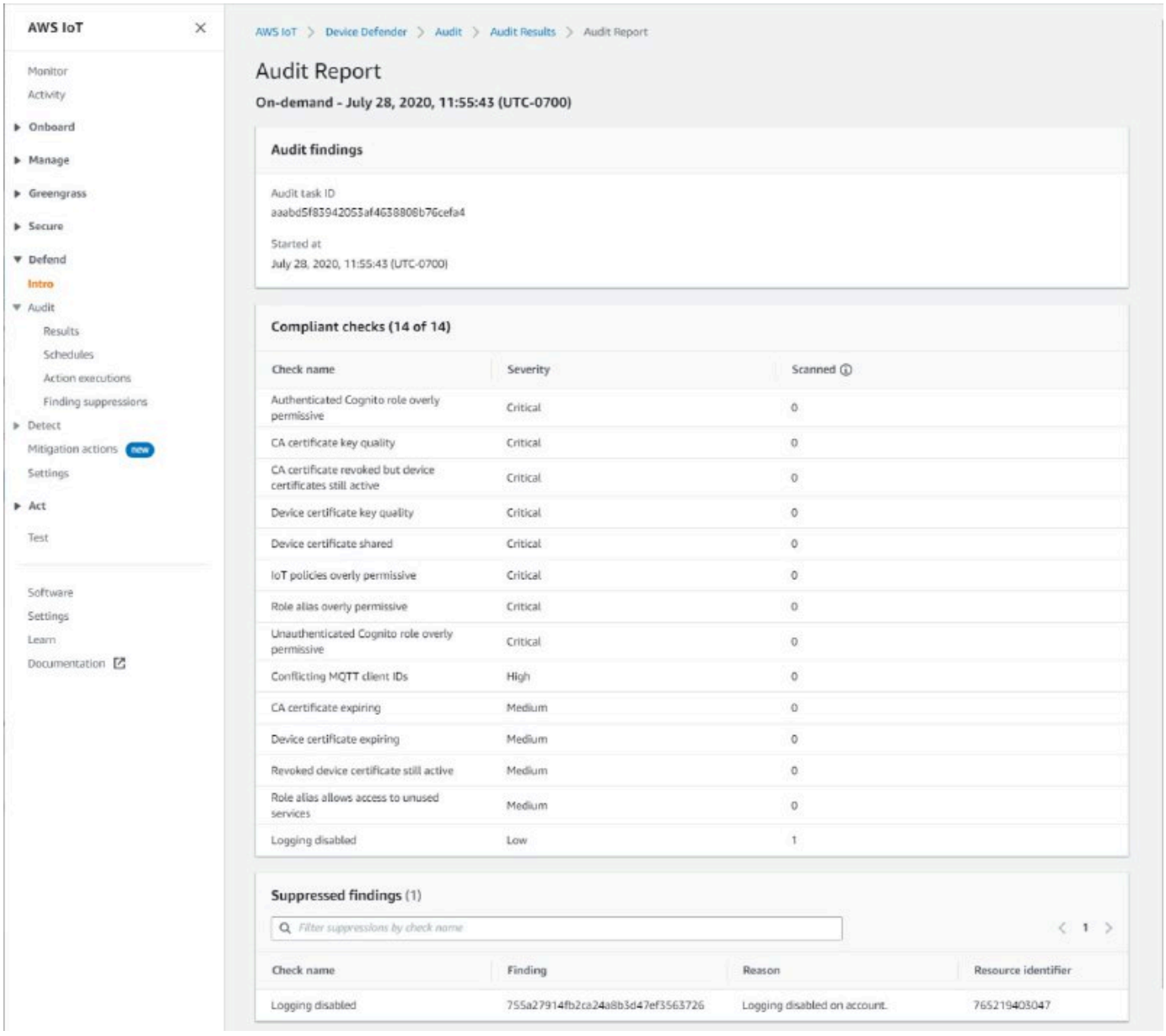
6. After you've created the audit finding suppression, a banner appears confirming your audit finding suppression was created.



To view your suppressed findings in an audit report

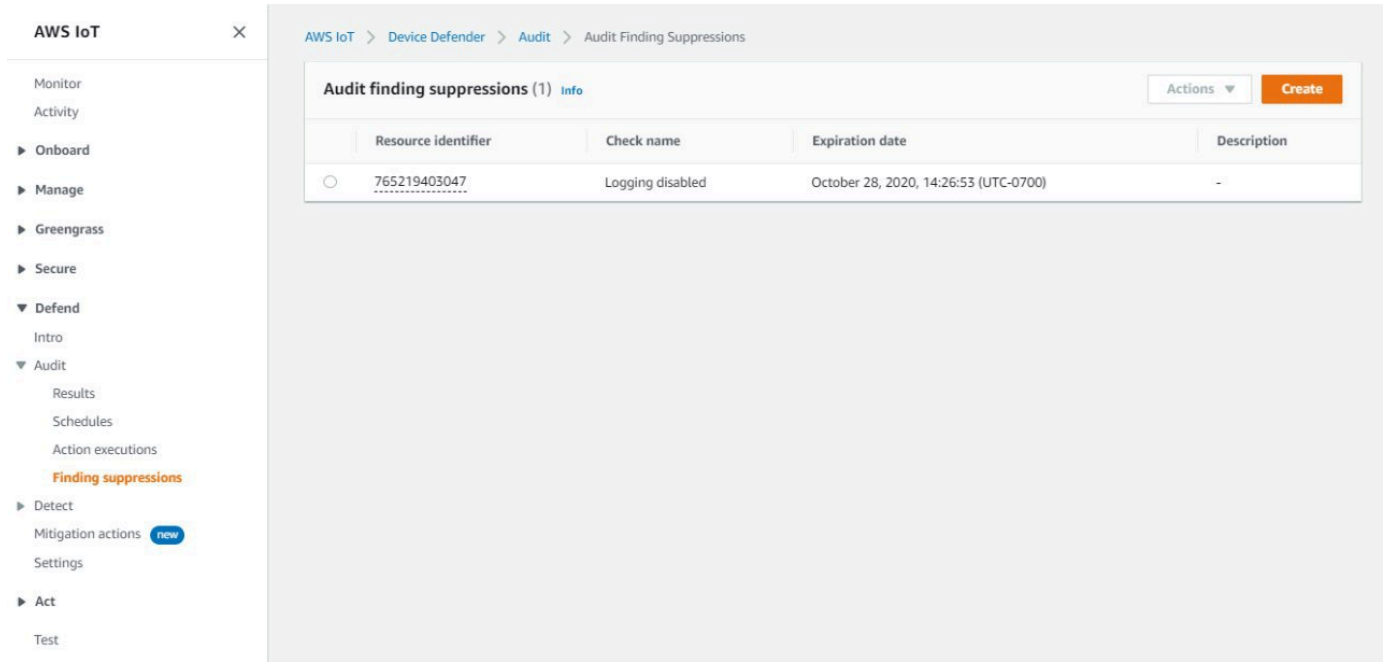
1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Results**.

2. Select an audit report you'd like to review.
3. In the **Suppressed findings** section, view which audit findings have been suppressed for your chosen audit report.



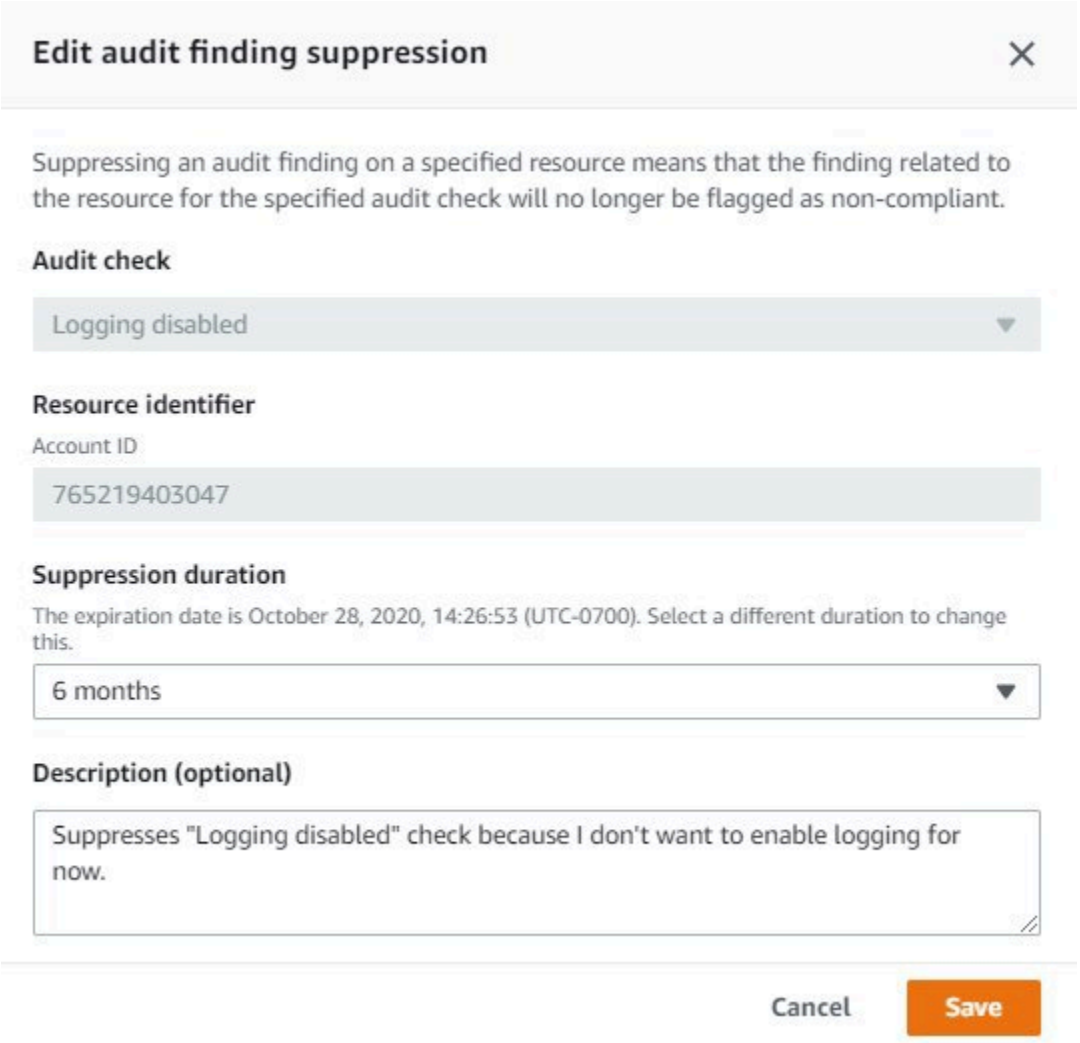
To list your audit finding suppressions

- In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Finding suppressions**.



To edit your audit finding suppression

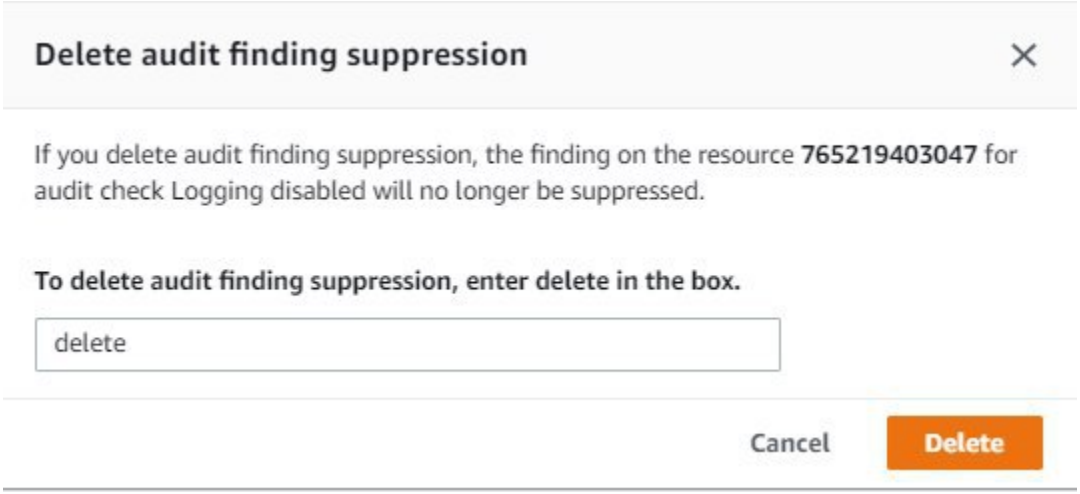
1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Finding suppressions**.
2. Select the option button next to the audit finding suppression you'd like to edit. Next, choose **Actions**, **Edit**.
3. On the **Edit audit finding suppression** window, you can change the **Suppression duration** or **Description (optional)**.



4. After you've made your changes, choose **Save**. The **Finding suppressions** window opens.

To delete an audit finding suppression

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Finding suppressions**.
2. Select the option button next to the audit finding suppression you'd like to delete, and then choose **Actions**, **Delete**.
3. On the **Delete audit finding suppression** window, enter `delete` in the text box to confirm your deletion, and then choose **Delete**. The **Finding suppressions** window opens.



How to use audit finding suppressions in the CLI

You can use the following CLI commands to create and manage audit finding suppressions.

- create-audit-suppression
- describe-audit-suppression
- update-audit-suppression
- delete-audit-suppression
- list-audit-suppressions

The `resource-identifier` you input depends on the `check-name` you're suppressing findings for. The following table details which checks require which `resource-identifier` for creating and editing suppressions.

> **Note**
>
> The suppression commands do not indicate turning off an audit. Audits will still run on your AWS IoT devices. Suppressions are only applicable to the audit findings.

check-name	resource-identifier
AUTHENTICATE_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	cognitoIdentityPoolId

check-name	resource-identifier
CA_CERT_APPROACHING_EXPIRATION_CHECK	caCertificateId
CA_CERTIFICATE_KEY_QUALITY_CHECK	caCertificateId
CONFLICTING_CLIENT_IDS_CHECK	clientId
DEVICE_CERT_APPROACHING_EXPIRATION_CHECK	deviceCertificateId
DEVICE_CERTIFICATE_KEY_QUALITY_CHECK	deviceCertificateId
DEVICE_CERTIFICATE_SHARED_CHECK	deviceCertificateId
IOT_POLICY_OVERLY_PERMISSIVE_CHECK	policyVersionIdentifier
IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK	roleAliasArn
IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK	roleAliasArn
LOGGING_DISABLED_CHECK	account
REVOKED_CA_CERT_CHECK	caCertificateId
REVOKED_DEVICE_CERT_CHECK	deviceCertificateId
UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	cognitoIdentityPoolId

To create and apply an audit finding suppression

The following procedure shows you how to create an audit finding suppression in the AWS CLI.

- Use the `create-audit-suppression` command to create an audit finding suppression. The following example creates an audit finding suppression for AWS account *123456789012* on the basis of the check **Logging disabled**.

```
aws iot create-audit-suppression \
    --check-name LOGGING_DISABLED_CHECK \
    --resource-identifier account=123456789012 \
    --client-request-token 28ac32c3-384c-487a-a368-c7bbd481f554 \
    --suppress-indefinitely \
    --description "Suppresses logging disabled check because I don't want to enable
 logging for now."
```

  There is no output for this command.

Audit finding suppressions APIs

The following APIs can be used to create and manage audit finding suppressions.

- CreateAuditSuppression
- DescribeAuditSuppression
- UpdateAuditSuppression
- DeleteAuditSuppression
- ListAuditSuppressions

To filter *for* specific audit findings, you can use the ListAuditFindings API.

Detect

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

AWS IoT Device Defender Detect lets you identify unusual behavior that might indicate a compromised device by monitoring the behavior of your devices. Using a combination of cloud-side metrics (from AWS IoT) and device-side metrics (from agents that you install on your devices) you can detect:

- Changes in connection patterns.
- Devices that communicate to unauthorized or unrecognized endpoints.
- Changes in inbound and outbound device traffic patterns.

You create security profiles, which contain definitions of expected device behaviors, and assign them to a group of devices or to all the devices in your fleet. AWS IoT Device Defender Detect uses these security profiles to detect anomalies and send alarms through Amazon CloudWatch metrics and Amazon Simple Notification Service notifications.

AWS IoT Device Defender Detect can detect security issues frequently found in connected devices:

- Traffic from a device to a known malicious IP address or to an unauthorized endpoint that indicates a potential malicious command and control channel.
- Anomalous traffic, such as a spike in outbound traffic, that indicates a device is participating in a DDoS.
- Devices with remote management interfaces and ports that are remotely accessible.
- A spike in the rate of messages sent to your account (for example, from a rogue device that can result in excessive per-message charges).

Use cases:

Measure attack surface

You can use AWS IoT Device Defender Detect to measure the attack surface of your devices. For example, you can identify devices with service ports that are often the target of attack campaigns (telnet service running on ports 23/2323, SSH service running on port 22, HTTP/S services running on ports 80/443/8080/8081). While these service ports might have legitimate reasons to be used on the devices, they are also usually part of the attack surface for adversaries and carry associated risks. After AWS IoT Device Defender Detect alarms you to the attack surface, you can minimize it (by eliminating unused network services) or run additional assessments to identify security weaknesses (for example, telnet configured with common, default, or weak passwords).

Detect device behavioral anomalies with possible security root causes

You can use AWS IoT Device Defender Detect to alarm you to unexpected device behavioral metrics (the number of open ports, number of connections, an unexpected open port, connections to unexpected IP addresses) that might indicate a security breach. For example, a higher than expected number of TCP connections might indicate a device is being used for a DDoS attack. A process listening on a port other than the one you expect might indicate a backdoor installed on a device for remote control. You can use AWS IoT Device Defender Detect to probe the health of your device fleets and verify your security assumptions (for example, no device is listening on port 23 or 2323).

You can enable machine learning (ML)-based threat detection to automatically identify potential threats.

Detect an incorrectly configured device

A spike in the number or size of messages sent from a device to your account might indicate an incorrectly configured device. Such a device might increase your per-message charges. Similarly, a device with many authorization failures might require a reconfigured policy.

Monitoring the behavior of unregistered devices

AWS IoT Device Defender Detect makes it possible to identify unusual behaviors for devices that are not registered in the AWS IoT registry. You can define security profiles that are specific to one of the following target types:

- All devices
- All registered devices (things in the AWS IoT registry)
- All unregistered devices
- Devices in a thing group

A security profile defines a set of expected behaviors for devices in your account and specifies the actions to take when an anomaly is detected. Security profiles should be attached to the most specific targets to give you granular control over which devices are being evaluated against that profile.

Unregistered devices must provide a consistent MQTT client identifier or thing name (for devices that report device metrics) over the device lifetime so all violations and metrics are attributed to the same device.

> **Important**
>
> Messages reported by devices are rejected if the thing name contains control characters or if the thing name is longer than 128 bytes of UTF-8 encoded characters.

AWS IoT Device Defender detect feature availability change

The AWS IoT Device Defender detect feature will no longer be available to new customers starting August 31, 2026. Existing AWS IoT Device Defender detect customers can continue using the feature.

As an active customer using the AWS IoT Device Defender detect feature, you can continue using rules detect and ML detect as normal. The detect feature is going into maintenance mode starting August 31, 2026, which means no new capabilities will be added to detect and it will not be available to new customers. AWS remains committed to providing security updates and maintaining feature availability to ensure your monitoring workflows continue to run smoothly.

> **Note**
>
> The AWS IoT Device Defender audit feature continues to be fully available. Only the detect feature (rules detect and ML detect) is entering maintenance mode and will no longer be available to new customers starting August 31, 2026.

AWS IoT Device Defender detect alternatives

Self-managed alternative for cloud-side metrics

Detailed instructions and sample code are available in the IoT Device Anomaly Detection project on the GitHub website. With this open-source, serverless pipeline, you can replicate functionality similar to the AWS IoT Device Defender detect feature for cloud-side metrics monitoring in your own AWS account. It is built on AWS serverless services including AWS IoT, Amazon Managed Service for Apache Flink, AWS Lambda, Amazon Kinesis, Amazon SageMaker AI, Amazon DynamoDB, Amazon API Gateway, Amazon Simple Notification Service, and Amazon CloudWatch. The project includes an AWS Serverless Application Model (SAM) template that allows you to deploy the pipeline in your account within minutes, a REST API for security profile management, and support for both rule-based thresholds and machine-learning-based (ML) anomaly detection. You can tune the metrics sliding-window interval to trade off detection latency against cost.

Key differences from AWS IoT Device Defender detect

- The sample code is an open-source software pipeline that you deploy and operate in your own AWS account. It is not an AWS-managed service.
- It covers cloud-side metrics out of the box. Device-side metrics and custom metrics require extending the pipeline as described in the project's README.
- Configuration is managed through a REST API and does not include AWS Management Console integration.
- Mitigation actions (for example, quarantining a device to a thing group or replacing a certificate) are not included. However, you can implement these by subscribing an Lambda function to the project's Amazon SNS alert topic.
- Improvements over the AWS IoT Device Defender detect feature:
  - **Customizable** – The fully customizable pipeline allows you to add dimensions, adjust aggregation windows, or extend detection logic without service limitations.
  - **Cost control** – The sample code provides levers to optimize costs based on the desired metrics. The pipeline enables you to leverage existing rules and supports messages published to AWS IoT rules basic ingest topics or to the AWS IoT message broker. You can also control the sliding window interval for metrics evaluation to trade off higher latency for lower costs. For ML metrics, the pipeline is also more cost effective at scale than ML detect because it runs on fixed infrastructure rather than incurring per-datapoint costs.

- **Simplified monitoring** – The sample code replaces per-security-profile CloudWatch metrics with a single aggregated metric across the fleet for simplified monitoring, while maintaining device-level behavioral detection.
- **Improved ML model granularity** – Unlike ML detect, which trains a single ML model across all behaviors within a security profile, the sample code trains a separate model per behavior. This gives each model a more focused baseline for a single metric's distribution and ensures that any failures in the model for one behavior do not affect detection accuracy for others.

