Validez les webhooks Cisco Intersight : Guide Basé Sur La Logique

Table des matières

Introduction

Ce document décrit comment valider un webhook dans Intersight.

Conditions préalables

Lorsque Cisco Intersight envoie un webhook à votre application, il envoie essentiellement un événement (comme une alarme de serveur). Mais comment savez-vous que le message provient en fait de Cisco et n'a pas été envoyé par quelqu'un d'autre essayant de déclencher une action factice dans votre système ?

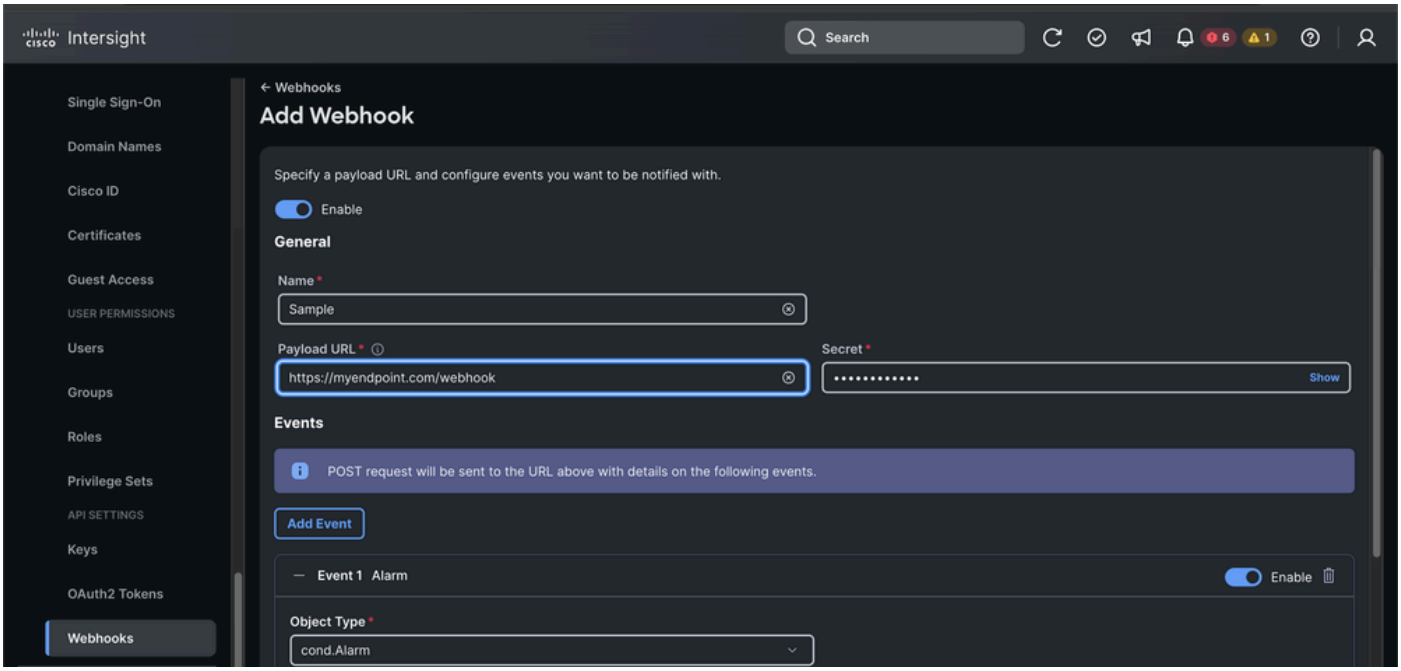
Pour résoudre ce problème, Intersight utilise un secret Webhook. Imaginez-le comme un sceau sur une enveloppe : si le sceau est brisé ou s'il semble différent de ce que vous attendiez, vous ne faites pas confiance à la lettre.

Définition du secret

La première étape consiste à configurer le Webhook dans Intersight et à établir un secret partagé.

