Validieren Sie die Cisco Intersight-Webhooks: Logischer Leitfaden

Inhalt

Einleitung

In diesem Dokument wird beschrieben, wie Sie einen Webhook in Intersight validieren können.

Voraussetzungen

Wenn Cisco Intersight einen Webhook an Ihre Anwendung sendet, wird im Grunde ein Ereignis (wie ein Serveralarm) gesendet. Aber woher wissen Sie, dass die Nachricht tatsächlich von Cisco stammt und nicht von jemand anderem gesendet wurde, der versucht hat, eine gefälschte Aktion in Ihrem System auszulösen?

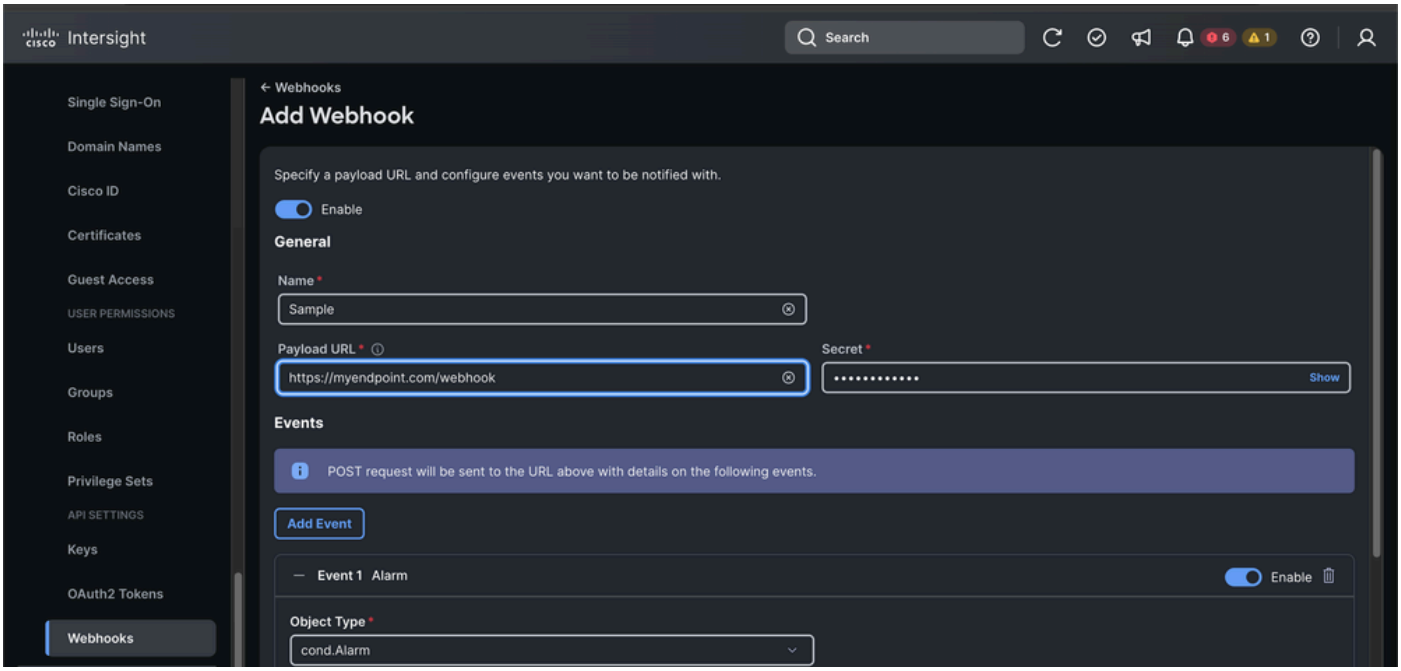
Um dies zu lösen, verwendet Intersight einen Webhook Secret. Stellen Sie es sich wie ein Siegel auf einem Umschlag vor: Wenn das Siegel gebrochen ist oder anders aussieht als erwartet, vertrauen Sie dem Brief nicht.

Das Geheimnis

Der erste Schritt besteht darin, den Webhook in Intersight zu konfigurieren und einen Shared Secret einzurichten.

1. Melden Sie sich bei Cisco Intersight an.

2. Navigieren Sie zu Einstellungen > Webhooks.
3. Wenn Sie einen Webhook erstellen oder bearbeiten, sehen Sie ein Feld mit der BezeichnungSecret.
4. Definieren Sie diesen String selbst (z.B. ecret). Nach dem Speichern verwendet Intersight diese zum Signieren jeder gesendeten Nachricht.
5. Wichtig: Speichern Sie dieses Geheimnis auf sichere Weise, und geben Sie es nicht öffentlich weiter.