Open-source device agents for device-side metrics

To collect device-side metrics (such as listening TCP and UDP ports, established TCP connections, destination IP addresses, and packet or byte counters), you can run an open-source agent on your devices and publish the metrics to AWS IoT over MQTT. Options include the AWS IoT Device Client, Telegraf (using its MQTT output plugin), or osquery combined with an Eclipse Paho MQTT client. The GitHub project README also includes instructions on how to ingest the device-published metrics through AWS IoT rules and extend the security profile schemas provided in the sample code.

Frequently asked questions

What does this mean for the AWS IoT Device Defender detect feature?

The AWS IoT Device Defender detect feature will no longer be available to new customers starting August 31, 2026. The feature will continue to operate for existing customers, but there will be no further development of additional functionality.

How are existing customers impacted?

Existing customers will not experience any disruption to their workloads. They can continue using the AWS IoT Device Defender detect feature as normal, and security updates will continue to be deployed to maintain feature reliability. There is no impact to security posture and workflows.

When do I need to migrate to the self-managed alternative?

The AWS IoT Device Defender detect feature remains available to existing customers in maintenance mode. You will receive advance notice if any changes to feature availability are planned. If you are interested in a more customizable solution, explore the self-managed alternative.

Is AWS IoT Device Defender audit affected?

No. Only the detect feature is entering maintenance mode. The AWS IoT Device Defender audit feature continues to be fully available.

Can I migrate gradually?

Yes. You can deploy the sample code in your account and run it alongside AWS IoT Device Defender detect. Both evaluate the same AWS IoT data, so you can compare results and migrate security profiles incrementally.

How can I get help if I have issues?

If you are experiencing issues, contact AWS Support.

Security use cases

Note

The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

This section describes the different types of attacks that threaten your device fleet and the recommended metrics you can use to monitor for these attacks. We recommend using metric anomalies as a starting point to investigate security issues, but you should not base your determination of any security threats solely on a metric anomaly.

To investigate an anomaly alarm, correlate the alarm details with other contextual information such as device attributes, device metric historical trends, Security Profile metric historical trends, custom metrics, and logs to determine if a security threat is present.

Cloud-side use cases

Device Defender can monitor the following use cases on the AWS IoT cloud side.

Intellectual property theft:

Intellectual property theft involves stealing a person's or companies' intellectual properties, including trade secrets, hardware, or software. It often occurs during the manufacturing stage of devices. Intellectual property theft can come in the form of piracy, device theft, or device certificate theft. Cloud-based intellectual property theft can occur due to the presence of policies that permit unintended access to IoT resources. You should review your IoT policies and turn on Audit overly permissive checks to identify overly permissive policies.

Related metrics:

Metric	Rationale
Source IP	If device is stolen, then its source IP address would fall outside of the normally expected IP address range for devices circulated in a normal supply chain.
Number of messages received	Because an attacker may use a device in cloud-based IP theft, metrics related to message counts or message sizes sent to the device from AWS IoT cloud can spike up, indicating a possible security issue.
Message size	

MQTT-based data exfiltration:

Data exfiltration occurs when a malicious actor carries out an unauthorized data transfer from an IoT deployment or from a device. The attacker launches this type of attacks through MQTT against cloud-side data sources.

Related metrics:

Metric	Rationale
Source IP	If a device is stolen, then its source IP address would fall outside of the normally expected IP address range for devices circulated in a standard supply chain.

Metric	Rationale
Number of messages received	Because an attacker may use a device in a MQTT-based data exfiltration, metrics related to message counts or message sizes sent to the device from AWS IoT cloud can spike up, indicating a possible security issue.
Message size	

Impersonation:

An impersonation attack is where attackers pose as known or trusted entities in an effort to access AWS IoT cloud-side services, applications, data, or engage in command and control of IoT devices.

Related metrics:

Metric	Rationale
Authorization failures	When attackers pose as trusted entities by using stolen identities, connectivity related metrics often spike, as the credentials may no longer be valid or may be used by a trusted device already. Anomalous behaviors in authorization failures, connection attempts, or disconnects point to a potential impersonation scenario.
Connection attempts	
Disconnects	

Cloud Infrastructure abuse:

Abuse to AWS IoT cloud services occurs when publishing or subscribing to topics with a high message volume or with messages in large sizes. Overly permissive policies or device vulnerability exploit for command and control can also cause cloud infrastructure abuse. One of the main objectives of this attack is to increase your AWS bill. You should review your IoT policies and turn on Audit overly permissive checks to identify overly permissive policies.

Related metrics:

Metric	Rationale
Number of messages received	The objective of this attack is to increase your AWS bill, metrics that monitor activitie s like message count, messages received and message size will spike up.
Number of messages sent	
Message size	
Source IP	Suspicious source IP lists may appear, from which attackers generate their messaging volume.

Device-side use cases

Device Defender can monitor the following use cases on your device side.

Denial-of-service attack:

A denial-of-service (DoS) attack is aimed at shutting down a device or network, making the device or network inaccessible to their intended users. DoS attacks block access by flooding the target with traffic, or sending it requests that start a system slow-down or cause the system to fail. Your IoT devices can be used in DoS attacks.

Related metrics:

Metric	Rationale
Packets out	DoS attacks typically involve higher rates of outbound communication from a given device, and depending on the type of DoS attack, there could be an increase in either or both of the numbers of packets out and bytes out.
Bytes out	
Destination IP	If you define the IP addresses/CIDR ranges your devices should communicate with, then an anomaly in destination IP can indicate

Metric	Rationale
	unauthroized IP communication from your devices.
Listening TCP ports	A DoS attack usually requires a larger command and control infrastructure where malware installed on your devices receives commands and information about who to attack and when to attack. Therefore , in order to receive such information, the malware would typically listen on ports that aren't normally used by your devices.
Listening TCP port count	
Listening UDP ports	
Listening UDP port count	

Lateral threat escalation:

Lateral threat escalation usually begins with an attacker gaining access to one point of a network, for example a connected device. The attacker then tries to increase their level of privileges, or their access to other devices through methods such as stolen credentials or vulnerability exploits.

Related metrics:

Metric	Rationale
Packets out	In typical situations, the attacker would have to run a scan on the local area network in order to perform reconnaisance and identify the available devices in order to narrow down their attack target selection. This kind of scan could result in a spike of bytes and packets out counts.
Bytes out	
Destination IP	If a device is supposed to communicate with a known set of IP addresses or CIDRs, you can identify if it attempts to communicate with an abnormal IP address, which would often be a private IP address on the local

Metric	Rationale
	network in a lateral threat escalation use case.
Authorization failures	As the attacker tries to increase their level of priviledges across an IoT network, they may use stolen credentials that have been revoked or have expired, which would cause increased authorization failures.

Data exfiltration or surveillance:

Data exfiltration occurs when malware or a malicious actor carries out an unauthorized data transfer from a device or a network endpoint. Data exfiltration normally serves two purposes for the attacker, obtaining data or intellectual property, or conducting reconnaissance of a network. Surveillance means that malicious code is used to monitor user activities for the purpose of stealing credentials and gathering information. The metrics below can provide a starting point of investigating either type of attacks.

Related metrics:

Metric	Rationale
Packets out	When data exfiltration or surveillance attacks occur, the attacker would often mirror the data being sent from the device rather than simply redirecting the data, which would be identified by the defender when they don't see the intended data coming. Such mirrored data would increase the total amount of data sent from the device significantly, resulting in a spike of packets and bytes out counts.
Bytes out	
Destination IP	When an attacker is using a device in data exfiltration or surveilance attacks, the data would have to be sent to an abnormal IP

Metric	Rationale
	address controlled by the attacker. Monitoring the destination IP can help identify such an attack.

Cryptocurrency mining

Attackers leverage processing power from devices to mine cryptocurrency. Crypto-mining is a computationally intensive process, typically requiring network communication with other mining peers and pools.

Related metrics:

Metric	Rationale
Destination IP	Network communication is typically a requirement during cryptomining. Having a tightly controlled list of IP addresses the device should communicate with can help identify unintended communication on a device, like cryptocurrency mining.
CPU usage custom metric	Cryptocurrency mining requires intensive computation resulting in high utilization of the device CPU. If you choose to collect and monitor this metric, a higher-than-normal CPU usage could be an indicator of crypto-mining activities.

Command and control, malware and ransomware

Malware or ransomware restricts your control over your devices, and limits your device functionality. In the case of a ransomware attack, data access would be lost due to encryption the ransomware uses.

Related metrics:

Metric	Rationale
Destination IP	Network or remote attacks represent a large portion of attacks on IoT devices. A tightly controlled list of IP addresses the device should communicate with can help identify abnormal destination IPs resulted from a malware or ransomware attack.
Listening TCP ports	Several malware attacks involve starting a command-and-control server that sends commands to execute on a device. This type of server is critical to a malware or ransomware operation and can be identifie d by tightly monitoring the open TCP/UDP ports and port counts.
Listening TCP port count	
Listening UDP ports	
Listening UDP port count	

Concepts

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

metric

AWS IoT Device Defender Detect uses metrics to detect anomalous behavior of devices. AWS IoT Device Defender Detect compares the reported value of a metric with the expected value you provide. These metrics can be taken from two sources: cloud-side metrics and device-side metrics. ML Detect supports 6 cloud-side metrics and 7 device-side metrics. For a list of supported metrics for ML Detect, see Supported metrics.

Abnormal behavior on the AWS IoT network is detected by using cloud-side metrics such as the number of authorization failures, or the number or size of messages a device sends or receives through AWS IoT.

AWS IoT Device Defender Detect can also collect, aggregate, and monitor metrics data generated by AWS IoT devices (for example, the ports a device is listening on, the number of bytes or packets sent, or the device's TCP connections).

You can use AWS IoT Device Defender Detect with cloud-side metrics alone. To use device-side metrics, you must first deploy the AWS IoT SDK on your AWS IoT connected devices or device gateways to collect the metrics and send them to AWS IoT. See Sending metrics from devices.

Security Profile

A Security Profile defines anomalous behaviors for a group of devices (a Static thing group) or for all devices in your account, and specifies which actions to take when an anomaly is detected. You can use the AWS IoT console or API commands to create a Security Profile and associate it with a group of devices. AWS IoT Device Defender Detect starts recording security-related data and uses the behaviors defined in the Security Profile to detect anomalies in the behavior of the devices.

behavior

A behavior tells AWS IoT Device Defender Detect how to recognize when a device is doing something anomalous. Any device action that doesn't match a behavior triggers an alert. A Rules Detect behavior consists of a metric and an absolute-value or statistical threshold with an operator (for example, less than or equal to, greater than or equal to), which describe the expected device behavior. An ML Detect behavior consists of a metric and an ML Detect configuration, which set an ML model to learn the normal behavior of devices.

ML model

An ML model is a machine learning model created to monitor each behavior a customer configures. The model trains on metric data patterns from targeted device groups and generates three anomaly confidence thresholds (high, medium, and low) for the metric-based behavior. It inferences anomalies based on ingested metric data at the device level. In the context of ML Detect, one ML model is created to evaluate one metric-based behavior. For more information, see ML Detect.

confidence level

ML Detect supports three confidence levels: `High`, `Medium`, and `Low`. `High` confidence means low sensitivity in anomalous behavior evaluation and frequently a lower number of alarms. `Medium` confidence means medium sensitivity and `Low` confidence means high sensitivity and frequently a higher number of alarms.

dimension

You can define a dimension to adjust the scope of a behavior. For example, you can define a topic filter dimension that applies a behavior to MQTT topics that match a pattern. For information about defining a dimension for use in a Security Profile, see CreateDimension.

alarm

When an anomaly is detected, an alarm notification can be sent through a CloudWatch metric (see Monitor AWS IoT alarms and metrics using Amazon CloudWatch in the *AWS IoT Core Developer Guide*) or an SNS notification. An alarm notification is also displayed in the AWS IoT console along with information about the alarm, and a history of alarms for the device. An alarm is also sent when a monitored device stops exhibiting anomalous behavior or when it had been causing an alarm but stops reporting for an extended period.

alarm verification state

After an alarm has been created, you can verify the alarm as True positive, Benign positive, False positive, or Unknown. You can also add a description to your alarm verification state. You can view, organize, and filter AWS IoT Device Defender alarms by using one of the four verification states. You can use alarm verification states and related descriptions to inform members of your team. This helps your team to take follow-up actions, for example, performing mitigation actions on True positive alarms, skipping Benign positive alarms, or continuing investigation on Unknown alarms. The default verification state for all alarms is Unknown.

alarm suppression

Manage Detect alarm SNS notifications by setting behavior notification to on or `suppressed`. Suppressing alarms doesn't stop Detect from performing device behavior evaluations; Detect continues to flag anomalous behaviors as violation alarms. However, suppressed alarms wouldn't be forwarded for SNS notification. They can only be accessed through the AWS IoT console or API.

Behaviors

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

A Security Profile contains a set of behaviors. Each behavior contains a metric that specifies the normal behavior for a group of devices or for all devices in your account. Behaviors fall into two categories: Rules Detect behaviors and ML Detect behaviors. With Rules Detect behaviors, you define how your devices should behave whereas ML Detect uses ML models built on historical device data to evaluate how your devices should behave.

A Security Profile can be one of two threshold types: **ML** or **Rule-based**. ML Security Profiles automatically detect device-level operational and security anomalies across your fleet by learning from past data. Rule-based Security Profiles require that you manually set static rules to monitor your device behaviors.

The following describes some of the fields that are used in the definition of a `behavior`:

Common to Rules Detect and ML Detect

`name`

The name for the behavior.

`metric`

The name of the metric used (that is, what is measured by the behavior).

`consecutiveDatapointsToAlarm`

If a device is in violation of the behavior for the specified number of consecutive data points, an alarm occurs. If not specified, the default is 1.

`consecutiveDatapointsToClear`

If an alarm has occurred and the offending device is no longer in violation of the behavior for the specified number of consecutive data points, the alarm is cleared. If not specified, the default is 1.

`threshold type`

A Security Profile can be one of two threshold types: ML or Rules based. ML Security Profiles automatically detect device-level operational and security anomalies across your fleet by learning from past data. Rule-based Security Profiles require that you manually set static rules to monitor your device behaviors.

`alarm suppressions`

You can manage Detect alarm Amazon SNS notifications by setting behavior notification to on or `suppressed`. Suppressing alarms doesn't stop Detect from performing device behavior evaluations; Detect continues to flag anomalous behaviors as violation alarms. However, suppressed alarms aren't forwarded for Amazon SNS notifications. They can be accessed only through the AWS IoT console or API.

Rules Detect

`dimension`

You can define a dimension to adjust the scope of a behavior. For example, you can define a topic filter dimension that applies a behavior to MQTT topics that match a pattern. To define a dimension for use in a Security Profile, see CreateDimension. Applies to Rules Detect only.

`criteria`

The criteria that determine if a device is behaving normally in regard to the `metric`.

> **Note**
>
> In the AWS IoT console, you can choose **Alert me** to be notified through Amazon SNS when AWS IoT Device Defender detects that a device is behaving anomalously.

`comparisonOperator`

The operator that relates the thing measured (`metric`) to the criteria (`value` or `statisticalThreshold`).

Possible values are: "less-than", "less-than-equals", "greater-than", "greater-than-equals", "in-cidr-set", "not-in-cidr-set", "in-port-set", and "not-in-port-set". Not all operators are valid for

every metric. Operators for CIDR sets and ports are only for use with metrics involving such entities.

`value`

The value to be compared with the `metric`. Depending on the type of metric, this should contain a `count` (a value), `cidrs` (a list of CIDRs), or `ports` (a list of ports).

`statisticalThreshold`

The statistical threshold by which a behavior violation is determined. This field contains a `statistic` field that has the following possible values: "p0", "p0.1", "p0.01", "p1", "p10", "p50", "p90", "p99", "p99.9", "p99.99", or "p100".

This `statistic` indicates a percentile. It resolves to a value by which compliance with the behavior is determined. Metrics are collected one or more times over the specified duration (`durationSeconds`) from all reporting devices associated with this Security Profile, and percentiles are calculated based on that data. After that, measurements are collected for a device and accumulated over the same duration. If the resulting value for the device falls above or below (`comparisonOperator`) the value associated with the percentile specified, then the device is considered to be in compliance with the behavior. Otherwise, the device is in violation of the behavior.

A percentile indicates the percentage of all the measurements considered that fall below the associated value. For example, if the value associated with "p90" (the 90th percentile) is 123, then 90% of all measurements were below 123.

`durationSeconds`

Use this to specify the period of time over which the behavior is evaluated, for those criteria that have a time dimension (for example, `NUM_MESSAGES_SENT`). For a `statisticalThreshhold` metric comparison, this is the time period during which measurements are collected for all devices to determine the `statisticalThreshold` values, and then for each device to determine how its behavior ranks in comparison.

ML Detect

`ML Detect confidence`

ML Detect supports three confidence levels: `High`, `Medium`, and `Low`. `High` confidence means low sensitivity in anomalous behavior evaluation and frequently a lower number of alarms,

`Medium` confidence means medium sensitivity, and `Low` confidence means high sensitivity and frequently a higher number of alarms.

ML Detect

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

> **Note**
>
> ML Detect is not available in the following regions:
>
> - Asia Pacific (Malaysia)

With machine learning Detect (ML Detect), you create Security Profiles that use machine learning to learn expected device behaviors by automatically creating models based on historical device data, and assign these profiles to a group of devices or all the devices in your fleet. AWS IoT Device Defender then identifies anomalies and triggers alarms using the ML models.

For information about how to get started with ML Detect, see ML Detect guide.

This chapter contains the following sections:

- Use cases of ML Detect
- How ML Detect works
- Minimum requirements
- Limitations
- Marking false positives and other verification states in alarms
- Supported metrics
- Service quotas
- ML Detect CLI commands

- ML Detect APIs
- Pause or delete an ML Detect Security Profile

Use cases of ML Detect

You can use ML Detect to monitor your fleet devices when it's difficult to set the expected behaviors of devices. For example, to monitor the number of disconnects metric, it might not be clear what is considered an acceptable threshold. In this case, you can enable ML Detect to identify anomalous disconnect metric datapoints based off historical data reported from devices.

Another use case of ML Detect is to monitor device behaviors that change dynamically over time. ML Detect periodically learns the dynamic expected device behaviors based on changing data patterns from devices. For example, device message sent volume could vary between weekdays and weekends, and ML detect will learn this dynamic behavior.

How ML Detect works

Using ML Detect, you can create behaviors to identify operational and security anomalies across 6 cloud-side metrics and 7 device-side metrics. After the initial model training period, ML Detect refreshes the models daily based on the trailing 14 days of data. It monitors datapoints for these metrics with the ML models and triggers an alarm if an anomaly is detected.

ML Detect works best if you attach a Security Profile to a collection of devices with similar expected behaviors. For example, if some of your devices are used at customers' homes and other devices at business offices, the device behavior patterns might differ significantly between the two groups. You can organize the devices into a *home-device* thing group and an *office-device* thing group. For the best anomaly detection efficacy, attach each thing group to a separate ML Detect Security Profile.

While ML Detect is building the initial model, it requires 14 days and a minimum of 25,000 datapoints per metric over the trailing 14-day period to generate a model. Afterwards, it updates the model every day there is a minimum number of metric datapoints. If the minimum requirement isn't met, ML Detect attempts to build the model the next day, and will retry daily for the next 30 days before discontinuing the model for evaluations.

Minimum requirements

For training and creating the initial ML model, ML Detect has the following minimum requirements.

Minimum training period

It takes 14 days for the initial models to be built. After that, the model refreshes every day with metric data from a 14-day trailing period.

Minimum total datapoints

The minimum required datapoints to build an ML model is 25,000 datapoints per metric for the last 14 days. For ongoing training and refreshing of the model, ML Detect requires the minimum datapoints be met from monitored devices. It's roughly the equivalent of the following setups:

- 60 devices connecting and having activity on AWS IoT at 45-minute intervals.
- 40 devices at 30-minute intervals.
- 15 devices at 10-minute intervals.
- 7 devices at 5-minute intervals.

Device group targets

To collect data, you must have things in the target thing groups for the Security Profile.

After the initial model is created, ML models refresh every day and require at least 25,000 datapoints for 14-day trailing period.

Limitations

You can use ML Detect with dimensions on the following cloud-side metrics:

- Authorization failures (aws:num-authorization-failures)
- Messages received (aws:num-messages-received)
- Messages sent (aws:num-messages-sent)
- Message size (aws:message-byte-size)

The following metrics are not supported with ML Detect.

Cloud-side metrics not supported with ML Detect:

- Source IP (aws:source-ip-address)

Device-side metrics not supported with ML Detect:

- Destination IPs (`aws:destination-ip-addresses`)
- Listening TCP ports (`aws:listening-tcp-ports`)
- Listening UDP ports (`aws:listening-udp-ports`)

Custom metrics only support the **number** type.

Marking false positives and other verification states in alarms

If you verify that an ML Detect alarm is a false positive through your investigation, you can set the verification state of the alarm to False positive. This can help you and your team identify alarms you don't have to respond to. You can also mark alarms as True positive, Benign positive, or Unknown.

You can mark alarms through the AWS IoT Device Defender console or by using the PutVerificationStateOnViolation API action.

Supported metrics

You can use the following cloud-side metrics with ML Detect:

- Authorization failures (aws:num-authorization-failures)
- Connection attempts (aws:num-connection-attempts)
- Disconnects (aws:num-disconnects)
- Message size (aws:message-byte-size)
- Messages sent (aws:num-messages-sent)
- Messages received (aws:num-messages-received)

You can use the following device-side metrics with ML Detect:

- Bytes out (`aws:all-bytes-out`)
- Bytes in (`aws:all-bytes-in`)
- Listening TCP port count (`aws:num-listening-tcp-ports`)
- Listening UDP port count (`aws:num-listening-udp-ports`)
- Packets out (`aws:all-packets-out`)

- Packets in (`aws:all-packets-in`)
- Established TCP connections count (`aws:num-established-tcp-connections`)

Service quotas

For information about ML Detect service quotas and limits, see AWS IoT Device Defender endpoints and quotas.