1. Connectez-vous à Cisco Intersight.
2. Accédez à Paramètres > Webhooks.

3. Lorsque vous créez ou modifiez un Webhook, vous pouvez voir un champ intitulé Secret.
4. Définissez cette chaîne vous-même (par exemple, secret). Une fois enregistré, Intersight l'utilise pour signer tous les messages qu'il envoie.
5. Important : enregistrez ce secret de manière sécurisée et ne le partagez pas publiquement.



La logique de validation

Étape 1: Vérification du scellement (The Body Digest)

La première chose à vérifier est si le corps du message a été modifié pendant son voyage. Pour ce faire, nous utilisons un hachage (en particulier SHA-256).

Qu'est-ce qu'un hachage ?

C'est comme une empreinte. Même si vous modifiez une virgule dans un document de 10 pages, l'empreinte digitale change complètement.

La logique :

1. Prenez le corps de requête brut (le texte JSON exactement tel qu'il est arrivé).
2. Exécutez-le via une fonction de hachage SHA-256.
3. Convertissez cette empreinte binaire en une chaîne lisible à l'aide du codage Base64.
4. Comparez votre résultat avec le Digestheader envoyé par Intersight.
5. Cela doit être comme ceci : SHA-256=votre_chaîne_calculée.

Étape 2: Préparation de la cible de requête

L'aperçu inclut la destination du message dans sa signature pour empêcher les attaques de relecture (lorsqu'une personne vole un message valide et l'envoie à un autre terminal).

La logique : créez une chaîne qui combine la méthode HTTP et le chemin d'accès.

Format :(request-target) : post /your/endpoint/path

Étape 3: Créer la chaîne de signature

Doit être suivi dans un ordre strict.

C'est là que la plupart des développeurs rencontrent des problèmes. L'aperçu est extrêmement strict en ce qui concerne l'ordre, la sensibilité à la casse et le formatage des en-têtes utilisés pour la signature. Vous devez créer un bloc de texte unique où chaque ligne est nom-en-tête : valeur.

L'ordre exact requis :

1. (request-target) (À partir de l'étape 2)
2. animateur
3. sortir avec
4. digest (valeur complète de l'en-tête Digest de l'étape 1)
5. content-type (type de contenu)
6. content-length

```
(request-target): post /api/webhook
host: myapp.example.com
date: Mon, 09 Mar 2026 12:50:29 GMT
digest: SHA-256=L6Y...
content-type: application/json
content-length: 542
```

Étape 4: Générer la signature (HMAC)

Maintenant, vous utilisez la clé secrète (de l'interface d'Intersight) pour signer la chaîne que nous venons de construire. Nous utilisons une méthode appelée HMAC-SHA256.

Qu'est-ce que HMAC ?

C'est un moyen de signer un message à l'aide d'une clé secrète. Seule une personne possédant la même clé secrète peut produire la même signature.

La logique :

1. Entrée : la chaîne de signature de l'étape 3.
2. Clé : Votre Secret Webhook.
3. Processus : exécutez l’algorithme HMAC-SHA256.
4. Sortie : prenez les données binaires résultantes etBase64 Encodeit.

Étape 5: La comparaison finale

Intersight envoie un en-tête Authorization. Vous devez reconstruire ce que vous attendez de cet en-tête à l'aide de la signature que vous venez de générer.