Die Validierungslogik

Schritt 1: Überprüfen Sie das Siegel (die Body Digest).

Als Erstes muss überprüft werden, ob der Nachrichtentext während der Übertragung geändert wurde. Dies geschieht mit einem Hash (speziell SHA-256).

Was ist ein Hash?

Es ist wie ein Fingerabdruck. Selbst wenn Sie in einem zehnseitigen Dokument ein Komma ändern, ändert sich der Fingerabdruck vollständig.

Die Logik:

1. Nehmen Sie die Raw Request Body (den JSON-Text genau so, wie er angekommen ist).
2. Führen Sie es durch eine SHA-256-Hashing-Funktion.
3. Konvertieren Sie diesen binären Fingerabdruck mithilfe von Base64 Encoding in eine lesbare Zeichenfolge.
4. Vergleichen Sie Ihr Ergebnis mit dem Digestheader von Intersight.
5. Es muss wie folgt aussehen:SHA-256=your_calculation_string.

Phase 2: Ziel für Anforderung vorbereiten

Intersight enthält das Ziel der Nachricht in seiner Signatur, um Replay-Angriffe zu verhindern (bei denen jemand eine gültige Nachricht stiehlt und an einen anderen Endpunkt sendet).

Die Logik:Erstellen Sie eine Zeichenfolge, die die HTTP-Methode und den Pfad kombiniert.

Format: (Anforderungsziel): /Ihr/Endpunkt/Pfad bereitstellen

Schritt 3: Erstellen der Signaturzeichenfolge

Muss in strikter Reihenfolge befolgt werden.

Hier geraten die meisten Entwickler in Schwierigkeiten. Intersight ist hinsichtlich der Reihenfolge, der Groß-/Kleinschreibung und der Formatierung der für die Signatur verwendeten Header äußerst streng. Sie müssen einen einzelnen Textblock erstellen, in dem jede Zeile den Header-Namen hat: wert.

Die genaue Bestellung ist erforderlich:

1. (Anforderungsziel) (Von Schritt 2)
2. Gastgeber
3. Datum
4. Digest (Der vollständige Wert des Digest-Headers aus Schritt 1)
5. Inhaltstyp
6. Inhaltslänge

```
(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
```

Schritt 4: Signatur erstellen (HMAC)

Jetzt verwenden Sie den Geheimschlüssel (aus der Intersight-Benutzeroberfläche), um die gerade erstellte Zeichenfolge zu signieren. Wir verwenden eine Methode namens HMAC-SHA256.

Was ist HMAC?

Es ist eine Möglichkeit, eine Nachricht mit einem geheimen Schlüssel zu signieren. Nur jemand mit dem gleichen geheimen Schlüssel kann die gleiche Signatur erzeugen.

Die Logik:

1. Eingabe: Die Signaturzeichenfolge aus Schritt 3.
2. Schlüssel: Ihr Webhook Secret.
3. Prozess:Führen Sie den HMAC-SHA256-Algorithmus aus.

4. Ausgabe:Nehmen Sie die resultierenden Binärdaten undBase64 Encoding.

Schritt 5: Der abschließende Vergleich

Intersight sendet einen Autorisierungs-Header. Sie müssen anhand der gerade generierten Signatur rekonstruieren, wie der Header aussehen soll.