ML Detect CLI commands

You can use the following CLI commands to create and manage ML Detect.

- create-security-profile
- attach-security-profile
- list-security-profiles
- describe-security-profile
- update-security-profile
- delete-security-profile
- get-behavior-model-training-summaries
- list-active-violations
- list-violation-events

ML Detect APIs

The following APIs can be used to create and manage ML Detect Security Profiles.

- CreateSecurityProfile
- AttachSecurityProfile
- ListSecurityProfiles
- DescribeSecurityProfile
- UpdateSecurityProfile
- DeleteSecurityProfile
- GetBehaviorModelTrainingSummaries

- ListActiveViolations
- ListViolationEvents
- PutVerificationStateOnViolation

Pause or delete an ML Detect Security Profile

You can pause your ML Detect Security Profile to stop monitoring device behaviors temporarily, or delete your ML Detect Security Profile to stop monitoring device behaviors for an extended period of time.

Pause ML Detect Security Profile by using the console

To pause an ML Detect Security Profile using the console, you must first have an empty thing group. To create an empty thing group, see Static thing groups in the *AWS IoT Core Developer Guide*. If you have created an empty thing group, then set the empty thing group as the target of the ML Detect Security Profile.

> **Note**
>
> You need to set the target of your Security Profile back to a device group with devices within 30 days, or you won't be able to reactivate the Security Profile.

Delete ML Detect Security Profile by using the console

To delete a Security Profile, follow these steps:

1. In the AWS IoT console navigate to the sidebar and choose the **Defend** section.
2. Under **Defend**, choose **Detect** and then **Security Profiles**.
3. Choose the ML Detect Security Profile you want to delete.
4. Choose **Actions**, and then from the options, choose **Delete**.

> **Note**
>
> After an ML Detect Security Profile is deleted, you won’t be able to reactivate the Security Profile.

Pause an ML Detect Security Profile by using the CLI

To pause a ML Detect Security Profile by using the CLI, use the `detach-security-security-profile` command:

```
$aws iot detach-security-profile --security-profile-name SecurityProfileName --
security-profile-target-arn arn:aws:iot:us-east-1:123456789012:all/registered-things
```

> **Note**
>
> This option is only available in AWS CLI. Similar to the console workflow, you need to set the target of your Security Profile back to a device group with devices within 30 days, or you won't be able to reactivate the Security Profile. To attach a Security Profile to a device group, use the `attach-security-profile` command.

Delete a ML Detect Security Profile by using the CLI

You can delete a Security Profile by using the `delete-security-profile` command below:

```
delete-security-profile --security-profile-name SecurityProfileName
```

> **Note**
>
> After an ML Detect Security Profile is deleted, you won’t be able to reactivate the Security Profile.

Custom metrics

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

With AWS IoT Device Defender custom metrics, you can define and monitor metrics that are unique to your fleet or use case, such as number of devices connected to Wi-Fi gateways, charge levels for batteries, or number of power cycles for smart plugs. Custom metric behaviors are defined in Security Profiles, which specify expected behaviors for a group of devices (a thing group) or for all devices. You can monitor behaviors by setting up alarms, which you can use to detect and respond to issues that are specific to the devices.

This chapter contains the following sections:

- How to use custom metrics in the console
- How to use custom metrics from the CLI
- Custom metrics CLI commands
- Custom metrics APIs

How to use custom metrics in the console

Tutorials

- AWS IoT Device Defender Agent SDK (Python)
- Create a custom metric and add it to a Security Profile
- View custom metric details
- Update a custom metric
- Delete a custom metric

AWS IoT Device Defender Agent SDK (Python)

To get started, download the AWS IoT Device Defender Agent SDK (Python) sample agent. The agent gathers the metrics and publishes reports. Once your device-side metrics are publishing, you can view the metrics being collected and determine thresholds for setting up alarms. Instructions for setting up the device agent are available on the AWS IoT Device Defender Agent SDK (Python) Readme. For more information, see AWS IoT Device Defender Agent SDK (Python).

Create a custom metric and add it to a Security Profile

The following procedure shows you how to create a custom metric in the console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.

2. On the **Custom metrics** page, choose **Create**.
3. On the **Create custom metric** page, do the following.

    1. Under **Name**, enter a name for your custom metric. You can't modify this name after you create the custom metric.
    2. Under **Display name (optional)**, you can enter a friendly name for your custom metric. It doesn't have to be unique and it can be modified after creation.
    3. Under **Type**, choose the type of metric you'd like to monitor. Metric types include **string-list**, **ip-address-list**, **number-list**, and **number**. The type can't be modified after creation.

        > **Note**
        >
        > ML Detect only allows the **number** type.

    4. Under **Tags**, you can select tags to be associated with the resource.

    When you're done, choose **Confirm**.
4. After you've created your custom metric, the **Custom metrics** page appears, where you can see your newly created custom metric.
5. Next, you need to add your custom metric to a Security Profile. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Security profiles**.
6. Choose the Security Profile you'd like to add your custom metric to.
7. Choose **Actions**, **Edit**.
8. Choose **Additional Metrics to retain**, and then choose your custom metric. Choose **Next** on the following screens until you reach the **Confirm** page. Choose **Save** and **Continue**. After your custom metric has been successfully added, the Security Profile details page appears.

    > **Note**
    >
    > Percentile statistics are not available for metrics when any of the metric values are negative numbers.

View custom metric details

The following procedure shows you how to view a custom metric's details in the console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. Choose the **Metric name** of the custom metric you'd like to view the details of.

Update a custom metric

The following procedure shows you how to update a custom metric in the console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. Choose the option button next to the custom metric you'd like to update. Then, for **Actions**, choose **Edit**.
3. On the **Update custom metric** page, you can edit the display name and remove or add tags.
4. After you're done, choose **Update**. The **Custom metrics** page.

Delete a custom metric

The following procedure shows you how to delete a custom metric in the console.

1. First, remove your custom metric from any Security Profile it's referenced in. You can view which Security Profiles contain your custom metric on your custom metric details page. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. Choose the custom metric you'd like to remove. Remove the custom metric from any Security Profile listed under **Security Profiles** on the custom metric details page.
3. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
4. Choose the option button next to the custom metric you'd like to delete. Then, for **Actions**, choose **Delete**.
5. On the **Are you sure you want to delete custom metric?** message, choose **Delete custom metric**.

> **Warning**
>
> After you've deleted a custom metric, you lose all data associated with the metric. This action can't be undone.

How to use custom metrics from the CLI

Tutorials

- AWS IoT Device Defender Agent SDK (Python)
- Create a custom metric and add it to a Security Profile
- View custom metric details
- Update a custom metric
- Delete a custom metric

AWS IoT Device Defender Agent SDK (Python)

To get started, download the AWS IoT Device Defender Agent SDK (Python) sample agent. The agent gathers the metrics and publishes reports. After your device-side metrics are publishing, you can view the metrics being collected and determine thresholds for setting up alarms. Instructions for setting up the device agent are available on the AWS IoT Device Defender Agent SDK (Python) Readme. For more information, see AWS IoT Device Defender Agent SDK (Python).

Create a custom metric and add it to a Security Profile

The following procedure shows you how to create a custom metric and add it to a Security Profile from the CLI.

1. Use the `create-custom-metric` command to create your custom metric. The following example creates a custom metric that measures battery percentage.

```
aws iot create-custom-metric \
    --metric-name "batteryPercentage" \
    --metric-type "number" \
    --display-name "Remaining battery percentage." \
    --region us-east-1
    --client-request-token "02ccb92b-33e8-4dfa-a0c1-35b181ed26b0" \
```

Output:

```
{
    "metricName": "batteryPercentage",
    "metricArn": "arn:aws:iot:us-
east-1:1234564789012:custommetric/batteryPercentage"
```

```
}
```

2. After you've created your custom metric, you can either add the custom metric to an existing profile using [update-security-profile] or create a new security profile to add the custom metric to using [create-security-profile]. Here, we create a new security profile called *batteryUsage* to add our new *batteryPercentage* custom metric to. We also add a Rules Detect metric called *cellularBandwidth*.

```
aws iot create-security-profile \
    --security-profile-name batteryUsage \
    --security-profile-description "Shows how much battery is left in percentile."
 \
    --behaviors "[{\"name\":\"great-than-75\",\"metric\":\"batteryPercentage\",
\"criteria\":{\"comparisonOperator\":\"greater-than\",\"value\":{\"number
\":75},\"consecutiveDatapointsToAlarm\":5,\"consecutiveDatapointsToClear
\":1}},{\"name\":\"cellularBandwidth\",\"metric\":\"aws:message-byte-size\",
\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]" \
    --region us-east-1
```

Output:

```
{
    "securityProfileArn": "arn:aws:iot:us-east-1:1234564789012:securityprofile/
batteryUsage",
    "securityProfileName": "batteryUsage"
}
```

> **Note**
>
> Percentile statistics are not available for metrics when any of the metric values are negative numbers.

View custom metric details

The following procedure shows you how to view the details for a custom metric from the CLI.

- Use the [list-custom-metrics] command to view all of your custom metrics.

```
aws iot list-custom-metrics \
    --region us-east-1
```

The output of this command looks like the following.

```
{
    "metricNames": [
        "batteryPercentage"
    ]
}
```

Update a custom metric

The following procedure shows you how to update a custom metric from the CLI.

- Use the update-custom-metric command to update a custom metric. The following example updates the `display-name`.

```
aws iot update-custom-metric \
    --metric-name batteryPercentage \
    --display-name 'remaining battery percentage on device' \
    --region us-east-1
```

The output of this command looks like the following.

```
{
    "metricName": "batteryPercentage",
    "metricArn": "arn:aws:iot:us-
east-1:1234564789012:custommetric/batteryPercentage",
    "metricType": "number",
    "displayName": "remaining battery percentage on device",
    "creationDate": "2020-11-17T23:01:35.110000-08:00",
    "lastModifiedDate": "2020-11-17T23:02:12.879000-08:00"
}
```

Delete a custom metric

The following procedure shows you how to delete a custom metric from the CLI.

1. To delete a custom metric, first remove it from any Security Profiles that it's attached to. Use the `list-security-profiles` command to view Security Profiles with a certain custom metric.

2. To remove a custom metric from a Security Profile, use the `update-security-profiles` command. Enter all information that you want to keep, but exclude the custom metric.

```
aws iot update-security-profile \
  --security-profile-name batteryUsage \
  --behaviors "[{\"name\":\"cellularBandwidth\",\"metric\":\"aws:message-byte-size
\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]"
```

   The output of this command looks like the following.

```
{
  "behaviors": [{\"name\":\"cellularBandwidth\",\"metric\":\"aws:message-byte-size
\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}],
  "securityProfileName": "batteryUsage",
  "lastModifiedDate": 2020-11-17T23:02:12.879000-09:00,
  "securityProfileDescription": "Shows how much battery is left in percentile.",
  "version": 2,
  "securityProfileArn": "arn:aws:iot:us-east-1:1234564789012:securityprofile/
batteryUsage",
  "creationDate": 2020-11-17T23:02:12.879000-09:00
}
```

3. After the custom metric is detached, use the `delete-custom-metric` command to delete the custom metric.

```
aws iot delete-custom-metric  \
  --metric-name batteryPercentage \
  --region us-east-1
```

   The output of this command looks like the following

```
HTTP 200
```

Custom metrics CLI commands

You can use the following CLI commands to create and manage custom metrics.

- create-custom-metric
- describe-custom-metric
- list-custom-metrics
- update-custom-metric
- delete-custom-metric
- list-security-profiles

Custom metrics APIs

The following APIs can be used to create and manage custom metrics.

- CreateCustomMetric
- DescribeCustomMetric
- ListCustomMetrics
- UpdateCustomMetric
- DeleteCustomMetric
- ListSecurityProfiles

Device-side metrics

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

When creating a Security Profile, you can specify your IoT device's expected behavior by configuring behaviors and thresholds for metrics generated by IoT devices. The following are device-side metrics, which are metrics from agents that you install on your devices.

Bytes out (`aws:all-bytes-out`)

The number of outbound bytes from a device during a given time period.

Use this metric to specify the maximum or minimum amount of outbound traffic that a device should send, measured in bytes, in a given period of time.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: bytes

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-bytes-out",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 4096
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-bytes-out",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p50"
```

```
    },
    "durationSeconds": 900,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Outbound traffic ML behavior",
  "metric": "aws:all-bytes-out",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Bytes in (`aws:all-bytes-in`)

The number of inbound bytes to a device during a given time period.

Use this metric to specify the maximum or minimum amount of inbound traffic that a device should receive, measured in bytes, in a given period of time.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: bytes

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
```

```
  "name": "TCP inbound traffic",
  "metric": "aws:all-bytes-in",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 4096
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-bytes-in",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Inbound traffic ML behavior",
  "metric": "aws:all-bytes-in",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
```

```
  "suppressAlerts": true
}
```

Listening TCP port count (`aws:num-listening-tcp-ports`)

The number of TCP ports the device is listening on.

Use this metric to specify the maximum number of TCP ports that each device should monitor.

Compatible with: Rules Detect | ML Detect

Unit: failures

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: failures

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Max TCP Ports",
  "metric": "aws:num-listening-tcp-ports",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 5
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "Max TCP Ports",
  "metric": "aws:num-listening-tcp-ports",
  "criteria": {
```

```
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML detect

```
{
  "name": "Max TCP Port ML behavior",
  "metric": "aws:num-listening-tcp-ports",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Listening UDP port count (`aws:num-listening-udp-ports`)

The number of UDP ports the device is listening on.

Use this metric to specify the maximum number of UDP ports that each device should monitor.

Compatible with: Rules Detect | ML Detect

Unit: failures

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: failures

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Max UDP Ports",
  "metric": "aws:num-listening-udp-ports",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 5
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "Max UDP Ports",
  "metric": "aws:num-listening-udp-ports",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Max UPD Port ML behavior",
  "metric": "aws:num-listening-tcp-ports",
  "criteria": {
  "consecutiveDatapointsToAlarm": 1,
  "consecutiveDatapointsToClear": 1,
  "mlDetectionConfig": {
```

```
    "confidenceLevel": "HIGH"
   }
 },
  "suppressAlerts": true
}
```

Packets out (`aws:all-packets-out`)

The number of outbound packets from a device during a given time period.

Use this metric to specify the maximum or minimum amount of total outbound traffic that a device should send in a given period of time.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: packets

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-packets-out",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 100
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
```

```
  "name": "TCP outbound traffic",
  "metric": "aws:all-packets-out",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Outbound sent ML behavior",
  "metric": "aws:all-packets-out",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Packets in (`aws:all-packets-in`)

The number of inbound packets to a device during a given time period.

Use this metric to specify the maximum or minimum amount of total inbound traffic that a device should receive in a given period of time.

Compatible with: Rule Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: packets

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800 or 3600 seconds.

Example

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-packets-in",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 100
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example

Example using a `statisticalThreshold`

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-packets-in",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Inbound sent ML behavior",
  "metric": "aws:all-packets-in",
```

```
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Destination IPs (`aws:destination-ip-addresses`)

A set of IP destinations.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) Classless Inter-Domain Routings (CIDR) from which each device must or must not connect to AWS IoT.

Compatible with: Rules Detect

Operators: in-cidr-set | not-in-cidr-set

Values: a list of CIDRs

Units: n/a

Example

```
{
  "name": "Denied source IPs",
  "metric": "aws:destination-ip-address",
  "criteria": {
    "comparisonOperator": "not-in-cidr-set",
    "value": {
      "cidrs": [ "12.8.0.0/16", "15.102.16.0/24" ]
    }
  },
  "suppressAlerts": true
}
```

Listening TCP ports (`aws:listening-tcp-ports`)

The TCP ports that the device is listening on.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) TCP ports on which each device must or must not listen.

Compatible with: Rules Detect

Operators: in-port-set | not-in-port-set

Values: a list of ports

Units: n/a

Example

```
{
  "name": "Listening TCP Ports",
  "metric": "aws:listening-tcp-ports",
  "criteria": {
    "comparisonOperator": "in-port-set",
    "value": {
      "ports": [ 443, 80 ]
    }
  },
  "suppressAlerts": true
}
```

Listening UDP ports (`aws:listening-udp-ports`)

The UDP ports that the device is listening on.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) UDP ports on which each device must or must not listen.

Compatible with: Rules Detect

Operators: in-port-set | not-in-port-set

Values: a list of ports

Units: n/a

Example

```
{
```

```
  "name": "Listening UDP Ports",
  "metric": "aws:listening-udp-ports",
  "criteria": {
    "comparisonOperator": "in-port-set",
    "value": {
      "ports": [ 1025, 2000 ]
    }
  }
}
```

Established TCP connections count (`aws:num-established-tcp-connections`)

The number of TCP connections for a device.

Use this metric to specify the maximum or minimum number of active TCP connections that each device should have (All TCP states).

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: connections

Example

```
{
  "name": "TCP Connection Count",
  "metric": "aws:num-established-tcp-connections",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 3
    },
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "TCP Connection Count",
  "metric": "aws:num-established-tcp-connections",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 900,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Connection count ML behavior",
  "metric": "aws:num-established-tcp-connections",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Device metrics document specification

Overall structure

Long name	Short name	Required	Type	Constraints	Notes
header	hed	Y	Object		Complete block required for

Long name	Short name	Required	Type	Constraints	Notes
					well-formed report.
metrics	met	Y	Object		A report can have both or at least one `metrics` or `custom_metrics` block.
custom_metrics	cmet	Y	Object		A report can have both or at least one `metrics` or `custom_metrics` block.

Header block

Long name	Short name	Required	Type	Constraints	Notes
report_id	rid	Y	Integer		Monotonically increasing value. Epoch timestamp recommended.
version	v	Y	String	Major.Minor	Minor increments with addition of field. Major

Long name	Short name	Required	Type	Constraints	Notes
					increments if metrics removed.

Metrics block:

TCP connections

Long name	Short name	Parent element	Required	Type	Constraints	Notes
tcp_connections	tc	metrics	N	Object		
established_connections	ec	tcp_connections	N	Object		Established TCP state
connections	cs	established_connections	N	List<Object>		
remote_addr	rad	connections	Y	Number	ip:port	IP can be IPv6 or IPv4
local_port	lp	connections	N	Number	>= 0	
local_interface	li	connections	N	String		Interface name
total	t	established_connections	N	Number	>= 0	Number of established connections

Listening TCP ports

Long name	Short name	Parent element	Required	Type	Constraints	Notes
listening_tcp_ports	tp	metrics	N	Object		
ports	pts	listening_tcp_ports	N	List<Object>	> 0	
port	pt	ports	N	Number	> 0	ports should be numbers greater than 0
interface	if	ports	N	String		Interface name
total	t	listening_tcp_ports	N	Number	>= 0	

Listening UDP ports

Long name	Short name	Parent element	Required	Type	Constraints	Notes
listening_udp_ports	up	metrics	N	Object		
ports	pts	listening_udp_ports	N	List<Port>	> 0	
port	pt	ports	N	Number	> 0	Ports should be numbers

Long name	Short name	Parent element	Required	Type	Constraints	Notes
						greater than 0
interface	if	ports	N	String		Interface name
total	t	listening_udp_ports	N	Number	>= 0	

Network statistics

Long name	Short name	Parent element	Required	Type	Constraints	Notes
network_stats	ns	metrics	N	Object		
bytes_in	bi	network_stats	N	Number	Delta Metric, >= 0	
bytes_out	bo	network_stats	N	Number	Delta Metric, >= 0	
packets_in	pi	network_stats	N	Number	Delta Metric, >= 0	
packets_out	po	network_stats	N	Number	Delta Metric, >= 0	

Example

The following JSON structure uses long names.

```
{
  "header": {
    "report_id": 1530304554,
    "version": "1.0"
  },
  "metrics": {
    "listening_tcp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 24800
        },
        {
          "interface": "eth0",
          "port": 22
        },
        {
          "interface": "eth0",
          "port": 53
        }
      ],
      "total": 3
    },
    "listening_udp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 5353
        },
        {
          "interface": "eth0",
          "port": 67
        }
      ],
      "total": 2
    },
    "network_stats": {
      "bytes_in": 29358693495,
      "bytes_out": 26485035,
      "packets_in": 10013573555,
```

```
      "packets_out": 11382615
    },
    "tcp_connections": {
      "established_connections": {
        "connections": [
          {
            "local_interface": "eth0",
            "local_port": 80,
            "remote_addr": "192.168.0.1:8000"
          },
          {
            "local_interface": "eth0",
            "local_port": 80,
            "remote_addr": "192.168.0.1:8000"
          }
        ],
        "total": 2
      }
    }
  },
  "custom_metrics": {
    "MyMetricOfType_Number": [
      {
        "number": 1
      }
    ],
    "MyMetricOfType_NumberList": [
      {
        "number_list": [
          1,
          2,
          3
        ]
      }
    ],
    "MyMetricOfType_StringList": [
      {
        "string_list": [
          "value_1",
          "value_2"
        ]
      }
    ],
    "MyMetricOfType_IpList": [
```

```
      {
        "ip_list": [
          "172.0.0.0",
          "172.0.0.10"
        ]
      }
    ]
  }
}
```

Example JSON structure using short names

```
{
  "hed": {
    "rid": 1530305228,
    "v": "1.0"
  },
  "met": {
    "tp": {
      "pts": [
        {
          "if": "eth0",
          "pt": 24800
        },
        {
          "if": "eth0",
          "pt": 22
        },
        {
          "if": "eth0",
          "pt": 53
        }
      ],
      "t": 3
    },
    "up": {
      "pts": [
        {
          "if": "eth0",
          "pt": 5353
        },
        {
          "if": "eth0",
```

```
          "pt": 67
        }
      ],
      "t": 2
    },
    "ns": {
      "bi": 29359307173,
      "bo": 26490711,
      "pi": 10014614051,
      "po": 11387620
    },
    "tc": {
      "ec": {
        "cs": [
          {
            "li": "eth0",
            "lp": 80,
            "rad": "192.168.0.1:8000"
          },
          {
            "li": "eth0",
            "lp": 80,
            "rad": "192.168.0.1:8000"
          }
        ],
        "t": 2
      }
    }
  },
  "cmet": {
    "MyMetricOfType_Number": [
      {
        "number": 1
      }
    ],
    "MyMetricOfType_NumberList": [
      {
        "number_list": [
          1,
          2,
          3
        ]
      }
    ],
```

```
    "MyMetricOfType_StringList": [
      {
        "string_list": [
          "value_1",
          "value_2"
        ]
      }
    ],
    "MyMetricOfType_IpList": [
      {
        "ip_list": [
          "172.0.0.0",
          "172.0.0.10"
        ]
      }
    ]
  }
}
```

Sending metrics from devices

AWS IoT Device Defender Detect can collect, aggregate, and monitor metrics data generated by AWS IoT devices to identify devices that exhibit abnormal behavior. This section shows you how to send metrics from a device to AWS IoT Device Defender.