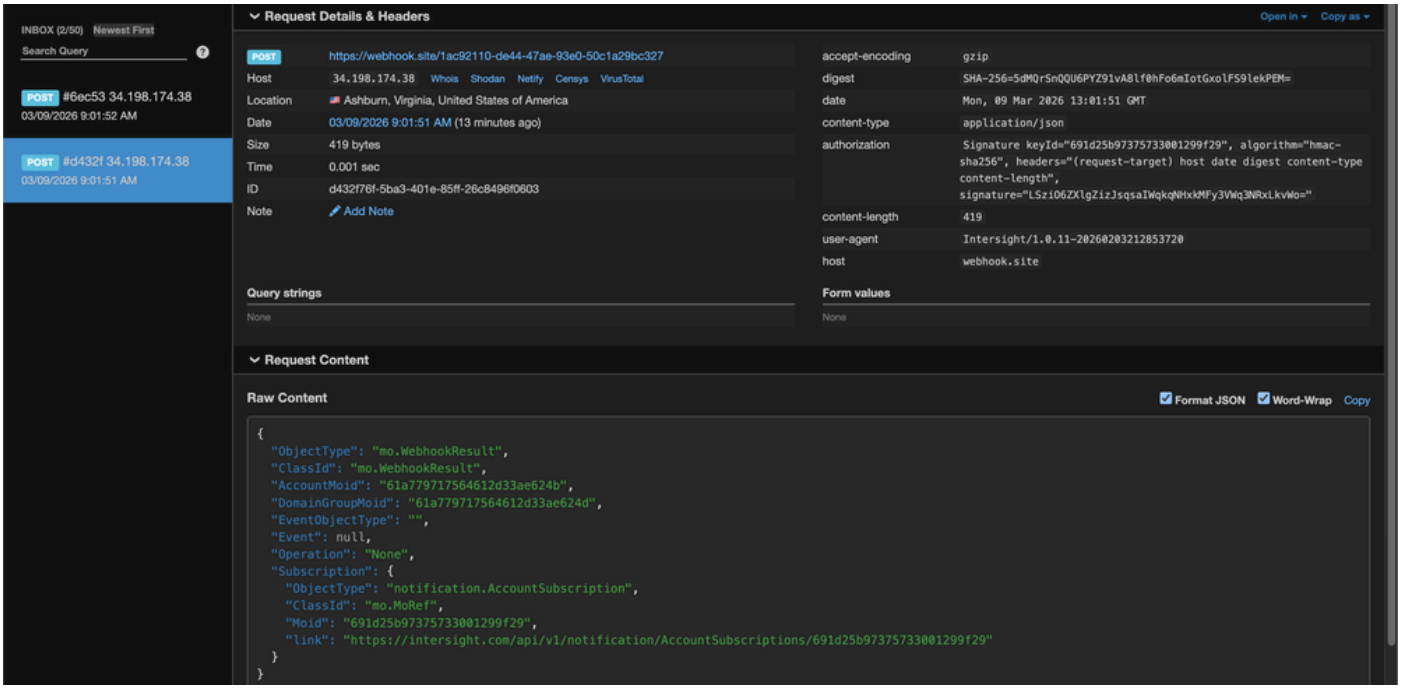
Si votre chaîne calculée correspond à l'en-tête Authorization fourni dans la demande, le message est authentique.

Considérations importantes

1. Inclinaison d'horloge : Vérifiez toujours l'en-tête de date. Si la requête a plus de 5 minutes par rapport à l'heure du serveur, rejetez-la pour empêcher les attaques de relecture.
2. Corps brut : N'analysez pas le fichier JSON, puis restreignez-le avant de valider le résumé. Différentes bibliothèques ajoutent un espacement différent, ce qui interrompt le hachage.
3. Ordre des en-têtes : Intersight valide la signature en fonction de l'ordre défini dans la section headers="..." de l'en-tête Authorization. Assurez-vous que la chaîne à signer correspond exactement à cet ordre.

Exemples vérifiables

Pour vous aider à tester votre code, voici un exemple basé sur une charge utile webhook réelle envoyée par Intersight.



Paramètres de test

<#root>

Secret

:secret

Host

:webhook.site

Path:

/1ac92110-de44-47ae-93e0-50c1a29bc327

Date

:Mon, 09 Mar 2026 13:01:51 GMT

Content-Length

:419

Payload

:{"ObjectType":"mo.WebhookResult","ClassId":"mo.WebhookResult","AccountMoid":"61a779717564612d33ae624b"