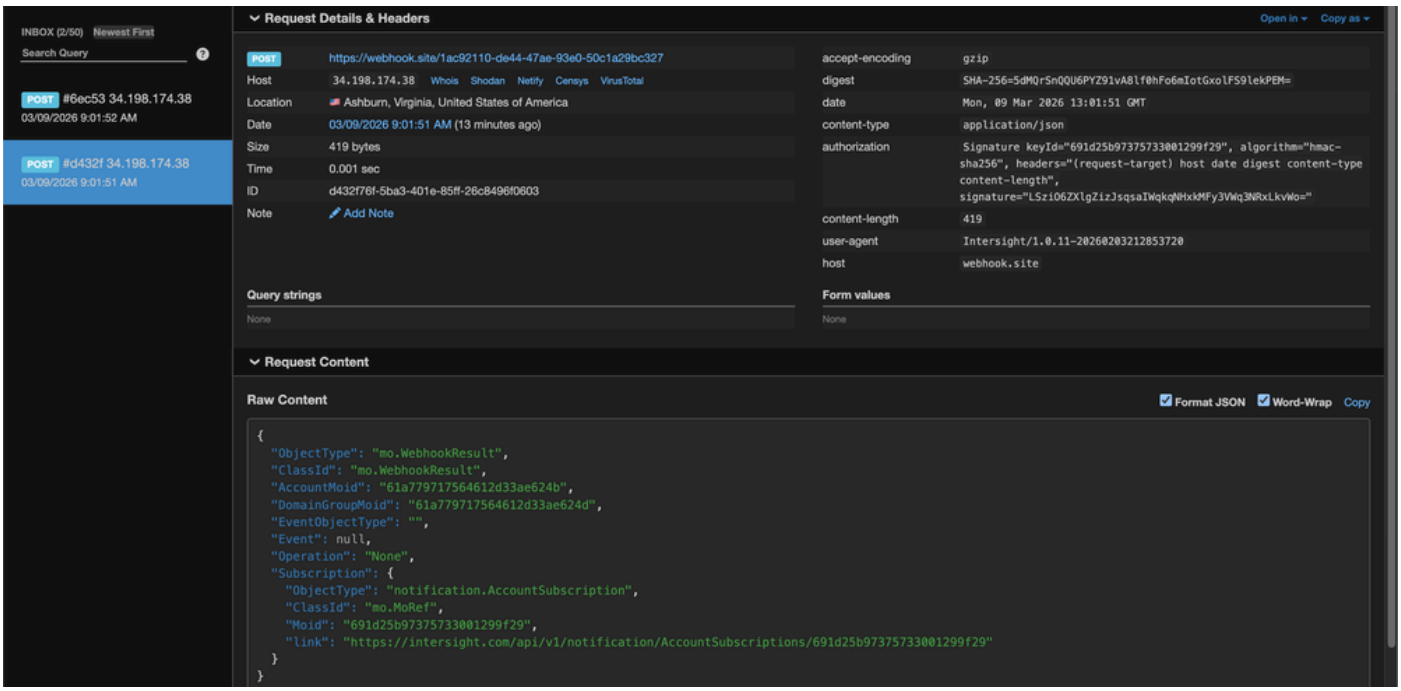
Wenn die berechnete Zeichenfolge mit dem in der Anforderung angegebenen Autorisierungs-Header übereinstimmt, ist die Nachricht authentisch.

Wichtige Überlegungen

1. Uhr schief: Überprüfen Sie immer den Datumskopf. Wenn die Anfrage im Vergleich zur Serverzeit mehr als 5 Minuten alt ist, weisen Sie sie zurück, um Replay-Angriffe zu verhindern.
2. Rohtext: Analysieren Sie die JSON nicht, und stringieren Sie sie anschließend erneut, bevor Sie den Digest validieren. Verschiedene Bibliotheken fügen unterschiedliche Abstände hinzu, wodurch der Hash unterbrochen wird.
3. Kopfzeilenreihenfolge: Intersight validiert die Signatur anhand der Reihenfolge, die im Abschnitt header="..." des Headers Authorization definiert ist. Stellen Sie sicher, dass Ihre zu signierende Zeichenfolge genau dieser Reihenfolge entspricht.

Verifizierbare Beispiele

Um Ihnen beim Testen des Codes zu helfen, finden Sie hier ein Beispiel, das auf einer echten Webhook-Payload basiert, die von Intersight gesendet wurde.



Testparameter

<#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=

Bash und OpenSSL-Verifizierung

```
#!/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
```

PowerShell-Überprüfung

```
# 1. Setup the inputs
$Secret = "secret"
$ExpectedDigest = "5dMQrSnQQU6PYZ91vA8lfOhFo6mIotGxolFS9lekPEM="
$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
}
```

Zugehörige Informationen

- Web-API-Anforderung aufrufen
- Konfiguration des Cisco Interview-Webhooks
- Webhook/Endgeräte

Informationen zu dieser Übersetzung

Cisco hat dieses Dokument maschinell übersetzen und von einem menschlichen Übersetzer editieren und korrigieren lassen, um unseren Benutzern auf der ganzen Welt Support-Inhalte in ihrer eigenen Sprache zu bieten. Bitte beachten Sie, dass selbst die beste maschinelle Übersetzung nicht so genau ist wie eine von einem professionellen Übersetzer angefertigte. Cisco Systems, Inc. übernimmt keine Haftung für die Richtigkeit dieser Übersetzungen und empfiehlt, immer das englische Originaldokument (siehe bereitgestellter Link) heranzuziehen.