You must securely deploy the AWS IoT SDK version two on your AWS IoT connected devices or device gateways to collect device-side metrics. See the full list of SDKs here.

You can use AWS IoT Device Client to publish metrics as it provides a single agent that covers the features present in both AWS IoT Device Defender and AWS IoT Device Management. These features include jobs, secure tunneling, AWS IoT Device Defender metrics publishing, and more.

You publish device-side metrics to the reserved topic in AWS IoT for AWS IoT Device Defender to collect and evaluate.

Using the AWS IoT Device Client to publish metrics

To install AWS IoT Device Client, you can download it from Github. After you've installed the AWS IoT Device Client on the device for which you want to collect device-side data, you must configure it to send device-side metrics to AWS IoT Device Defender. Verify that the AWS IoT Device Client configuration file has the following parameters set in the `device-defender` section:

```
"device-defender":    {
      "enabled":    true,
      "interval-in-seconds": 300
  }
```

Warning

You should set the time interval to a minimum of 300 seconds. If you set the time interval to anything less than 300 seconds, your metric data may be throttled.

After you've updated your configuration, you can create security profiles and behaviors in the AWS IoT Device Defender console to monitor the metrics that your devices publish to the cloud. You can find published metrics in the AWS IoT Core console by choosing Defend, Detect, and then Metrics.

Cloud-side metrics

Note

The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

When creating a Security Profile, you can specify your IoT device's expected behavior by configuring behaviors and thresholds for metrics generated by IoT devices. The following are cloud-side metrics, which are metrics from AWS IoT.

Message size (aws:message-byte-size)

The number of bytes in a message. Use this metric to specify the maximum or minimum size (in bytes) of each message transmitted from a device to AWS IoT.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: bytes

Example

```
{
  "name": "Max Message Size",
  "metric": "aws:message-byte-size",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 1024
    },
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{

  "name": "Large Message Size",
  "metric": "aws:message-byte-size",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Message size ML behavior",
  "metric": "aws:message-byte-size",
```

```
  "criteria": {
  "consecutiveDatapointsToAlarm": 1,
  "consecutiveDatapointsToClear": 1,
  "mlDetectionConfig": {
    "confidenceLevel": "HIGH"
   }
 },
  "suppressAlerts": true
}
```

An alarm occurs for a device if during three consecutive five-minute periods, it transmits messages where the cumulative size is more than that measured for 90 percent of all other devices reporting for this Security Profile behavior.

Messages sent (aws:num-messages-sent)

The number of messages sent by a device during a given time period.

Use this metric to specify the maximum or minimum number of messages that can be sent between AWS IoT and each device in a given period of time.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: messages

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{

  "name": "Out bound message count",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 50
    },
    "durationSeconds": 300,
```

```
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
    },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{

  "name": "Out bound message rate",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p99"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Messages sent ML behavior",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Messages received (aws:num-messages-received)

The number of messages received by a device during a given time period.

Use this metric to specify the maximum or minimum number of messages that can be received between AWS IoT and each device in a given period of time.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: messages

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "In bound message count",
  "metric": "aws:num-messages-received",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 50
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
    },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "In bound message rate",
  "metric": "aws:num-messages-received",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p99"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
```

```
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Messages received ML behavior",
  "metric": "aws:num-messages-received",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Authorization failures (aws:num-authorization-failures)

Use this metric to specify the maximum number of authorization failures allowed for each device in a given period of time. An authorization failure occurs when a request from a device to AWS IoT is denied (for example, if a device attempts to publish to a topic for which it does not have sufficient permissions).

Compatible with: Rules Detect | ML Detect

Unit: failures

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
```

```
    "comparisonOperator": "less-than",
    "value": {
      "count": 5
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Authorization failures ML behavior",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Source IP (aws:source-ip-address)

The IP address from which a device has connected to AWS IoT.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) Classless Inter-Domain Routings (CIDR) from which each device must or must not connect to AWS IoT.

Compatible with: Rules Detect

Operators: in-cidr-set | not-in-cidr-set

Values: a list of CIDRs

Units: n/a

Example

```
{
  "name": "Denied source IPs",
  "metric": "aws:source-ip-address",
  "criteria": {
    "comparisonOperator": "not-in-cidr-set",
    "value": {
      "cidrs": [ "12.8.0.0/16", "15.102.16.0/24" ]
    }
  },
  "suppressAlerts": true
}
```

Connection attempts (aws:num-connection-attempts)

The number of times a device attempts to make a connection in a given time period.

Use this metric to specify the maximum or minimum number of connection attempts for each device. Successful and unsuccessful attempts are counted.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: connection attempts

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Connection Attempts",
  "metric": "aws:num-connection-attempts",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 5
    },
    "durationSeconds": 600,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "Connection Attempts",
  "metric": "aws:num-connection-attempts",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p10"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Connection attempts ML behavior",
  "metric": "aws:num-connection-attempts",
```

```
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": false
}
```

Disconnects (aws:num-disconnects)

The number of times a device disconnects from AWS IoT during a given time period.

Use this metric to specify the maximum or minimum number of times a device disconnected from AWS IoT during a given time period.

Compatible with: Rules Detect | ML Detect

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: disconnects

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Disconnections",
  "metric": "aws:num-disconnects",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "value": {
      "count": 5
    },
    "durationSeconds": 600,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using a statisticalThreshold

```
{
  "name": "Disconnections",
  "metric": "aws:num-disconnects",
  "criteria": {
    "comparisonOperator": "less-than-equals",
    "statisticalThreshold": {
      "statistic": "p10"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1
  },
  "suppressAlerts": true
}
```

Example using ML Detect

```
{
  "name": "Disconnects ML behavior",
  "metric": "aws:num-disconnects",
  "criteria": {
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 1,
    "mlDetectionConfig": {
      "confidenceLevel": "HIGH"
    }
  },
  "suppressAlerts": true
}
```

Disconnect duration (aws:disconnect-duration)

The duration for which a device stays disconnected from AWS IoT.

Use this metric to specify the maximum duration for which a device remains disconnected from AWS IoT.

Compatible with: Rules Detect

Operators: less-than | less-than-equals

Value: a non-negative integer (in minutes)

Example

```
{
"name": "DisconnectDuration",
  "metric": "aws:disconnect-duration",
  "criteria": {
"comparisonOperator": "less-than-equals",
    "value": {
"count": 5
    }
  },
  "suppressAlerts": true
}
```

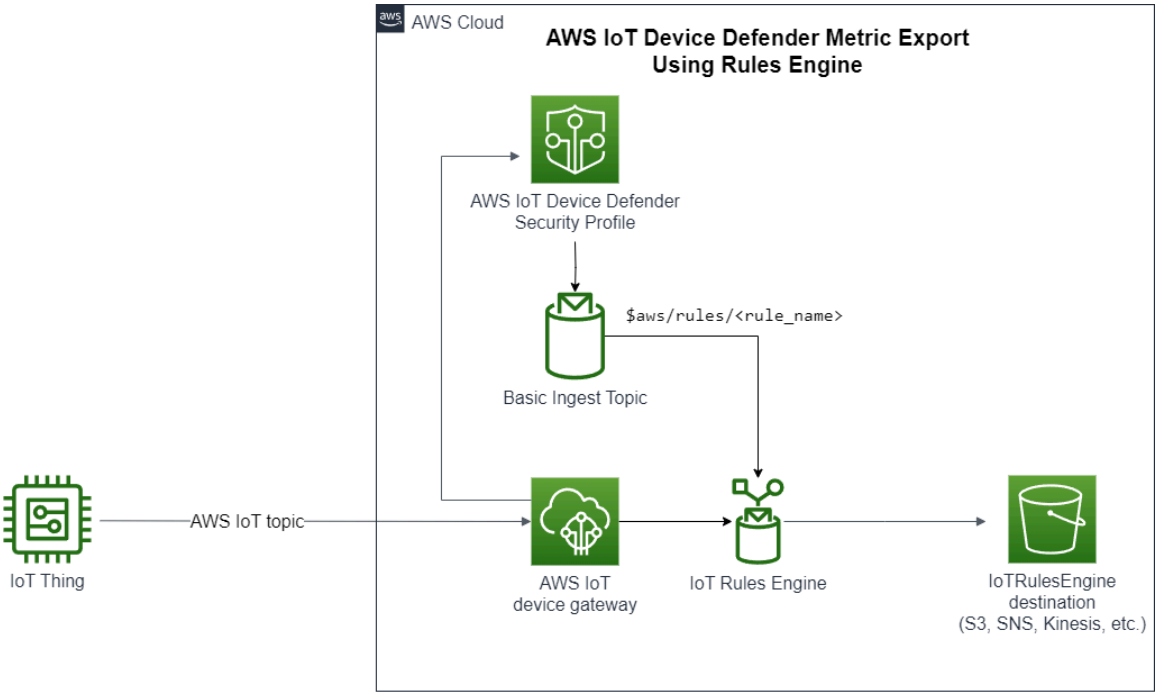
Detect metrics export

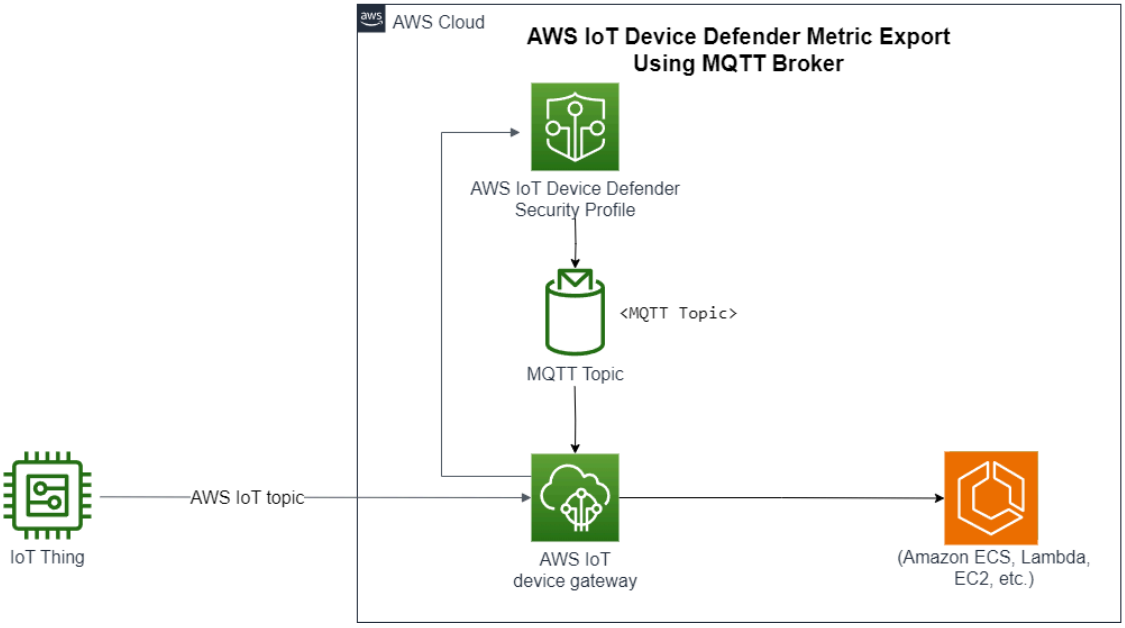
> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

With metrics export, you can export cloud-side, device-side, or custom metrics from AWS IoT Device Defender and publish them to an MQTT topic that you configure. This feature supports the bulk export of Detect metrics, which not only allows for more efficient data reporting and analysis, but also helps control costs. You can choose your MQTT topic as an AWS IoT Rules Basic Ingest Topic or create and subscribe to your own MQTT topic. Configure metrics export by using the AWS IoT Device Defender console, API, or CLI. This feature is available in all AWS Regions where AWS IoT Device Defender is available.

The following illustration shows how you can configure AWS IoT Device Defender to export metrics. The first diagram demonstrates how to configure export metrics on a Basic Ingest topic. You can then route the exported metrics to various destinations supported by AWS IoT Rules. The second diagram shows how to configure AWS IoT Device Defender to publish data to an MQTT topic. The MQTT client then subscribes to that topic. You can run an MQTT client in a container on Amazon

Elastic Container Service, Lambda, or an Amazon EC2 instance that subscribes to the same MQTT topic. Whenever AWS IoT Device Defender publishes data, the MQTT client receives and processes it. For more information, see MQTT topics.





How detect metric export works

When you set up a security profile, you choose the metrics for export and specify the MQTT topic. You also configure an IAM role that grants AWS IoT Device Defender Detect the necessary permissions to publish messages to the configured MQTT topic. You can configure an AWS IoT Rules Basic Ingest MQTT topic and send the exported metrics to AWS IoT Rules supported destinations. For instructions on setting up and configuring AWS IoT Rules, see Rules for AWS IoT in the *AWS IoT Developer Guide*.

AWS IoT Device Defender Detect batches metric values for each configured metric and publishes them to a configured MQTT topic at regular intervals. Except for message byte size and total byte size, cloud-side metrics are aggregated by summing metric values for the batched duration. Custom and device-side metrics aren't aggregated. For message byte size, the export values are the minimum, maximum, and total byte size for the batched duration. For disconnect duration, the export value is the disconnect duration—in seconds— for all tracked devices. This occurs every one-hour interval and also for connection or a disconnection events. For connected devices or connection events, the value is zero. For more information on cloud-side metrics, device-side metrics, and custom metrics, see the following topics in the *AWS IoT Device Defender Developer Guide:*

- Custom metrics
- Cloud-side metrics
- Device-side metrics

You can export batched metrics to different destinations with AWS IoT Rules. For a list of supported destinations, see AWS IoT rule actions. To send individual metrics within a batched export message to a supported destination, use the batchMode option for AWS IoT rules actions. If your preferred AWS IoT Rules destination lacks `batchMode` support, you can still send individual metrics within a batched message by using intermediary actions such as Lambda or Kinesis Data Streams.

Metrics export schema

See the following schema for batched metrics export data.

```
{
 "version": "1.0",
```

```
 "metrics": [
 {
 "name": "{metricName}",
 "thing": "{thingName}",
 "value": {
 # a list of Classless Inter-Domain Routings (CIDR) specifying metric
# source-ip-address and destination-ip-address
 "cidrs": ["string"],
 # a single metric value for cloud/device metrics
 "count": number,
 # a single metric value for custom metric
 "number": number,
 # a list of numbers for custom metrics
 "numbers": [number],
 # a list of ports for cloud/device metrics
 "ports": [number],
 # a list of strings for custom metrics
 "strings": ["string"]
 },
 # In some rare cases we may send multiple values for the same thing, metric and
 timestamp.
 # When there are multiple values, please use the value with highest version number
 # and discard other values.
 "version": number,
 # For cloud-side metrics, this is the time when AWS IoT Device Defender Detect
 aggregates the
 # metrics data received from AWS IoT.
 # For device-side and custom metrics, this is the time at which the metrics data
 # is reported by the devices.
 "timestamp": number,
 # The dimension parameters are optional. It's set only if
 # the metrics are configured with a dimension in the security profile.
 "dimension": {
 "name": "{dimensionName}",
 "operator": "{dimensionOperator}"
 }
 }
 ]
}
```

Detect metrics export pricing

When you publish cloud-side, device-side, or custom metrics to an MQTT topic that you configure, you will not incur charges for this step of the export process. However, in the subsequent steps when you transfer the published metrics to a destination of your choice, by using Rules Engine or Messaging, you will incur costs based on the transfer method you choose. AWS IoT Device Defender publishes batched metrics to MQTT topics as a single message that contains metrics data for multiple devices, which helps control costs. For more information regarding pricing, see the AWS Pricing Calculator.

Permissions

This section contains information about how to set up the IAM roles and policies required to manage AWS IoT Device Defender Detect metrics export. For more information, see the IAM User Guide.

Give AWS IoT Device Defender detect permission to publish messages to an MQTT topic

If you enable metrics export in CreateSecurityProfile, you must specify an IAM role with two policies: a permissions policy and a trust policy. The permissions policy grants permission to AWS IoT Device Defender to publish messages that include metrics to an MQTT topic. The trust policy grants AWS IoT Device Defender permission to assume the required role.

Permission policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/your-topic-name"
            ]
        }
```

```
    ]
}
```

Trust policy

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Pass role policy

You also need an IAM permissions policy attached to the IAM user that allows the user to pass roles. See Granting a User Permissions to Pass a Role to an AWS Service.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::123456789012:role/Role_To_Pass"
```

```
        }
    ]
}
```

Setting up Detect metrics export in the AWS IoT console

Create, view, and edit a new security profile that includes metrics export in the console.

Prerequisites

Before you set up Detect metrics export, make sure you have the following prerequisites:

- An IAM role. For more information about creating an IAM role, see Creating IAM role in the *IAM User Guide*.
- An AWS account that you can sign in to as an AWS Identity and Access Management (IAM) user with correct permissions. For more information on AWS IoT Device Defender Detect permissions, see Permissions in the *AWS IoT Core Developer Guide*.

Creating a new security profile with metrics export (console)

To export metric behavior data, first configure a security profile to include metric exporting. The following procedure details how to set up a rule-based security profile that includes Detect metrics export.

To create a new security profile with metrics export

1. Open the AWS IoT console. On the navigation bar, expand **Security**, **Detect**, **Security profiles**.
2. For **Create Security Profile**, choose **Create Rule-based anomaly Detect profile**.
3. To specify your security profile properties, enter your **Security Profile name** and, for **Target**, choose a group of devices to target for anomalies. (Optional) Include a description and tags to label AWS resources. Choose **Next.**
4. For **Metric**, choose the metrics to define device behavior. You can define the behavior threshold to alert you when your device doesn't meet behavior expectations.
5. To receive alerts for behavior anomalies, choose **Send an alert (define metric behavior)**, and then specify the **Behavior name** and conditions. To retain the metrics without alerts, choose **Don't send an alert (retain metric)**. Choose **Next**.

6. To configure metrics export, choose **Turn on metrics export**.
7. Enter an MQTT topic name for publishing your metric data to AWS IoT Core. Choose an IAM role to grant AWS IoT the permission "AWS IoT:Publish" to publish messages to the configured topic. Choose the metrics that you want to export, and then choose **Next.**

   > **Note**
   >
   > Use the forward slash to represent hierarchical information when entering your MQTT topic name. For example, `$AWS/rules/rule-name/`.

8. To send alerts sent to your AWS console when a device violates a set behavior, choose or create an Amazon SNS topic and IAM role. Choose **Next.**
9. Review your configurations, and then choose **Next.**

Viewing and editing security profile details (console)

To view and edit security profile details

1. Open the AWS IoT console. On the navigation bar, expand **Security**, **Detect**, **Security profiles**.
2. Choose the security profile that you created to include metrics export, and then for **Actions**, choose **Edit**.
3. Under **Target**, select the target device groups you want to edit, and then choose **Next.**
4. To edit metric behavior configurations, choose **Alert me (Define metric behavior)** and then define the conditions when the metric behaviors are met. Choose **Next.**
5. To turn off metrics export configurations, choose **Turn off export metrics.** Choose **Next**.
6. To configure Amazon SNS to send alerts to your AWS IoT console when a device violates a set behavior, choose or create an Amazon SNS topic and IAM role. Choose **Next.**
7. Review your configurations, then choose **Next.**

Creating a security profile to enable metrics export

Use the `create-security-profile` command to create your security profile and enable metrics export.

To create a security profile with metrics export

1. To enable metrics export and indicate if Detect needs to export the corresponding metrics, set the value `exportMetric` as true in both `Behavior` and `AdditionalMetricsToRetainV2`.
2. Include the value for `MetricsExportConfig`. This specifies the MQTT topic and role Amazon Resource Name (ARN) required for metrics export.

> **Note**
>
> Include `mqttTopic` so that AWS IoT Device Defender Detect can publish messages. The role ARN has permission to publish MQTT messages, after which AWS IoT Device Defender Detect can assume the role and publish messages on your behalf.