Expected Signature

:LSziO6ZXlgZizJsqsaIWqkqNHxkMFy3VWq3NRxLkvWo=

Vérification Bash et OpenSSL

```
#!/bin/bash

# 1. Setup the inputs
SECRET="secret"
EXPECTED_SIG="LSziO6ZXlgZizJsqsaIWqkqNHxkMFy3VWq3NRxLkvWo="
PAYLOAD='{"ObjectType":"mo.WebhookResult","ClassId":"mo.WebhookResult","AccountMoid":"61a779717564612d3

# 2. Calculate the Body Digest
# We use echo -n to ensure no trailing newline is added to the payload
DIGEST=$(echo -n "$PAYLOAD" | openssl dgst -sha256 -binary | openssl base64)
FULL_DIGEST="SHA-256=$DIGEST"

# 3. Build the Signing String (Strict Order!)
# Note: The format must be exactly: header: value\n
SIGNING_STR="(request-target): post /1ac92110-de44-47ae-93e0-50c1a29bc327
host: webhook.site
date: Mon, 09 Mar 2026 13:01:51 GMT
digest: $FULL_DIGEST
content-type: application/json
content-length: 419"

# 4. Generate the HMAC-SHA256 Signature
CALCULATED_SIG=$(echo -n "$SIGNING_STR" | openssl dgst -sha256 -hmac "$SECRET" -binary | openssl base64

# 5. Output the results for comparison
echo "--- Verification Results ---"
echo "Expected Signature:   $EXPECTED_SIG"
echo "Calculated Signature: $CALCULATED_SIG"

if [ "$EXPECTED_SIG" == "$CALCULATED_SIG" ]; then
    echo "SUCCESS: The signatures match!"
else
    echo "FAILURE: The signatures do not match."
```

```
fi
```

Vérification PowerShell

```
# 1. Setup the inputs
$Secret = "secret"
$ExpectedDigest = "5dMQrSnQQU6PYZ91vA8lf0hFo6mIotGxolFS9lekPEM="
$ExpectedSig    = "LSziO6ZXlgZizJsqsaIWqkqNHxkMFy3VWq3NRxLkvWo="

$Payload = '{"ObjectType":"mo.WebhookResult","ClassId":"mo.WebhookResult","AccountMoid":"61a77971756461

# 2. Calculate the Body Digest
$Sha256 = [System.Security.Cryptography.SHA256]::Create()
$PayloadBytes = [System.Text.Encoding]::UTF8.GetBytes($Payload)
$HashBytes = $Sha256.ComputeHash($PayloadBytes)
$CalculatedDigest = [Convert]::ToBase64String($HashBytes)

# 3. Build the Signing String (Strict Order!)
# Note: `n is the PowerShell newline character.
# The string must match the order in the Authorization header exactly.
$SigningStr = "(request-target): post /1ac92110-de44-47ae-93e0-50c1a29bc327`n" +
              "host: webhook.site`n" +
              "date: Mon, 09 Mar 2026 13:01:51 GMT`n" +
              "digest: SHA-256=$CalculatedDigest`n" +
              "content-type: application/json`n" +
              "content-length: 419"

# 4. Generate the HMAC-SHA256 Signature
$Hmac = New-Object System.Security.Cryptography.HMACSHA256
$Hmac.Key = [System.Text.Encoding]::UTF8.GetBytes($Secret)
$SigBytes = $Hmac.ComputeHash([System.Text.Encoding]::UTF8.GetBytes($SigningStr))
$CalculatedSig = [Convert]::ToBase64String($SigBytes)

# 5. Output the results for comparison
Write-Host "--- Verification Results ---" -ForegroundColor Cyan

Write-Host "Digest Match: " -NoNewline
if ($CalculatedDigest -eq $ExpectedDigest) {
    Write-Host "SUCCESS" -ForegroundColor Green
} else {
    Write-Host "FAILED" -ForegroundColor Red
}

Write-Host "Expected Signature:   $ExpectedSig"
Write-Host "Calculated Signature: $CalculatedSig"

if ($CalculatedSig -eq $ExpectedSig) {
    Write-Host "SUCCESS: The signatures match!" -ForegroundColor Green
} else {
    Write-Host "FAILURE: The signatures do not match." -ForegroundColor Red
}
```

Informations connexes

- Appeler la demande d'API Web
- Configuration du webhook Cisco Intersight
- Webhook/Terminaux

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.