```
aws iot create-security-profile \
    --security-profile-name CreateSecurityProfileWithMetricsExport \
    --security-profile-description "create security profile with metrics export
 enabled"  \
    --behaviors "[{\"name\":\"BehaviorNumAuthz\",\"metric\":\"aws:num-authorization-
failures\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count
\":5}, \"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1,
\"durationSeconds\":300},\"exportMetric\":true}]" \
    --metrics-export-config "{\"mqttTopic\":\"\$aws/rules/metricsExportRule\",\"roleArn
\":\"arn:aws:iam::123456789012:role/iot-test-role\"}" \
    --region us-east-1
```

Output:

```
{
    "securityProfileName": "CreateSecurityProfileWithMetricsExport",
    "securityProfileArn": "arn:aws:iot:us-east-1:123456789012:securityprofile/
CreateSecurityProfileWithMetricsExport"
}
```

Updating a security profile to enable metrics export (CLI)

Use the `update-security-profile` command to update an existing security profile and enable metrics export.

To update a security profile to enable metrics export

1. To enable metrics export and indicate if Detect needs to export the corresponding metrics, set the value `exportMetric` as true in both `Behavior` and `AdditionalMetricsToRetainV2`.
2. Include the value for `MetricsExportConfig`. This specifies the MQTT topic and role Amazon Resource Name ARN) required for metrics export.

> **Note**
>
> Include `mqttTopic` so that AWS IoT Device Defender Detect can publish messages. The role ARN has permission to publish MQTT messages, after which AWS IoT Device Defender Detect can assume the role and publish messages on your behalf.

```
aws iot update-security-profile \
    --security-profile-name UpdateSecurityProfileWithMetricsExport \
    --security-profile-description "update an existing security profile to enable
 metrics export"  \
    --behaviors "[{\"name\":\"BehaviorNumAuthz\",\"metric\":\"aws:num-authorization-
failures\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count
\":5}, \"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1,
\"durationSeconds\":300},\"exportMetric\":true}]" \
    --metrics-export-config "{\"mqttTopic\":\"\$aws/rules/metricsExportRule\",\"roleArn
\":\"arn:aws:iam::123456789012:role/iot-test-role\"}" \
    --region us-east-1
```

Output:

```
{
    "securityProfileName": "UpdateSecurityProfileWithMetricsExport",
    "securityProfileArn": "arn:aws:iot:us-east-1:123456789012:securityprofile/
UpdateSecurityProfileWithMetricsExport",
    "securityProfileDescription": "update an existing security profile to enable
 metrics export",
    "behaviors": [
        {
            "name": "BehaviorNumAuthz",
            "metric": "aws:num-authorization-failures",
            "criteria": {
                "comparisonOperator": "less-than",
                "value": {
```

```
                    "count": 5
                },
                "durationSeconds": 300,
                "consecutiveDatapointsToAlarm": 1,
                "consecutiveDatapointsToClear": 1
            },
            "exportMetric": true
        }
    ],
    "version": 2,
    "creationDate": "2023-11-09T16:18:37.183000-08:00",
    "lastModifiedDate": "2023-11-09T16:20:15.486000-08:00",
    "metricsExportConfig": {
        "mqttTopic": "$aws/rules/metricsExportRule",
        "roleArn": "arn:aws:iam::123456789012:role/iot-test-role"
    }
}
```

Updating a security profile to turn off metrics export (CLI)

Use the `update-security-profile` command to update an existing security profile and turn off metrics export.

To update a security profile to turn off metrics export

- To update your security profile and remove the metrics export configuration, use the command `--delete-metrics-export-config`.

```
aws iot update-security-profile \
    --security-profile-name UpdateSecurityProfileToDisableMetricsExport \
    --security-profile-description "update an existing security profile to disable
 metrics export"  \
    --behaviors "[{\"name\":\"BehaviorNumAuthz\",\"metric\":\"aws:num-authorization-
failures\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count
\":5}, \"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1,
\"durationSeconds\":300}}]" \
    --delete-metrics-export-config \
    --region us-east-1
```

Output:

```
{
    "securityProfileName": "UpdateSecurityProfileToDisableMetricsExport",
    "securityProfileArn": "arn:aws:iot:us-east-1:123456789012:securityprofile/
UpdateSecurityProfileWithMetricsExport",
    "securityProfileDescription": "update an existing security profile to disable
 metrics export",
    "behaviors": [
        {
            "name": "BehaviorNumAuthz",
            "metric": "aws:num-authorization-failures",
            "criteria": {
                "comparisonOperator": "less-than",
                "value": {
                    "count": 5
                },
                "durationSeconds": 300,
                "consecutiveDatapointsToAlarm": 1,
                "consecutiveDatapointsToClear": 1
            }
        }
    ],
    "version": 2,
    "creationDate": "2023-11-09T16:18:37.183000-08:00",
    "lastModifiedDate": "2023-11-09T16:31:16.265000-08:00"
}
```

For more information, see Detect Commands in the *AWS IoT Developer Guide*.

Metrics export CLI commands

You can use the following CLI commands to create and manage Detect metrics export.

- CreateSecurityProfile
- UpdateSecurityProfile
- DescribeSecurityProfile

Metrics export API operations

You can use the following API operations to create and manage Detect metrics export.

- CreateSecurityProfile

- UpdateSecurityProfile
- DescribeSecurityProfile

Scoping metrics in security profiles using dimensions

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

Dimensions are attributes that you can define to get more precise data about metrics and behaviors in your security profile. You define the scope by providing a value or pattern that is used as a filter. For example, you can define a topic filter dimension that applies a metric only to MQTT topics that match a particular value, such as "data/bulb/+/activity". For information about defining a dimension that you can use in your security profile, see CreateDimension.

Dimension values support MQTT wildcards. MQTT wildcards help you subscribe to multiple topics simultaneously. There are two different kinds of wildcards: single-level (+) and multi-level (#). For example, the dimension value `Data/bulb/+/activity` creates a subscription that matches all topics that exist on the same level as the +. Dimension values also support the MQTT client ID substitution variable ${iot:ClientId}.

Dimensions of type TOPIC_FILTER are compatible with the following set of cloud-side metrics:

- Number of authorization failures
- Message byte size
- Number of messages received
- Number of messages sent
- Source IP address (only available for Rules Detect)

How to use dimensions in the console

To create and apply a dimension to a security profile behavior

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Detect**, and then choose **Security profiles**.
2. On the **Security Profiles** page, choose **Create Security Profile**, and then choose **Create Rule-based anomaly Detect profile**. Or, to apply a dimension to an existing Rule-based security profile, select the security profile and choose **Edit**.
3. On the **Specify security profile properties** page, enter a name for the security profile.
4. Choose the group of devices that you want to target for anomalies.
5. Choose **Next**.
6. On the **Configure metric behaviors** page, choose one of the cloud-side metric dimensions under **Metric type**.
7. For **Metric behavior**, choose **Send an alert (define metric behavior)** to define the expected metric behavior.
8. Choose when you want to be notified for unusual device behavior.
9. Choose **Next**.
10. Review the security profile configuration and choose **Create**.

To view your alarms

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Detect**, and then choose **Alarms**.
2. In the **Thing name** column, choose the thing to see information about what caused the alarm.

To view and update your dimensions

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Detect**, and then choose **Dimensions**.
2. Select the dimension and choose **Edit**.
3. Edit the dimension and choose **Update**.

To delete a dimension

1. Open the AWS IoT console. In the navigation pane, expand **Security**, **Detect**, and then choose **Dimensions**.
2. Before deleting a dimension, you must delete the metric behavior that references the dimension. Confirm that the dimension isn't attached to a security profile by checking the **Security Profiles** column. If the dimension is attached to a security profile, open the **Security profiles** page on the left, and edit the security profile that the dimension is attached to. Then you can proceed with deleting the behavior. If you want to delete another dimension, follow the steps in this section.
3. Select the dimension and choose **Delete**.
4. Enter the dimension name to confirm, and then choose **Delete**.

How to use dimensions on the AWS CLI

To create and apply a dimension to a security profile behavior

1. First create the dimension before attaching it to a security profile. Use the CreateDimension command to create a dimension:

```
aws iot create-dimension \
  --name TopicFilterForAuthMessages \
  --type TOPIC_FILTER \
  --string-values device/+/auth
```

   The output of this command looks like the following:

```
{
    "arn": "arn:aws:iot:us-west-2:123456789012:dimension/
TopicFilterForAuthMessages",
    "name": "TopicFilterForAuthMessages"
}
```

2. Either add the dimension to an existing security profile by using UpdateSecurityProfile, or add the dimension to a new security profile by using CreateSecurityProfile. In the following example, we create a new security profile that checks if messages to `TopicFilterForAuthMessages` are under 128 bytes, and retains the number of messages sent to non-auth topics.

```
aws iot create-security-profile \
  --security-profile-name ProfileForConnectedDevice \
  --security-profile-description "Check to see if messages to
 TopicFilterForAuthMessages are under 128 bytes and retains the number of messages
 sent to non-auth topics." \
  --behaviors "[{\"name\":\"CellularBandwidth\",\"metric\":\"aws:message-byte-size
\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}},{\"name
\":\"Authorization\",\"metric\":\"aws:num-authorization-failures\",\"criteria\":
{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":10},\"durationSeconds
\":300,\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]" \
  --additional-metrics-to-retain-v2 "[{\"metric\": \"aws:num-authorization-failures
\",\"metricDimension\": {\"dimensionName\": \"TopicFilterForAuthMessages\",
\"operator\": \"NOT_IN\"}}]"
```

The output of this command looks like the following:

```
{
    "securityProfileArn": "arn:aws:iot:us-west-2:1234564789012:securityprofile/
ProfileForConnectedDevice",
    "securityProfileName": "ProfileForConnectedDevice"
}
```

To save time, you can also load a parameter from a file instead of typing it as a command line parameter value. For more information, see Loading AWS CLI Parameters from a File. The following shows the `behavior` parameter in expanded JSON format:

```
[
  {
    "criteria": {
      "comparisonOperator": "less-than",
      "consecutiveDatapointsToAlarm": 1,
      "consecutiveDatapointsToClear": 1,
      "value": {
        "count": 128
      }
    },
    "metric": "aws:message-byte-size",
    "metricDimension": {
      "dimensionName:": "TopicFilterForAuthMessages"
    },
```

```
    "name": "CellularBandwidth"
  }
]
```

Or use CreateSecurityProfile using dimension with ML like the following example:

```
aws iot create-security-profile --security-profile-name ProfileForConnectedDeviceML
 \
   --security-profile-description “Check to see if messages to
 TopicFilterForAuthMessages are abnormal”  \
   --behaviors “[{\“name\”:\“test1\”,\“metric\”:\“aws:message-byte-size\”,
\“metricDimension\”:{\“dimensionName\”: \“TopicFilterForAuthMessages\”,\“operator
\”: \“IN\”},\“criteria\”:{\“mlDetectionConfig\”:{\“confidenceLevel\”:\“HIGH\”},
\“consecutiveDatapointsToAlarm\”:1,\“consecutiveDatapointsToClear\”:1}}]” \
   --region us-west-2
```

To view security profiles with a dimension

- Use the ListSecurityProfiles command to view security profiles with a certain dimension:

```
aws iot list-security-profiles \
  --dimension-name TopicFilterForAuthMessages
```

The output of this command looks like the following:

```
{
    "securityProfileIdentifiers": [
        {
            "name": "ProfileForConnectedDevice",
            "arn": "arn:aws:iot:us-west-2:1234564789012:securityprofile/
ProfileForConnectedDevice"
        }
    ]
}
```

To update your dimension

- Use the UpdateDimension command to update a dimension:

```
aws iot update-dimension \
  --name TopicFilterForAuthMessages \
  --string-values device/${iot:ClientId}/auth
```

The output of this command looks like the following:

```
{
    "name": "TopicFilterForAuthMessages",
    "lastModifiedDate": 1585866222.317,
    "stringValues": [
        "device/${iot:ClientId}/auth"
    ],
    "creationDate": 1585854500.474,
    "type": "TOPIC_FILTER",
    "arn": "arn:aws:iot:us-west-2:1234564789012:dimension/
TopicFilterForAuthMessages"
}
```

To delete a dimension

1. To delete a dimension, first detach it from any security profiles that it's attached to. Use the ListSecurityProfiles command to view security profiles with a certain dimension.
2. To remove a dimension from a security profile, use the UpdateSecurityProfile command. Enter all information that you want to keep, but exclude the dimension:

```
aws iot update-security-profile \
  --security-profile-name ProfileForConnectedDevice \
  --security-profile-description "Check to see if authorization fails 10 times in 5
 minutes or if cellular bandwidth exceeds 128" \
  --behaviors "[{\"name\":\"metric\":\"aws:message-byte-size\",\"criteria
\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}},{\"name
\":\"Authorization\",\"metric\":\"aws:num-authorization-failures\",\"criteria\":
{\comparisonOperator\":\"less-than\",\"value\"{\"count\":10},\"durationSeconds
\":300,\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]"
```

The output of this command looks like the following:

```
{
```

```
  "behaviors": [
    {
      "metric": "aws:message-byte-size",
      "name": "CellularBandwidth",
      "criteria": {
        "consecutiveDatapointsToClear": 1,
        "comparisonOperator": "less-than",
        "consecutiveDatapointsToAlarm": 1,
        "value": {
          "count": 128
        }
      }
    },
    {
      "metric": "aws:num-authorization-failures",
      "name": "Authorization",
      "criteria": {
        "durationSeconds": 300,
        "comparisonOperator": "less-than",
        "consecutiveDatapointsToClear": 1,
        "consecutiveDatapointsToAlarm": 1,
        "value": {
          "count": 10
        }
      }
    }
  ],
  "securityProfileName": "ProfileForConnectedDevice",
  "lastModifiedDate": 1585936349.12,
  "securityProfileDescription": "Check to see if authorization fails 10 times in 5
 minutes or if cellular bandwidth exceeds 128",
  "version": 2,
  "securityProfileArn": "arn:aws:iot:us-west-2:123456789012:securityprofile/Preo/
ProfileForConnectedDevice",
  "creationDate": 1585846909.127
}
```

3. After the dimension is detached, use the DeleteDimension command to delete the dimension:

```
aws iot delete-dimension \
  --name TopicFilterForAuthMessages
```

Permissions

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

This section contains information about how to set up the IAM roles and policies required to manage AWS IoT Device Defender Detect. For more information, see the IAM User Guide.

Give AWS IoT Device Defender detect permission to publish alarms to an SNS topic

If you use the `alertTargets` parameter in CreateSecurityProfile, you must specify an IAM role with two policies: a permissions policy and a trust policy. The permissions policy grants permission to AWS IoT Device Defender to publish notifications to your SNS topic. The trust policy grants AWS IoT Device Defender permission to assume the required role.

Permission policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:123456789012:your-topic-name"
            ]
        }
    ]
}
```

Trust policy

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Pass role policy

You also need an IAM permissions policy attached to the IAM user that allows the user to pass roles. See Granting a User Permissions to Pass a Role to an AWS Service.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::123456789012:role/Role_To_Pass"
        }
    ]
}
```

Detect commands

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

You can use the Detect commands in this section to configure ML Detect or Rules Detect Security Profiles, to identify and monitor unusual behaviors that may indicate a compromised device.

DetectMitigation action commands

Start and manage Detect execution
CancelDetectMitigationActionsTask
DescribeDetectMitigationActionsTask
ListDetectMitigationActionsTasks
StartDetectMitigationActionsTask
ListDetectMitigationActionsExecutions

Dimension action commands

Start and manage Dimension execution
CreateDimension
DescribeDimension
ListDimensions
DeleteDimension
UpdateDimension

CustomMetric action commands

Start and manage CustomMetric execution
CreateCustomMetric
UpdateCustomMetric
DescribeCustomMetric
ListCustomMetrics
DeleteCustomMetric

Security Profile action commands

Start and manage Security Profile execution
CreateSecurityProfile
AttachSecurityProfile
DetachSecurityProfile
DeleteSecurityProfile
DescribeSecurityProfile
ListTargetsForSecurityProfile
UpdateSecurityProfile
ValidateSecurityProfileBehaviors
ListSecurityProfilesForTarget

Alarm action commands

Manage alarms and targets
ListActiveViolations

Manage alarms and targets
ListViolationEvents
PutVerificationStateOnViolation

ML Detect action commands

List ML model training data
GetBehaviorModelTrainingSummaries

How to use AWS IoT Device Defender detect

> **Note**
>
> The AWS IoT Device Defender detect feature is no longer available to new customers. To learn about alternatives to AWS IoT Device Defender detect, see AWS IoT Device Defender detect feature availability change. There is no change to AWS IoT Device Defender audit availability.

1. You can use AWS IoT Device Defender Detect with just cloud-side metrics, but if you plan to use device-reported metrics, you must first deploy the AWS IoT SDK on your AWS IoT connected devices or device gateways. For more information, see Sending metrics from devices.
2. Consider viewing the metrics that your devices generate before you define behaviors and create alarms. AWS IoT can collect metrics from your devices so you can first identify usual or unusual behavior for a group of devices, or for all devices in your account. Use CreateSecurityProfile, but specify only those `additionalMetricsToRetain` that you're interested in. Don't specify `behaviors` at this point.

   Use the AWS IoT console to look at your device metrics to see what constitutes typical behavior for your devices.
3. Create a set of behaviors for your security profile. Behaviors contain metrics that specify normal behavior for a group of devices or for all devices in your account. For more information

and examples, see Cloud-side metrics and Device-side metrics. After you create a set of behaviors, you can validate them with ValidateSecurityProfileBehaviors.

4. Use the CreateSecurityProfile action to create a security profile that includes your behaviors. You can use the `alertTargets` parameter to have alarms sent to a target (an SNS topic) when a device violates a behavior. (If you send alarms using SNS, be aware that these count against your AWS account's SNS topic quota. It's possible that a large burst of violations can exceed your SNS topic quota. You can also use CloudWatch metrics to check for violations. For more information, see Monitor AWS IoT alarms and metrics using Amazon CloudWatch in the *AWS IoT Core Developer Guide*.

5. Use the AttachSecurityProfile action to attach the security profile to a group of devices (a thing group), all registered things in your account, all unregistered things, or all devices. AWS IoT Device Defender Detect starts checking for abnormal behavior and, if any behavior violations are detected, sends alarms. You might want to attach a security profile to all unregistered things if, for example, you expect to interact with mobile devices that are not in your account's thing registry. You can define different sets of behaviors for different groups of devices to meet your needs.

   To attach a security profile to a group of devices, you must specify the ARN of the thing group that contains them. A thing group ARN has the following format.

   ```
   arn:aws:iot:region:account-id:thinggroup/thing-group-name
   ```

   To attach a security profile to all of the registered things in an AWS account (ignoring unregistered things), you must specify an ARN with the following format.

   ```
   arn:aws:iot:region:account-id:all/registered-things
   ```

   To attach a security profile to all unregistered things, you must specify an ARN with the following format.

   ```
   arn:aws:iot:region:account-id:all/unregistered-things
   ```

   To attach a security profile to all devices, you must specify an ARN with the following format.

   ```
   arn:aws:iot:region:account-id:all/things
   ```

6. You can also keep track of violations with the ListActiveViolations action, which lets you to see which violations were detected for a given security profile or target device.

   Use the ListViolationEvents action to see which violations were detected during a specified time period. You can filter these results by security profile, device, or alarm verification state.
7. You can verify, organize, and manage your alarms, by marking their verification state and providing a description of that verification state, by using the PutVerificationStateOnViolation action.
8. If your devices violate the defined behaviors too often, or not often enough, you should fine-tune the behavior definitions.
9. To review the security profiles that you set up and the devices that are being monitored, use the ListSecurityProfiles, ListSecurityProfilesForTarget, and ListTargetsForSecurityProfile actions.

   Use the DescribeSecurityProfile action to get more details about a security profile.
10. To update a security profile, use the UpdateSecurityProfile action. Use the DetachSecurityProfile action to detach a security profile from an account or target thing group. Use the DeleteSecurityProfile action to delete a security profile entirely.

Mitigation actions

You can use AWS IoT Device Defender to take actions to mitigate issues that were found in an Audit finding or Detect alarm.

> **Note**
>
> Mitigation actions won't be performed on suppressed audit findings. For more information about audit finding suppressions, see Audit finding suppressions.

Audit mitigation actions

AWS IoT Device Defender provides predefined actions for the different audit checks. You configure those actions for your AWS account and then apply them to a set of findings. Those findings can be:

- All findings from an audit. This option is available in both the AWS IoT console and by using the AWS CLI.
- A list of individual findings. This option is only available by using the AWS CLI.
- A filtered set of findings from an audit.

The following table lists the types of audit checks and the supported mitigation actions for each:

Audit check to mitigation action mapping

Audit check	Supported mitigation actions
REVOKED_CA_CERT_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_CA_CERTIFICATE
INTERMEDIATE_CA_REVOKED_FOR_ACTIVE_DEVICE_CERTIFICATES_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP

Audit check	Supported mitigation actions
DEVICE_CERTIFICATE_SHARED_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS
AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS
IOT_POLICY_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS, REPLACE_DEFAULT_POLICY_VERSION
IOT_POLICY_POTENTIAL_MISCONFIGURATION_CHECK	PUBLISH_FINDING_TO_SNS, REPLACE_DEFAULT_POLICY_VERSION
CA_CERTIFICATE_EXPIRING_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_CA_CERTIFICATE
CONFLICTING_CLIENT_IDS_CHECK	PUBLISH_FINDING_TO_SNS
DEVICE_CERTIFICATE_EXPIRING_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
REVOKED_DEVICE_CERTIFICATE_STILL_ACTIVE_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
LOGGING_DISABLED_CHECK	PUBLISH_FINDING_TO_SNS, ENABLE_IOT_LOGGING
DEVICE_CERTIFICATE_KEY_QUALITY_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP

Audit check	Supported mitigation actions
CA_CERTIFICATE_KEY_QUALITY_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_CA_CERTIFICATE
IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS
IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK	PUBLISH_FINDING_TO_SNS

All audit checks support publishing the audit findings to Amazon SNS so you can take custom actions in response to the notification. Each type of audit check can support additional mitigation actions:

REVOKED_CA_CERT_CHECK

- Change the state of the certificate to mark it as inactive in AWS IoT.

DEVICE_CERTIFICATE_SHARED_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK

- No additional supported actions.

AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK

- No additional supported actions.

IOT_POLICY_OVERLY_PERMISSIVE_CHECK

- Add a blank AWS IoT policy version to restrict permissions.

IOT_POLICY_POTENTIAL_MISCONFIGURATION_CHECK

- Identify potential misconfigurations in AWS IoT policies.

CA_CERT_APPROACHING_EXPIRATION_CHECK

- Change the state of the certificate to mark it as inactive in AWS IoT.

CONFLICTING_CLIENT_IDS_CHECK

- No additional supported actions.

DEVICE_CERT_APPROACHING_EXPIRATION_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

DEVICE_CERTIFICATE_KEY_QUALITY_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

CA_CERTIFICATE_KEY_QUALITY_CHECK

- Change the state of the certificate to mark it as inactive in AWS IoT.

REVOKED_DEVICE_CERT_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

LOGGING_DISABLED_CHECK

- Enable logging.

AWS IoT Device Defender supports the following types of mitigation actions on Audit findings:

Action type	Notes
ADD_THINGS_TO_THING_GROUP	You specify the group to which you want to add the devices. You also specify whether membership in one or more dynamic groups should be overridden if that would exceed the maximum number of groups to which the thing can belong.
ENABLE_IOT_LOGGING	You specify the logging level and the role with permissions for logging. You cannot specify a logging level of `DISABLED`.
PUBLISH_FINDING_TO_SNS	You specify the topic to which the finding should be published.
REPLACE_DEFAULT_POLICY_VERSION	You specify the template name. Replaces the policy version with a default or blank policy.

Action type	Notes
	Only a value of `BLANK_POLICY` is currently supported.
UPDATE_CA_CERTIFICATE	You specify the new state for the CA certificate. Only a value of `DEACTIVATE` is currently supported.
UPDATE_DEVICE_CERTIFICATE	You specify the new state for the device certificate. Only a value of `DEACTIVATE` is currently supported.

By configuring standard actions when issues are found during an audit, you can respond to those issues consistently. Using these defined mitigation actions also helps you resolve the issues more quickly and with less chance of human error.

Important

Applying mitigation actions that change certificates, add things to a new thing group, or replace the policy can have an impact on your devices and applications. For example, devices might be unable to connect. Consider the implications of the mitigation actions before you apply them. You might need to take other actions to correct the problems before your devices and applications can function normally. For example, you might need to provide updated device certificates. Mitigation actions can help you quickly limit your risk, but you must still take corrective actions to address the underlying issues.

Some actions, such as reactivating a device certificate, can only be performed manually. AWS IoT Device Defender does not provide a mechanism to automatically roll back mitigation actions that have been applied.

Detect mitigation actions

AWS IoT Device Defender supports the following types of mitigation actions on Detect alarms:

Action type	Notes
ADD_THINGS_TO_THING_GROUP	You specify the group to which you want to add the devices. You also specify whether membership in one or more dynamic groups should be overridden if that would exceed the maximum number of groups to which the thing can belong.

How to define and manage mitigation actions

You can use the AWS IoT console or the AWS CLI to define and manage mitigation actions for your AWS account.

Create mitigation actions

Each mitigation action that you define is a combination of a predefined action type and parameters specific to your account.

To use the AWS IoT console to create mitigation actions

1. Open the Mitigation actions page in the AWS IoT console.
2. On the **Mitigation actions** page, choose **Create**.
3. On the **Create a new mitigation action** page, in **Action name**, enter a unique name for your mitigation action.
4. In **Action type**, specify the type of action that you want to define.
5. In **Permissions**, choose the IAM role under whose permissions the action is applied.
6. Each action type requests a different set of parameters. Enter the parameters for the action. For example, if you choose the **Add things to thing group** action type, choose the destination group and select or clear **Override dynamic groups**.
7. Choose **Create** to save your mitigation action to your AWS account.

To use the AWS CLI to create mitigation actions

- Use the CreateMitigationAction command to create your mitigation action. The unique name that you give the action is used when you apply that action to audit findings. Choose a meaningful name.

To use the AWS IoT console to view and modify mitigation actions

1. Open the Mitigation actions page in the AWS IoT console.

   The **Mitigation actions** page displays a list of all of the mitigation actions that are defined for your AWS account.
2. Choose the action name link for the mitigation action that you want to change.
3. Choose **Edit** and make your changes to the mitigation action. You cannot change the name because the name of the mitigation action is used to identify it.
4. Choose **Update** to save the changes to the mitigation action to your AWS account.

To use the AWS CLI to list a mitigation action

- Use the ListMitigationAction command to list your mitigation actions. If you want to change or delete a mitigation action, make a note of the name.

To use the AWS CLI to update a mitigation action

- Use the UpdateMitigationAction command to change your mitigation action.

To use the AWS IoT console to delete a mitigation action

1. Open the Mitigation actions page in the AWS IoT console.

   The **Mitigation actions** page displays all of the mitigation actions that are defined for your AWS account.
2. Choose the the mitigation action that you want to delete, and then choose **Delete**.
3. In the **Are you sure you want to delete** window, choose **Delete**.

To use the AWS CLI to delete mitigation actions

- Use the UpdateMitigationAction command to change your mitigation action.

To use the AWS IoT console to view mitigation action details

1. Open the Mitigation actions page in the AWS IoT console.

   The **Mitigation actions** page displays all of the mitigation actions that are defined for your AWS account.
2. Choose the action name link for the mitigation action that you want to view.

To use the AWS CLI to view mitigation action details

- Use the DescribeMitigationAction command to view details for your mitigation action.

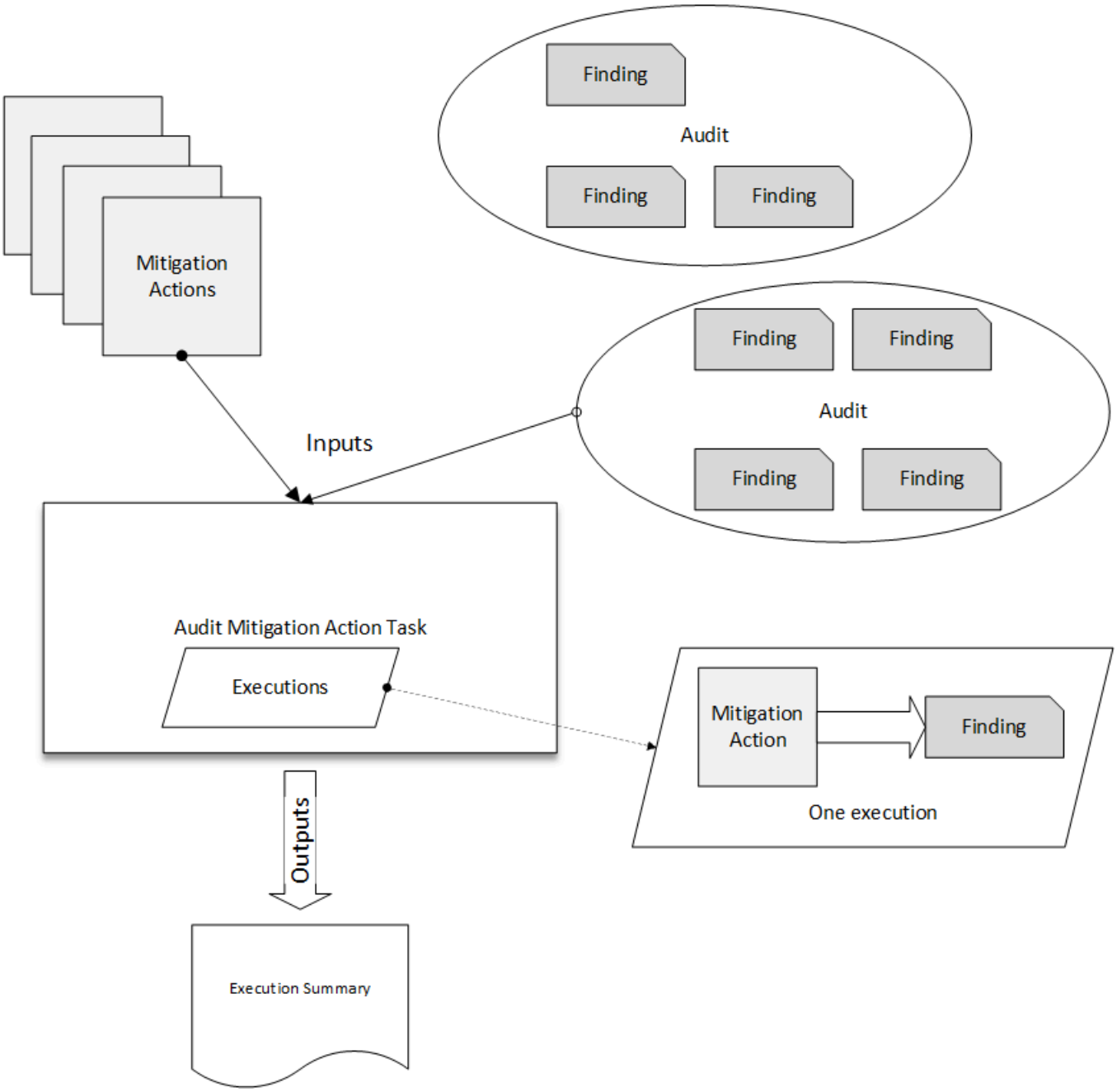
Apply mitigation actions

After you have defined a set of mitigation actions, you can apply those actions to the findings from an audit. When you apply actions, you start an audit mitigation actions task. This task might take some time to complete, depending on the set of findings and the actions that you apply to them. For example, if you have a large pool of devices whose certificates have expired, it might take some time to deactivate all of those certificates or to move those devices to a quarantine group. Other actions, such as enabling logging, can be completed quickly.

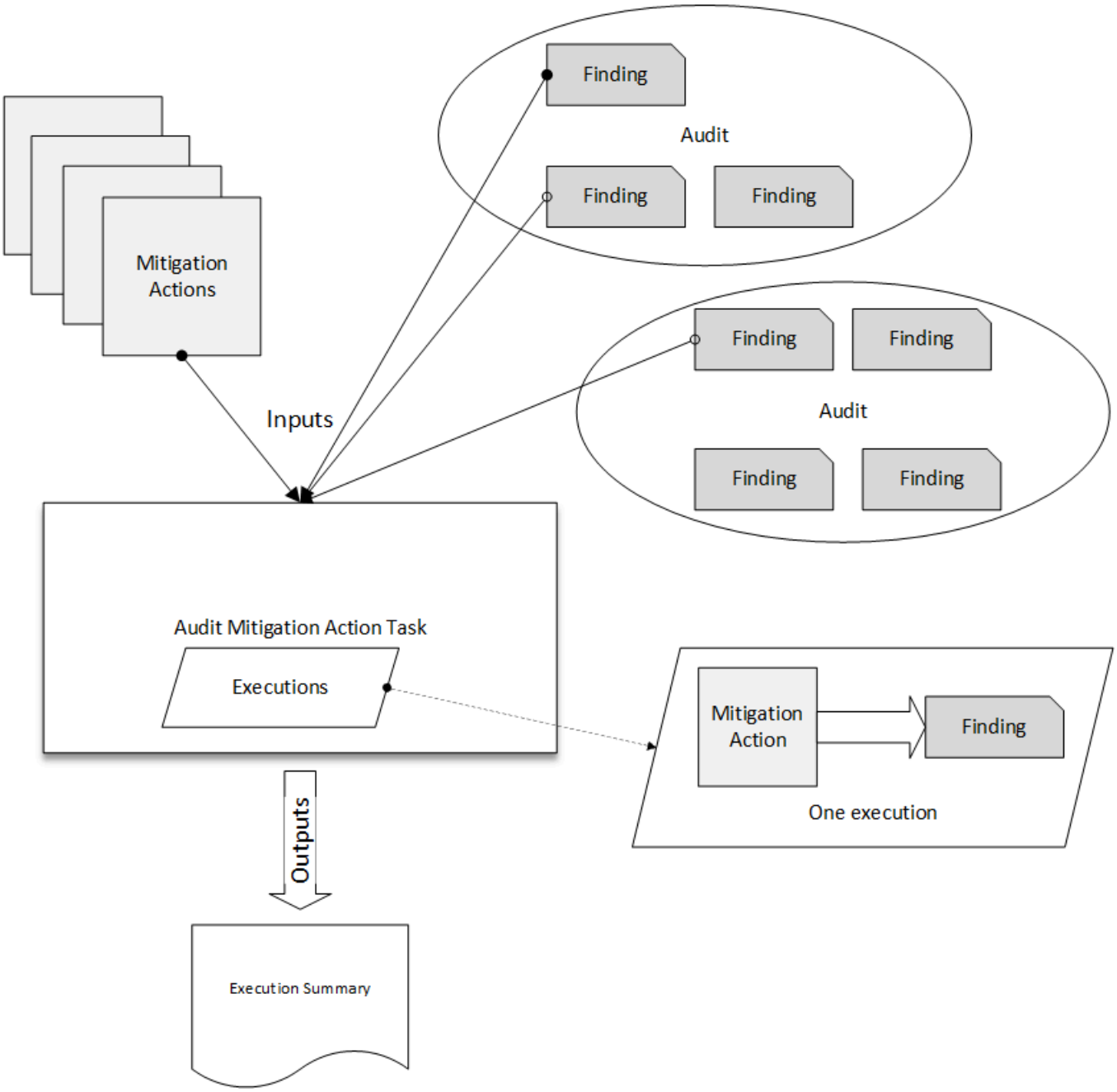
You can view the list of action executions and cancel an execution that has not yet been completed. Actions already performed as part of the canceled action execution are not rolled back. If you are applying multiple actions to a set of findings and one of those actions failed, the subsequent actions are skipped for that finding (but are still applied to other findings). The task status for the finding is FAILED. The `taskStatus` is set to failed if one or more of the actions failed when applied to the findings. Actions are applied in the order in which they are specified.

Each action execution applies a set of actions to a target. That target can be a list of findings or it can be all findings from an audit.

The following diagram shows how you can define an audit mitigation task that takes all findings from one audit and applies a set of actions to those findings. A single execution applies one action to one finding. The audit mitigation actions task outputs an execution summary.



The following diagram shows how you can define an audit mitigation task that takes a list of individual findings from one or more audits and applies a set of actions to those findings. A single execution applies one action to one finding. The audit mitigation actions task outputs an execution summary.



You can use the AWS IoT console or the AWS CLI to apply mitigation actions.

To use the AWS IoT console to apply mitigation actions by starting an action execution

1. Open the Audit results page in the AWS IoT console.
2. Choose the name for the audit to which you want to apply actions.
3. Choose **Start mitigation actions**. This button is not available if all of your checks are compliant.

4. In **Start a new mitigation action**, the task name defaults to the audit ID, but you can change it to something more meaningful.
5. For each type of check that had one or more noncompliant findings in the audit, you can choose one or more actions to apply. Only actions that are valid for the check type are displayed.

   > **Note**
   >
   > If you have not configured actions for your AWS account, the list of actions is empty. You can choose the **Create mitigation action** link to create one or more mitigation actions.

6. When you have specified all of the actions that you want to apply, choose **Start task**.

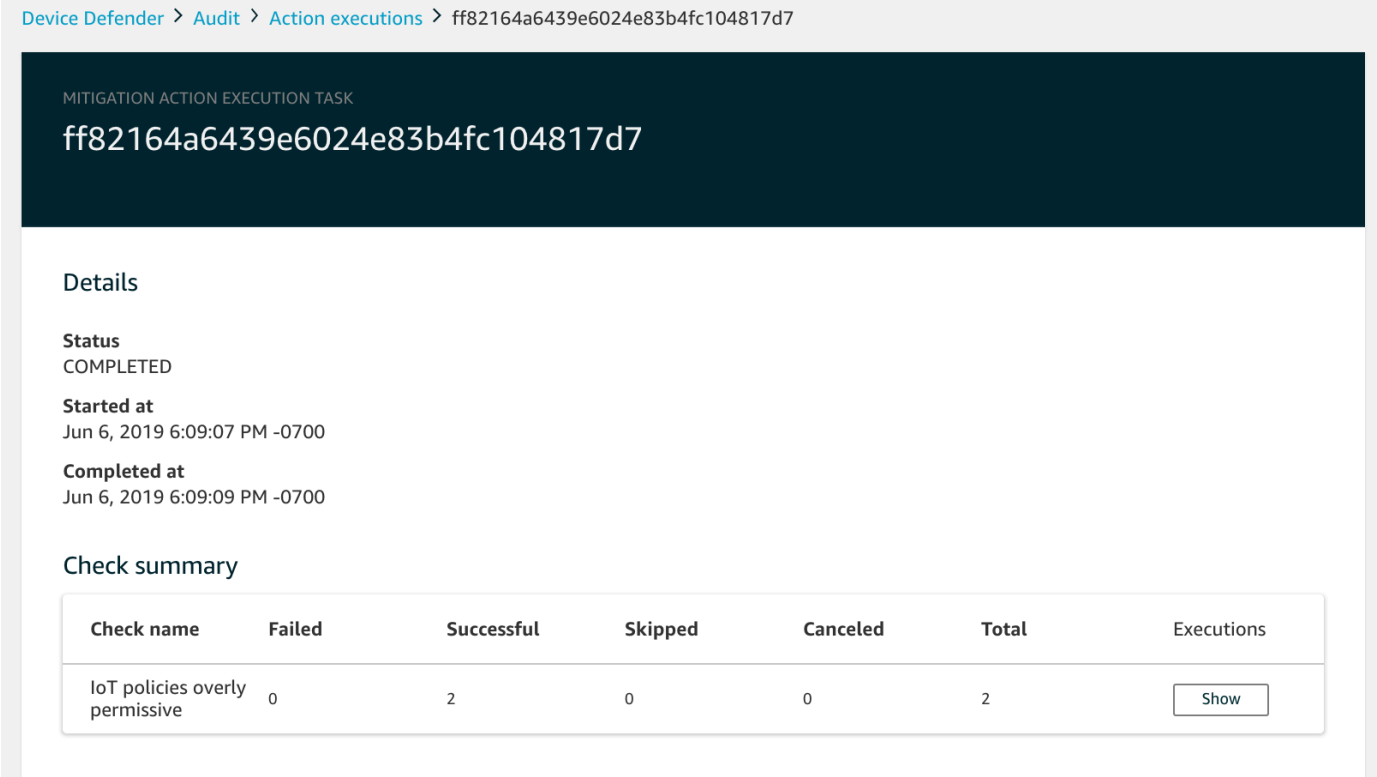
To use the AWS CLI to apply mitigation actions by starting an audit mitigation actions execution

1. If you want to apply actions to all findings for the audit, use the ListAuditTasks command to find the task ID.
2. If you want to apply actions to selected findings only, use the ListAuditFindings command to get the finding IDs.
3. Use the ListMitigationActions command and make note of the names of the mitigation actions that you want to apply.
4. Use the StartAuditMitigationActionsTask command to apply actions to the target. Make note of the task ID. You can use the ID to check the state of the action execution, review the details, or cancel it.

To use the AWS IoT console to view your action executions

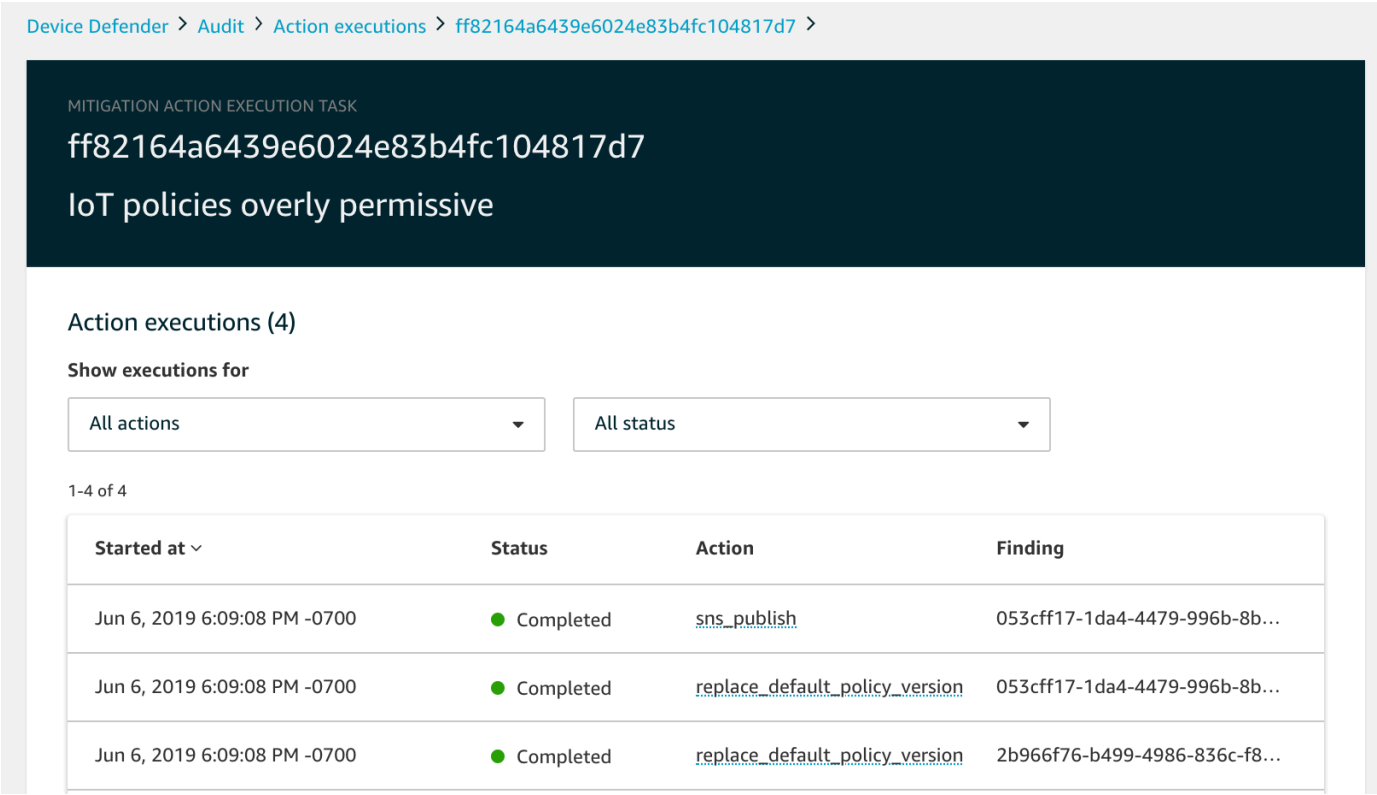
1. Open the Action tasks page in the AWS IoT console.

   A list of action tasks shows when each was started and the current status.
2. Choose the **Name** link to see details for the task. The details include all of the actions that are applied by the task, their target, and their status.



You can use the **Show executions for** filters to focus on types of actions or action states.

3. To see details for the task, in **Executions**, choose **Show**.



To use the AWS CLI to list your started tasks

1. Use ListAuditMitigationActionsTasks to view your audit mitigation actions tasks. You can provide filters to narrow the results. If you want to view details of the task, make note of the task ID.
2. Use ListAuditMitigationActionsExecutions to view execution details for a particular audit mitigation actions task.
3. Use DescribeAuditMitigationActionsTask to view details about the task, such as the parameters specified when it was started.

To use the AWS CLI to cancel a running audit mitigation actions task

1. Use the ListAuditMitigationActionsTasks command to find the task ID for the task whose execution you want to cancel. You can provide filters to narrow the results.
2. Use the ListDetectMitigationActionsExecutions command, using the task ID, to cancel your audit mitigation actions task. You cannot cancel tasks that have been completed. When you cancel a task, remaining actions are not applied, but mitigation actions that were already applied are not rolled back.

Permissions

For each mitigation action that you define, you must provide the role used to apply that action.

Permissions for mitigation actions

Action type	Permissions policy template	
UPDATE_DEVICE_CERTIFICATE	JSON `{ "Version" :"2012-10-17", "Statement":[ {`	

Action type	Permissions policy template	

	<pre>"Effect": "Allow",
	"Action":[
	"iot:UpdateCertificate"
	],
	"Resource":[
	"*"
	]
	}
	]
	}</pre>	

Action type	Permissions policy template	

UPDATE_CA_CERTIFICATE	JSON
	
	{
	"Version"
	:"2012-10-17",
	
	"Statement":[
	{
	
	"Effect":
	"Allow",
	
	"Action":[
	
	"iot:Upda
	teCACertificate"
	],
	
	"Resource":[
	
	"*"
	]
	}
	]
	}	

Action type	Permissions policy template	

ADD_THINGS_TO_THING_GROUP	JSON
	
	{
	"Version":"2012-10-17",
	"Statement":[
	{
	"Effect": "Allow",
	"Action":[
	"iot:ListPrincipalThings",
	"iot:AddThingToThingGroup"
	],
	"Resource":[
	"*"
	]
	}
	]
	}	

Action type	Permissions policy template	

REPLACE_DEFAULT_POLICY_VERSION	JSON
	
	{
	"Version":"2012-10-17",
	"Statement":[
	{
	"Effect": "Allow",
	"Action":[
	"iot:CreatePolicyVersion"
	],
	"Resource":[
	"*"
	]
	}
	]
	}	

Action type	Permissions policy template	
ENABLE_IOT_LOGGING	JSON	

```
{
  "Version"
:"2012-10-17",

  "Statement": [
    {
      "Effect":
 "Allow",
      "Action": [
        "iot:SetV
2LoggingOptions"
      ],
      "Resource":
 "*"
    },
    {
      "Effect":
 "Allow",
      "Action": [
        "iam:Pass
Role"
      ],
      "Resource
": "arn:aws:
iam::1234
56789012:role/
IoTLoggingRole"
    }
  ]
}
```

Action type	Permissions policy template	

PUBLISH_FINDING_TO_SNS	JSON
	
	{
	"Version"
	:"2012-10-17",
	
	"Statement":[
	{
	
	"Effect":
	"Allow",
	
	"Action":[
	
	"sns:Publish"
	],
	
	"Resource":[
	
	"arn:aws:sns: *us-east-1* :12345678
	9012: *example-t opic* "
	]
	}
	]
	}	

For all mitigation action types, use the following trust policy template:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:iot:*:123456789012::*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012:"
        }
      }
    }
  ]
}
```

Mitigation action commands

You can use these mitigation action commands to define a set of actions for your AWS account that you can later apply to one or more sets of audit findings. There are three command categories:

- Those used to define and manage actions.
- Those used to start and manage the application of those actions to Audit findings.
- Those used to start and manage the application of those actions to Detect alarms.

Mitigation action commands

Define and manage actions	Start and manage Audit execution	Start and manage Detect execution
CreateMitigationAction	CancelAuditMitigationActionsTask	CancelDetectMitigationActionsTask
DeleteMitigationAction	DescribeAuditMitigationActionsTask	DescribeDetectMitigationActionsTask

Define and manage actions	Start and manage Audit execution	Start and manage Detect execution
DescribeMitigationAction	ListAuditMitigationActionsTasks	ListDetectMitigationActionsTasks
ListMitigationActions	StartAuditMitigationActionsTask	StartDetectMitigationActionsTask
UpdateMitigationAction	ListAuditMitigationActionsExecutions	ListDetectMitigationActionsExecutions

Using AWS IoT Device Defender with other AWS services

Using AWS IoT Device Defender with devices running AWS IoT Greengrass

AWS IoT Greengrass provides pre-built integration with AWS IoT Device Defender to monitor device behaviors on an ongoing basis.

- Integrate Device Defender with AWS IoT Greengrass V1
- Integrate Device Defender with AWS IoT Greengrass V2

Using AWS IoT Device Defender with FreeRTOS and embedded devices

To use AWS IoT Device Defender on a FreeRTOS device, your device must have the FreeRTOS Embedded C SDK or the AWS IoT Device Defender library installed. The FreeRTOS Embedded C SDK includes the AWS IoT Device Defender library. For information about how to integrate AWS IoT Device Defender with your FreeRTOS devices, see the following demos:

- AWS IoT Device Defender for FreeRTOS standard metrics and custom metrics demos
- Using MQTT agent to submit metrics to AWS IoT Device Defender
- Using the MQTT core library to submit metrics to AWS IoT Device Defender

To use AWS IoT Device Defender on an embedded device without FreeRTOS, your device must have the AWS IoT Embedded C SDK or AWS IoT Device Defender library. The AWS IoT Embedded C SDK includes the AWS IoT Device Defender library. For information about how to integrate AWS IoT Device Defender with your embedded devices, see the following demos, AWS IoT Device Defender for AWS IoT Embedded SDK standard and custom metrics demos.

Using AWS IoT Device Defender with AWS IoT Device Management

You can use AWS IoT Device Management fleet indexing to index, search, and aggregate your AWS IoT Device Defender detect violations.

> **Note**
>
> The fleet indexing feature to support indexing AWS IoT Device Defender violations data is in preview release for AWS IoT Device Management and is subject to change.

- Managing fleet indexing
- Query syntax

Integration with AWS Security Hub CSPM

AWS Security Hub CSPM provides you with a comprehensive view of your security state in AWS and helps you check your environment against security industry standards and best practices. Security Hub CSPM collects security data from across AWS accounts, services, and supported third-party products. You can use Security Hub CSPM to analyze your security trends and identify the highest priority security issues.

With the AWS IoT Device Defender integration with Security Hub CSPM, you can send findings from AWS IoT Device Defender to Security Hub CSPM. Security Hub CSPM includes those findings in its analysis of your security posture.

Contents

- Enabling and configuring the integration
- How AWS IoT Device Defender sends findings to Security Hub CSPM
  - Types of findings that AWS IoT Device Defender sends
  - Latency for sending findings
  - Retrying when Security Hub CSPM isn't available
  - Updating existing findings in Security Hub CSPM
- Typical finding from AWS IoT Device Defender

- Stopping AWS IoT Device Defender from sending findings to Security Hub CSPM

Enabling and configuring the integration

Before you integrate AWS IoT Device Defender with Security Hub CSPM, you must first enable Security Hub CSPM. For information about how to enable Security Hub CSPM, see Setting up Security Hub in the *AWS Security Hub User Guide*.

After you enable both AWS IoT Device Defender and Security Hub CSPM, open the Integrations page in the Security Hub CSPM console, and then choose **Accept findings** for Audit, Detect, or both. AWS IoT Device Defender begins sending findings to Security Hub CSPM.

How AWS IoT Device Defender sends findings to Security Hub CSPM

In Security Hub CSPM, security issues are tracked as *findings*. Some findings come from issues that are detected by other AWS services or by third-party products.

Security Hub CSPM provides tools to manage findings from across all of these sources. You can view and filter lists of findings and view details for a finding. For more information, see Viewing findings in the *AWS Security Hub User Guide*. You can also track the status of an investigation into a finding. For more information, see Taking action on findings in the *AWS Security Hub User Guide*.

All findings in Security Hub CSPM use a standard JSON format called the *AWS Security Finding Format (ASFF)*. The ASFF includes details about the source of the issue, the affected resources, and the current status of the finding. For more information about ASFF, see AWS Security Finding Format (ASFF) in the *AWS Security Hub User Guide*.

AWS IoT Device Defender is one of the AWS services that sends findings to Security Hub CSPM.

Types of findings that AWS IoT Device Defender sends

After you enable the Security Hub CSPM integration, AWS IoT Device Defender Audit sends the findings it generates (called *check summaries*) to Security Hub CSPM. Check summaries are general information for a specific audit check type and a specific audit task. For more information, see Audit checks.

AWS IoT Device Defender Audit sends finding updates to Security Hub CSPM for both Audit Check Summaries and Audit Findings in each Audit task. If all resources found in Audit Checks are compliant, or an Audit Task is canceled, Audit updates the Check Summaries in Security Hub CSPM to an ARCHIVED record state. If a resource was reported as non-compliant for an Audit Check, but

was reported as compliant in the last Audit task, Audit changes it to compliant and also updates the finding in Security Hub CSPM to an ARCHIVED record state.

AWS IoT Device Defender Detect sends violation findings to Security Hub CSPM. These violation findings include machine learning (ML), statistical, and static behaviors.

To send the findings to Security Hub CSPM, AWS IoT Device Defender uses the AWS Security Finding Format (ASFF). In ASFF, the `Types` field provides the finding type. Findings from AWS IoT Device Defender can have the following values for `Types`.

Unusual behaviors

The finding type for conflicting MQTT client IDs and device certificate shared checks, and the finding type for Detect.

Software and Configuration Check/Vulnerabilities

The finding type for all other Audit checks.

Latency for sending findings

When AWS IoT Device Defender Audit creates a new finding, it's immediately sent to Security Hub CSPM after the audit task completes. The latency depends on the volume of the findings generated in the audit task. Security Hub CSPM typically receives the findings within one hour.

AWS IoT Device Defender Detect sends findings for violations in near real time. After a violation goes into or out of alarm (meaning the alarm is created or deleted), the corresponding Security Hub CSPM finding is immediately created or archived.

Retrying when Security Hub CSPM isn't available

If Security Hub CSPM isn't available, AWS IoT Device Defender Audit and AWS IoT Device Defender Detect retry sending the findings until they're received.

Updating existing findings in Security Hub CSPM

After an AWS IoT Device Defender Audit finding is sent to Security Hub CSPM, you can identify it by checked resource identifier and audit check type. If a new audit finding is generated with a subsequent audit task for the same resource and audit check, AWS IoT Device Defender Audit sends updates to reflect additional observations of the finding activity to Security Hub CSPM. If no additional audit finding is generated with a subsequent audit task for the same resource and audit

check, the resource changes to compliant with the audit check. AWS IoT Device Defender Audit then archives the findings in Security Hub CSPM.

AWS IoT Device Defender Audit also updates check summaries in Security Hub CSPM. If there are non-compliant resources found in an audit check or the check fails, the status of the Security Hub CSPM finding becomes active. Otherwise, AWS IoT Device Defender Audit archives the finding in Security Hub CSPM.

AWS IoT Device Defender Detect creates a Security Hub CSPM finding when there's a violation (for example, in-alarm). That finding is updated only if one of the following criteria is met:

- The finding is expiring soon in Security Hub CSPM so AWS IoT Device Defender sends an update to keep the finding current. Findings are deleted 90 days after the most recent update or 90 days after the creation date if no update occurs. For more information, see Security Hub CSPM quotas in the *AWS Security Hub User Guide*.
- The corresponding violation goes out of alarm, so AWS IoT Device Defender updates its finding status to ARCHIVED.

Typical finding from AWS IoT Device Defender

AWS IoT Device Defender uses the AWS Security Finding Format (ASFF) to send findings to Security Hub CSPM.

The following example shows a typical finding from Security Hub CSPM for an audit finding. The `ReportType` in `ProductFields` is `AuditFinding`.

```
  {
  "SchemaVersion": "2018-10-08",
  "Id": "336757784525/IOT_POLICY/policyexample/1/IOT_POLICY_OVERLY_PERMISSIVE_CHECK/
ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS",
  "ProductArn": "arn:aws:securityhub:us-west-2::product/aws/iot-device-defender-audit",
  "ProductName": "IoT Device Defender - Audit",
  "CompanyName": "AWS",
  "Region": "us-west-2",
  "GeneratorId": "1928b87ab338ee2f541f6fab8c41c4f5",
  "AwsAccountId": "123456789012",
  "Types": [
    "Software and Configuration Check/Vulnerabilities"
  ],
```

```
  "CreatedAt": "2022-11-06T22:11:40.941Z",
  "UpdatedAt": "2022-11-06T22:11:40.941Z",
  "Severity": {
    "Label": "CRITICAL",
    "Normalized": 90
  },
  "Title": "IOT_POLICY_OVERLY_PERMISSIVE_CHECK:
 ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS",
  "Description": "IOT_POLICY policyexample:1 is reported as non-compliant for
 IOT_POLICY_OVERLY_PERMISSIVE_CHECK by Audit task 9f71b6e90cfb57d4ac671be3a4898e6a.
 The non-compliant reason is Policy allows broad access to IoT data plane actions:
 [iot:Connect].",
  "SourceUrl": "https://us-west-2.console.aws.amazon.com/iot/home?region=us-west-2#/
policy/policyexample",
  "ProductFields": {
    "CheckName": "IOT_POLICY_OVERLY_PERMISSIVE_CHECK",
    "TaskId": "9f71b6e90cfb57d4ac671be3a4898e6a",
    "TaskType": "ON_DEMAND_AUDIT_TASK",
    "PolicyName": "policyexample",
    "IsSuppressed": "false",
    "ReasonForNonComplianceCode": "ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS",
    "ResourceType": "IOT_POLICY",
    "FindingId": "1928b87ab338ee2f541f6fab8c41c4f5",
    "PolicyVersionId": "1",
    "ReportType": "AuditFinding",
    "TaskStartTime": "1667772700554",
    "aws/securityhub/FindingId": "arn:aws:securityhub:us-west-2::product/
aws/iot-device-defender-audit/336757784525/IOT_POLICY/policyexample/1/
IOT_POLICY_OVERLY_PERMISSIVE_CHECK/ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS",
    "aws/securityhub/ProductName": "IoT Device Defender - Audit",
    "aws/securityhub/CompanyName": "AWS"
  },
  "Resources": [
    {
      "Type": "AwsIotPolicy",
      "Id": "policyexample",
      "Partition": "aws",
      "Region": "us-west-2",
      "Details": {
        "Other": {
          "PolicyVersionId": "1"
        }
      }
    }
```

```
  ],
  "WorkflowState": "NEW",
  "Workflow": {
    "Status": "NEW"
  },
  "RecordState": "ACTIVE",
  "FindingProviderFields": {
    "Severity": {
      "Label": "CRITICAL"
    },
    "Types": [
      "Software and Configuration Check/Vulnerabilities"
    ]
  }
}
```

The following example shows a finding from Security Hub CSPM for an audit check summary. The `ReportType` in `ProductFields` is `CheckSummary`.

```
  {
  "SchemaVersion": "2018-10-08",
  "Id": "615243839755/SCHEDULED_AUDIT_TASK/daily_audit_schedule_checks/
DEVICE_CERTIFICATE_KEY_QUALITY_CHECK",
  "ProductArn": "arn:aws:securityhub:us-east-1::product/aws/iot-device-defender-audit",
  "ProductName": "IoT Device Defender - Audit",
  "CompanyName": "AWS",
  "Region": "us-east-1",
  "GeneratorId": "f3021945485adf92487c273558fcaa51",
  "AwsAccountId": "123456789012",
  "Types": [
    "Software and Configuration Check/Vulnerabilities/CVE"
  ],
  "CreatedAt": "2022-10-18T14:20:13.933Z",
  "UpdatedAt": "2022-10-18T14:20:13.933Z",
  "Severity": {
    "Label": "CRITICAL",
    "Normalized": 90
  },
  "Title": "DEVICE_CERTIFICATE_KEY_QUALITY_CHECK Summary: Completed with 2 non-
compliant resources",
```

```
  "Description": "Task f3021945485adf92487c273558fcaa51 of weekly scheduled Audit
 daily_audit_schedule_checks completes. 2 non-cimpliant resources are found for
 DEVICE_CERTIFICATE_KEY_QUALITY_CHECK out of 1000 resources in the account. The
 percentage of non-compliant resources is 0.2%.",
  "SourceUrl": "https://us-east-1.console.aws.amazon.com/iot/home?region=us-east-1#/dd/
audit/results/f3021945485adf92487c273558fcaa51/DEVICE_CERTIFICATE_KEY_QUALITY_CHECK",
  "ProductFields": {
    "TaskId": "f3021945485adf92487c273558fcaa51",
    "TaskType": "SCHEDULED_AUDIT_TASK",
    "ScheduledAuditName": "daily_audit_schedule_checks",
    "CheckName": "DEVICE_CERTIFICATE_KEY_QUALITY_CHECK",
    "ReportType": "CheckSummary",
    "CheckRunStatus": "COMPLETED_NON_COMPLIANT",
    "NonComopliantResourcesCount": "2",
    "SuppressedNonCompliantResourcesCount": "1",
    "TotalResourcesCount": "1000",
    "aws/securityhub/FindingId": "arn:aws:securityhub:us-east-1::product/aws/
iot-device-defender-audit/615243839755/SCHEDULED/daily_audit_schedule_checks/
DEVICE_CERTIFICATE_KEY_QUALITY_CHECK",
    "aws/securityhub/ProductName": "IoT Device Defender - Audit",
    "aws/securityhub/CompanyName": "AWS"
  },
  "Resources": [
    {
      "Type": "AwsIotAuditTask",
      "Id": "f3021945485adf92487c273558fcaa51",
      "Region": "us-east-1"
    }
  ],
  "WorkflowState": "NEW",
  "Workflow": {
    "Status": "NEW"
  },
  "RecordState": "ACTIVE",
  "FindingProviderFields": {
    "Severity": {
      "Label": "CRITICAL"
    },
    "Types": [
      "Software and Configuration Check/Vulnerabilities/CVE"
    ]
  }
}
```

The following example shows a typical finding from Security Hub CSPM for an AWS IoT Device Defender Detect violation.


```
  {
  "SchemaVersion": "2018-10-08",
  "Id": "e92a782593c6f5b1fc7cb6a443dc1a12",
  "ProductArn": "arn:aws:securityhub:us-east-1::product/aws/iot-device-defender-
detect",
  "ProductName": "IoT Device Defender - Detect",
  "CompanyName": "AWS",
  "Region": "us-east-1",
  "GeneratorId": "arn:aws:iot:us-east-1:123456789012:securityprofile/
MySecurityProfile",
  "AwsAccountId": "123456789012",
  "Types": [
    "Unusual Behaviors"
  ],
  "CreatedAt": "2022-11-09T22:45:00Z",
  "UpdatedAt": "2022-11-09T22:45:00Z",
  "Severity": {
    "Label": "MEDIUM",
    "Normalized": 40
  },
  "Title": "Registered thing MyThing is in alarm for STATIC behavior MyBehavior.",
  "Description": "Registered thing MyThing violates STATIC behavior MyBehavior of
 security profile MySecurityProfile. Violation was triggered because the device did not
 conform to aws:num-disconnects less-than 1.",
  "SourceUrl": "https://us-east-1.console.aws.amazon.com/iot/home?region=us-east-1#/dd/
securityProfile/MySecurityProfile?tab=violations",
  "ProductFields": {
    "ComparisonOperator": "less-than",
    "BehaviorName": "MyBehavior",
    "ViolationId": "e92a782593c6f5b1fc7cb6a443dc1a12",
    "ViolationStartTime": "1668033900000",
    "SuppressAlerts": "false",
    "ConsecutiveDatapointsToAlarm": "1",
    "ConsecutiveDatapointsToClear": "1",
    "DurationSeconds": "300",
    "Count": "1",
    "MetricName": "aws:num-disconnects",
```

```
    "BehaviorCriteriaType": "STATIC",
    "ThingName": "MyThing",
    "SecurityProfileName": "MySecurityProfile",
    "aws/securityhub/FindingId": "arn:aws:securityhub:us-east-1::product/aws/iot-
device-defender-detect/e92a782593c6f5b1fc7cb6a443dc1a12",
    "aws/securityhub/ProductName": "IoT Device Defender - Detect",
    "aws/securityhub/CompanyName": "AWS"
  },
  "Resources": [
    {
      "Type": "AwsIotRegisteredThing",
      "Id": "MyThing",
      "Region": "us-east-1",
      "Details": {
        "Other": {
          "SourceUrl": "https://us-east-1.console.aws.amazon.com/iot/home?region=us-
east-1#/thing/MyThing?tab=violations",
          "IsRegisteredThing": "true",
          "ThingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyThing"
        }
      }
    }
  ],
  "WorkflowState": "NEW",
  "Workflow": {
    "Status": "NEW"
  },
  "RecordState": "ACTIVE",
  "FindingProviderFields": {
    "Severity": {
      "Label": "MEDIUM"
    },
    "Types": [
      "Unusual Behaviors"
    ]
  }
}
```

Stopping AWS IoT Device Defender from sending findings to Security Hub CSPM

To stop sending findings to Security Hub CSPM, you can use either the Security Hub CSPM console or the API.

For more information, see Disabling and enabling the flow of findings from an integration (console) or Disabling the flow of findings from an integration (Security Hub CSPM API, AWS CLI) in the *AWS Security Hub User Guide*.

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 through the called service 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.

There are three resources AWS IoT Device Defender acesses from you that can be effected by the confused deptuy security issue, running audits, sending SNS notifications for security profile violations and running mitigation actions. For each of these actions, the values for `aws:SourceArn` must be as follows:

- For resources passed in UpdateAccountAuditConfiguration API (RoleArn and notificationTarget RoleArn attributes) you should scope down the resource policy by using `aws:SourceArn` as `arn:`*`arnPartition`*`:iot:`*`region`*`:`*`accountId`*`:.`
- For resources passed in CreateMitigationAction API (The RoleArn attribute) you should scope down the resource policy by using `aws:SourceArn` as `arn:`*`arnPartition`*`:iot:`*`region`*`:`*`accountId`*`:mitigationaction/`*`mitigationActionName`*.
- For resources passed in CreateSecurityProfile API (the alertTargets attribute) you should scope down the resource policy by using `aws:SourceArn` as `arn:`*`arnPartition`*`:iot:`*`region`*`:`*`accountId`*`:securityprofile/`*`securityprofileName`*.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:*:`*`123456789012`*`:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in AWS IoT Device Defender to prevent the confused deputy problem.

JSON

```
 {
"Version":"2012-10-17",
"Statement": {
  "Sid": "ConfusedDeputyPreventionExamplePolicy",
  "Effect": "Allow",
  "Principal": {
    "Service": "iot.amazonaws.com"
  },
  "Action": "sts:AssumeRole",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "arn:aws:iot:*:123456789012::*"
    },
    "StringEquals": {
      "aws:SourceAccount": "123456789012:"
    }
  }
}
}
```

Security best practices for device agents

Least Privilege

The agent process should be granted only the minimum permissions required to perform its duties.

Basic mechanisms

- Agent should be run as non-root user.
- Agent should run as a dedicated user, in its own group.
- User/groups should be granted read-only permissions on the resources required to gather and transmit metrics.
- Example: read-only on /proc /sys for the sample agent.
- For an example of how to set up a process to run with reduced permissions, see the setup instructions that are included with the Python sample agent.

There are a number of well-known Linux mechanisms that can help you further restrict or isolate your agent process:

Advanced mechanisms

- CGroups
- SELinux
- Chroot
- Linux Namespaces

Operational Resiliency

An agent process must be resilient to unexpected operational errors and exceptions and must not crash or exit permanently. The code needs to gracefully handle exceptions and, as a precaution, it must be configured to automatically restart in case of unexpected termination (for example, due to system restarts or uncaught exceptions).

Least Dependencies

An agent must use the least possible number of dependencies (that is, third-party libraries) in its implementation. If use of a library is justified due to the complexity of a task (for example, transport layer security), use only well-maintained dependencies and establish a mechanism to keep them up to date. If the added dependencies contain functionality not used by the agent and active by default (for example, opening ports, domain sockets), disable them in your code or by means of the library's configuration files.

Process Isolation

An agent process must only contain functionality required for performing device metric gathering and transmission. It must not piggyback on other system processes as a container or implement functionality for other out of scope use cases. In addition, the agent process must

refrain from creating inbound communication channels such as domain socket and network service ports that would allow local or remote processes to interfere with its operation and impact its integrity and isolation.

Stealthiness

An agent process must not be named with keywords such as security, monitoring, or audit indicating its purpose and security value. Generic code names or random and unique-per-device process names are preferred. The same principle must be followed in naming the directory in which the agent's binaries reside and any names and values of process arguments.

Least Information Shared

Any agent artifacts deployed to devices must not contain sensitive information such as privileged credentials, debugging and dead code, or inline comments or documentation files that reveal details about server-side processing of agent-gathered metrics or other details about backend systems.

Transport Layer Security

To establish TLS secure channels for data transmission, an agent process must enforce all client-side validations, such as certificate chain and domain name validation, at the application level, if not enabled by default. Furthermore, an agent must use a root certificate store that contains trusted authorities and does not contain certificates belonging to compromised certificate issuers.

Secure Deployment

Any agent deployment mechanism, such as code push or sync and repositories containing its binaries, source code and any configuration files (including trusted root certificates), must be access-controlled to prevent unauthorized code injection or tampering. If the deployment mechanism relies on network communication, then use cryptographic methods to protect the integrity of deployment artifacts in transit.

Further Reading

- Security in AWS IoT Device Defender
- Understanding the AWS IoT Security Model
- Redhat: A Bite of Python
- 10 common security gotchas in Python and how to avoid them
- What Is Least Privilege & Why Do You Need It?
- OWASP Embedded Security Top 10

- OWASP IoT Project

AWS IoT Device Defender troubleshooting guide

Help us improve this topic

Let us know what would help make it better

General

Q: Are there any prerequisites for using AWS IoT Device Defender?

A: If you want to use device-reported metrics, you must first deploy an agent on your AWS IoT connected devices or device gateways. Devices must provide a consistent client identifier or thing name.

Audit

Q: I enabled a check and my audit has been showing "In-Progress" for a long time. Is something wrong? When can I expect results?

A: When a check is enabled, data collection starts immediately. However, if your account has a large amount of data to collect (for example, certificates, things, or policies), the results of the check might not be available for some time after you have enabled it.

Detect

Q: How do I know the thresholds to set in an AWS IoT Device Defender security profile behavior?

A: Start by creating a security profile behavior with low thresholds and attach it to a thing group that contains a representative set of devices. You can use AWS IoT Device Defender to view the current metrics, and then fine-tune the device behavior thresholds to match your use case.

Q: I created a behavior, but it is not triggering a violation when I expect it to. How should I fix it?

A: When you define a behavior, you are specifying how you expect your device to behave normally. For example, if you have a security camera that only connects to one central server on TCP port 8888, you don't expect it to make any other connections. To be alerted if the camera makes a connection on another port, you define a behavior like this:

```
{
  "name": "Listening TCP Ports",
  "metric": "aws:listening-tcp-ports",
  "criteria": {
    "comparisonOperator": "in-port-set",
    "value": {
      "ports": [ 8888 ]
    }
  }
}
```

If the camera makes a TCP connection on TCP port 443, the device behavior would be violated and an alert would be triggered.

Q: One or more of my behaviors are in violation. How do I clear the violation?

A: Alarms clear after the device returns to expected behavior, as defined in the behavior profiles. Behavior profiles are evaluated upon receipt of metrics data for your device. If the device doesn't publish any metrics for more than two days, the violation event is set to `alarm-invalidated` automatically.

Q: I deleted a behavior that was in violation, but how do I stop the alerts?

A: Deleting a behavior stops all future violations and alerts for that behavior. Earlier alerts must be drained from your notification mechanism. When you delete a behavior, the record of violations of that behavior is retained for the same time period as all other violations in your account.

Device Metrics

Q: I'm submitting metrics reports that I know violate my behaviors, but no violations are being triggered. What's wrong?

A: Check that your metrics reports are being accepted by subscribing to the following MQTT topics:

```
$aws/things/THING_NAME/defender/metrics/FORMAT/rejected
$aws/things/THING_NAME/defender/metrics/FORMAT/accepted
```

`THING_NAME` is the name of the thing reporting the metric and `FORMAT` is either "JSON" or "CBOR," depending on the format of the metrics report submitted by the thing.

After you have subscribed, you should receive messages on these topics for each metric report submitted. A `rejected` message indicates that there was a problem parsing the metric report. An error message is included in the message payload to help you correct any errors in your metric report. An `accepted` message indicates that the metric report was parsed properly.

Q: What happens if I send an empty metric in my metric report?

A: An empty list of ports or IP addresses is always considered in conformity with the corresponding behavior. If the corresponding behavior was in violation, the violation is cleared.

Q: Why do my device metric reports contain messages for devices that aren't in the AWS IoT registry?

If you have one or more security profiles attached to all things or to all unregistered things, AWS IoT Device Defender includes metrics from unregistered things. If you want to exclude metrics from unregistered things, you can attach the profiles to all registered devices instead of all devices.

Q: I'm not seeing messages from one or more unregistered devices even though I applied a security profile to all unregistered devices or all devices. How can I fix it?

Verify that you are sending a well-formed metrics report using one of the supported formats. For information, see Device metrics document specification. Verify that the unregistered devices are using a consistent client identifier or thing name. If the thing name contains control characters or is longer than 128 bytes of UTF-8 encoded characters, messages reported by devices are rejected .

Q: What happens if an unregistered device is added to the registry or a registered device becomes unregistered?

A: If a device is added to or removed from the registry:

- You see two separate violations for the device (one under its registered thing name, one under its unregistered identity) if it continues to publish metrics for violations. Active violations for the old identity stop appearing after two days, but are available in violations history for up to 14 days.

Q: Which value should I supply in the report ID field of my device metrics report?

A: Use a unique value for each metric report, expressed as a positive integer. A common practice is to use a Unix epoch timestamp.

Q: Should I create a dedicated MQTT connection for AWS IoT Device Defender metrics?

A: A separate MQTT connection is not required.

Q: Which client ID should I use when connecting to publish device metrics?

For devices (things) that are in the AWS IoT registry, use the registered thing name. For devices that are not in the AWS IoT registry, use a consistent identifier when you connect to AWS IoT. This practice helps match the violations to the thing name.

Q: Can I publish metrics for a device with a different client ID?

It is possible to publish metrics on behalf of another thing. You can do this by publishing the metrics to the AWS IoT Device Defender reserved topic for that device. For example, `Thing-1` would like to publish metrics for itself and also on behalf of `Thing-2`. `Thing-1` collects its own metrics and publishes them on the MQTT topic:

```
$aws/things/Thing-1/defender/metrics/json
```

`Thing-1` then obtains metrics from `Thing-2` and publishes those metrics on the MQTT topic:

```
$aws/things/Thing-2/defender/metrics/json
```

Q: How many security profiles and behaviors can I have in my account?

A: See AWS IoT Device Defender Endpoints and Quotas.

Q: What does a prototypical target role for an alert target look like?

A: A role that allows AWS IoT Device Defender to publish alerts on an alert target (SNS topic) requires two things:

- A trust relationship that specifies iot.amazonaws.com as the trusted entity.
- An attached policy that grants AWS IoT permission to publish on a specified SNS topic. For example:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:us-east-1:123456789012:example-topic"
        }
```

```
    ]
}
```

- If the SNS topic used for publishing alerts is an encrypted topic, then along with the permission to publish to SNS topic, AWS IoT must be granted two more permissions. For example:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "sns:Publish",
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": "arn:aws:sns:us-east-1:123456789012:example-topic"
       }
    ]
}
```

Q: My metric report submission with a custom metric type `number` fails with the error message `Malformed metrics report`. What's wrong?

A: The type `number` only takes a single metric value as an input, but while submitting the metrics value in the DeviceMetrics report, it must be passed as an array with a single value. Make sure you're submitting the metric value as an array.

Error payload:

```
{"header":{"report_id":12334567,"version":"1.0"},"metrics":{"network_stats":
{"bytes_in":30680,"bytes_out":10652,"packets_in":113,"packets_out":118}},"custom_metrics":
{"my_custom_metric":{"number":0}}}
```

Error message:

```
{"thingName":"myThing","status":"REJECTED","statusDetails":
{"ErrorCode":"InvalidPayload","ErrorMessage":"Malformed metrics
 report"},"timestamp":1635802047699}
```

No-error payload:


```
{"header":{"report_id":12334567,"version":"1.0"},"metrics":{"network_stats":
{"bytes_in":30680,"bytes_out":10652,"packets_in":113,"packets_out":118}},"custom_metrics":
{"my_custom_metric":[{"number":0}]}}
```


Response:


```
{"thingName":"myThing","12334567":1635800375,"status":"ACCEPTED","timestamp":1635801636023}
```

Security in AWS IoT Device Defender

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS IoT Device Defender, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS IoT Device Defender. The following topics show you how to configure AWS IoT Device Defender to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS IoT Device Defender resources. To learn more about security in AWS IoT Core, see the security chapter in the *AWS IoT Core Developer Guide*

Topics

- Data protection in AWS IoT Device Defender
- Identity and access management for AWS IoT Device Defender
- Compliance validation for AWS IoT Device Defender
- Resilience in AWS IoT Device Defender

Data protection in AWS IoT Device Defender

The AWS shared responsibility model applies to data protection in AWS IoT Device Defender. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this

infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS IoT Device Defender 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.

Identity and access management for AWS IoT Device Defender

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 IoT Device Defender resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS IoT Device Defender works with IAM
- Identity-based policy examples for AWS IoT Device Defender
- Troubleshooting AWS IoT Device Defender identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS IoT Device Defender identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS IoT Device Defender works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS IoT Device Defender)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you

don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For

more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS IoT Device Defender works with IAM

Before you use IAM to manage access to AWS IoT Device Defender, learn what IAM features are available to use with AWS IoT Device Defender.

IAM features you can use with AWS IoT Device Defender

IAM feature	AWS IoT Device Defender support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial

IAM feature	AWS IoT Device Defender support
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how AWS IoT Device Defender and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS IoT Device Defender

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS IoT Device Defender

To view examples of AWS IoT Device Defender identity-based policies, see Identity-based policy examples for AWS IoT Device Defender.

Resource-based policies within AWS IoT Device Defender

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified

principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS IoT Device Defender

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS IoT Device Defender actions, see in the *Service Authorization Reference*.

Policy actions in AWS IoT Device Defender use the following prefix before the action:

```
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      ":action1",
      ":action2"
         ]
```

To view examples of AWS IoT Device Defender identity-based policies, see Identity-based policy examples for AWS IoT Device Defender.

Policy resources for AWS IoT Device Defender

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS IoT Device Defender resource types and their ARNs, see in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see .

To view examples of AWS IoT Device Defender identity-based policies, see Identity-based policy examples for AWS IoT Device Defender.

Policy condition keys for AWS IoT Device Defender

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS IoT Device Defender condition keys, see in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see .

To view examples of AWS IoT Device Defender identity-based policies, see Identity-based policy examples for AWS IoT Device Defender.

ACLs in AWS IoT Device Defender

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AWS IoT Device Defender

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AWS IoT Device Defender

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for AWS IoT Device Defender

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS IoT Device Defender

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS IoT Device Defender functionality. Edit service roles only when AWS IoT Device Defender provides guidance to do so.

Service-linked roles for AWS IoT Device Defender

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS IoT Device Defender

By default, users and roles don't have permission to create or modify AWS IoT Device Defender resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS IoT Device Defender, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for AWS IoT Device Defender in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the AWS IoT Device Defender console
- Allow users to view their own permissions

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS IoT Device Defender resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API

operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS IoT Device Defender console

To access the AWS IoT Device Defender console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT Device Defender resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS IoT Device Defender console, also attach the AWS IoT Device Defender *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting AWS IoT Device Defender identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT Device Defender and IAM.

Topics

- I am not authorized to perform an action in AWS IoT Device Defender
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS IoT Device Defender resources

I am not authorized to perform an action in AWS IoT Device Defender

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: :GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS IoT Device Defender.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS IoT Device Defender. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS IoT Device Defender 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 IoT Device Defender supports these features, see How AWS IoT Device Defender works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS IoT Device Defender

Our new AWS sign-up experience is not designed for regulated workloads. If you're using our new AWS sign-up experience, but you want to use AWS for regulated workloads, you can sign up for AWS (advanced) or activate advanced features for your AWS environment.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS IoT Device Defender

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.

In addition to the AWS global infrastructure, AWS IoT Device Defender offers several features to help support your data resiliency and backup needs.

Document history for the AWS IoT Device Defender User Guide

The following table describes the documentation releases for AWS IoT Device Defender.

Change	Description	Date
AWS IoT Device Defender detect feature availability change	The AWS IoT Device Defender detect feature will no longer be available to new customers starting August 31, 2026. Existing customers can continue using the feature. For more information, see AWS IoT Device Defender detect feature availability change.	June 30, 2026
Generally Available	*This is the initial public release of AWS IoT Device Defender*.	August 2, 2023
AWS IoT Device Defender now supports monitoring of device disconnect durations	AWS IoT Device Defender Rules Detect now supports a new disconnect duration metric to monitor the duration of disconnect of each device. With this additional metric, you can track how long a device has been disconnected to learn whether it is operating as expected. You can also configure alarms at predefine d threshold levels and be alerted in the case of persistent device connectiv	July 20, 2023

	ity issues. For documenta tion, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	
AWS IoT Device Defender Audit feature identifies potential misconfiguration in IoT Policies	Identify flaws, troubleshoot issues, and take the necessary corrective actions using Audit feature. This new feature also helps in identifying IoT policies with permissive allow statements where devices could get access to unintende d resources. It also inspects for use of MQTT wildcards in deny statements that could potentially be circumvented by devices when replacing wildcards with specific strings. For more information, refer to Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*	December 6, 2022

AWS IoT Device Defender ML Detect Custom Metrics and Dimensions support	AWS IoT Device Defender now supports a new audit check for revoked intermediate Certificate Authority (CA). If a CA revokes an intermediate CA because it is potentially compromised, then all certificates issued by that intermediate CA are also potentially compromised and invalid. This new audit check identifies active device certificates issued by a revoked intermediate CA, and helps customers review and replace these active device certificates. For more information, refer to Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*	November 10, 2022

AWS IoT Device Defender ML Detect Custom Metrics and Dimensions support	ML Detect now supports monitoring of custom metrics, allowing you to evaluate operational health parameter s that are unique to your fleet. Besides setting static alarms manually with Rules Detect, you can now use machine learning to automatically learn your fleet's expected behaviors on custom metrics. Further, with the new Dimensions filter support for ML Detect, you can define attributes to evaluate more precise metrics in your ML security profile. Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*	September 14, 2022
AWS IoT Device Management and AWS IoT Device Defender device metrics support via ListMetricValues API	Access historical device-si de, cloud-side, and custom metrics from connected devices that belong to a security profile using ListMetri cValues API. In addition to viewing the data in the AWS IoT management console, you now have the flexibility to programmatically monitor and build your own visualiza tion. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*	April 5, 2022

AWS IoT Device Defender now supports Detect alarm verification states	Verify an alarm based on their investigation of detected behavior anomalies. They can verify an alarm as True positive, Benign positive, False positive, or Unknown and provide a description of their verification. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	September 24, 2021
AWS IoT Device Defender Audit One-Click release	Audit One-Click makes it easy for AWS IoT Core customers to improve their security baseline by making it possible to start auditing their account and IoT devices against security best practices with a single click. Audit One-Click allows customers to turn on an AWS IoT Device Defender audit with preset configurations including enabling all available audit checks and a daily audit schedule. It also provides contextual explanations for the benefits of regular security audits. Audit One-Click is only available from the AWS IoT console. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	September 22, 2021

AWS IoT Device Defender CloudFormation support	AWS IoT Device Defender now supports CloudFormation for creating and configuri ng AWS IoT Device Defender resources such as scheduled audits and Security Profiles in a secure, efficient, and repeatable way. To learn more about the CloudForm ation resource types AWS IoT Device Defender supports, visit AWS IoT resource type reference.	March 5, 2021
AWS IoT Device Defender adds support for custom metrics	Use AWS IoT Device Defender to monitor operational health metrics that are unique to your fleet or use case. The alerts can be viewed in the Device Defender console or shared through AWS Simple Notification Service (SNS). For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	December 15, 2020

AWS IoT Device Defender launches Audit Finding Suppression	The Audit Finding Suppression feature allows you to choose which audit findings you want to see and turn off non-compliant findings for specific resources. In addition, you can configure audit finding suppressions for a defined period of time or indefinitely. For documentation, see Audit in the *AWS IoT Device Defender Developer Guide*.	August 12, 2020
AWS IoT Device Defender now supports Dimensions for topic-based metric monitoring	The Dimensions feature enables customers to filter the metrics that Device Defender Detect evaluates by MQTT topic. Dimensions supports the following cloud-side metrics: number of messages received, message byte size, number of messages sent, source IP, and number of authorization failures. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	April 2, 2020

AWS IoT Device Defender ML Detect General Availability	The ML Detect feature of AWS IoT Device Defender automatically detects device-level operational and security anomalies across your fleet by learning from past data. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	March 24, 2020
AWS IoT Device Defender Adds Four New Checks to its Audit Capability	Use AWS IoT Device Defender Audit to check for devices in your fleet that have overly permissive permissions, have access to services that haven't been used in over 365 days, use OpenSSL versions on Debian-based operating systems that have been identified as having predictable cryptographic keys making them susceptible to brute force attacks, or use Infineon RSA library versions that have been identified to mishandle RSA key generatio n making them susceptible to hacking. For documenta tion, see Audit in the *AWS IoT Device Defender Developer Guide*.	November 25, 2019

AWS IoT Device Defender Supports Mitigation Actions for Audit Results	AWS IoT Device Defender supports the ability for customers to apply mitigation actions to audit findings. For documentation, see Audit in the *AWS IoT Device Defender Developer Guide*.	August 6, 2019
AWS IoT Device Defender supports monitoring behavior of unregistered devices	Identify unusual behavior for devices that are not registered with AWS IoT Core registry. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	May 15, 2019
AWS IoT Device Defender Now Provides Statistical Anomaly Detection and Data Visualization	Use statistical anomaly detection, and receive alerts when a device is not within the percentile-based threshold. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	February 19, 2019
AWS IoT Device Defender now supports monitoring of device disconnect durations	AWS IoT Device Defender now supports two additional Cloud-side metrics, number of connection attempts, and number of disconnects. For documentation, see Cloud-side metrics in the *AWS IoT Device Defender Developer Guide*.	December 19, 2018