Cisco Intersight Webhook：ロジックベースガイドの検証

内容

はじめに

このドキュメントでは、intersightでwebhookを検証する方法について説明します。

前提条件

Cisco IntersightがWebhookをアプリケーションに送信すると、基本的にイベント（サーバアラームなど）が送信されます。 しかし、メッセージが実際にシスコから送信されたものであり、システム内で偽のアクションをトリガーしようとしている他のユーザによって送信されたものではないことを知るにはどうすればよいでしょうか。

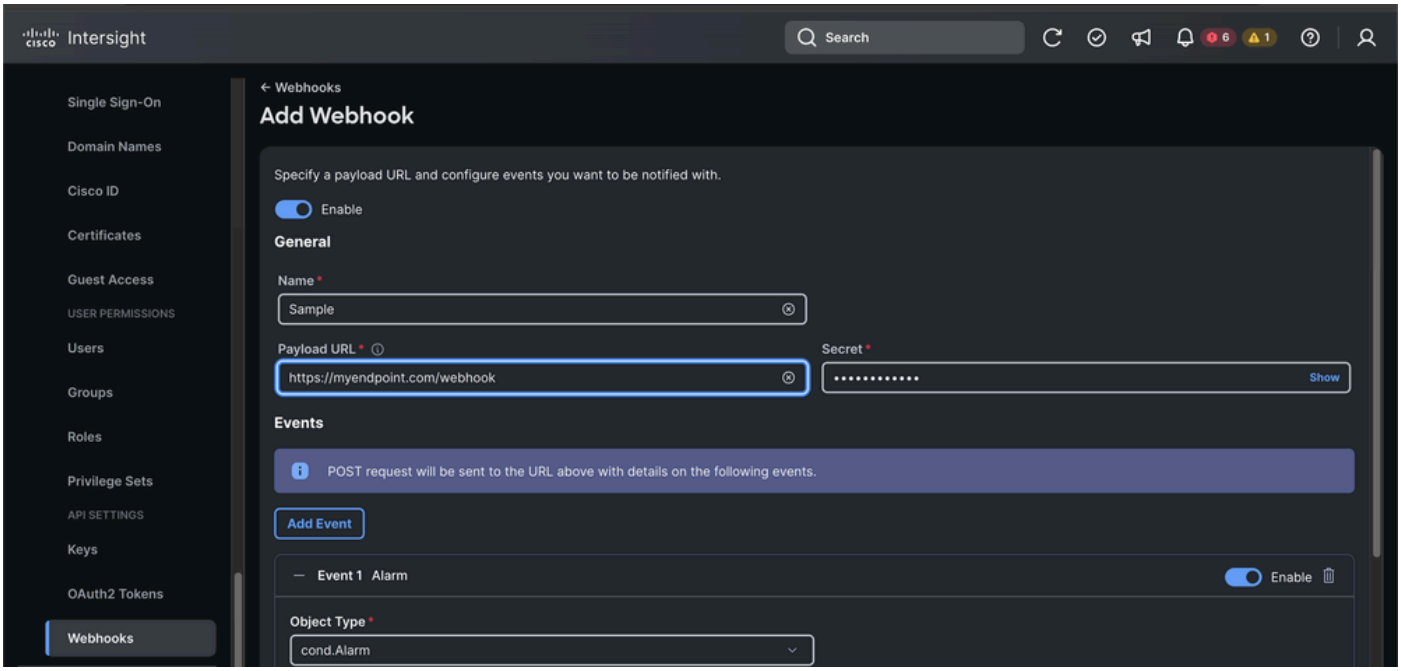
これを解決するために、IntersightはWebhookシークレットを使用します。封筒のシールのように考えてください。シールが壊れているか、または予想と異なって見える場合は、手紙を信頼しません。

シークレットの設定

最初のステップは、IntersightでWebhookを設定し、共有秘密を確立することです。

1. Cisco Intersightにログインします。
2. Settings > Webhookの順に移動します。

3. Webhookを作成または編集すると、Secretというラベルが付いたフィールドが表示されます。
4. この文字列を自分で定義します（たとえば、ecret）。 保存されると、Intersightはこれを使用して、送信するすべてのメッセージに署名します。
5. 重要：この秘密を安全な方法で保存し、公開しないでください。



検証ロジック

ステップ1：シール（ボディダイジェスト）を確認します。

最初に確認するのは、メッセージ本文が移動中に変更されたかどうかです。これを行うには、ハッシュ(具体的にはSHA-256)を使用します。

ハッシュとは

指紋のようなものだ。10ページのドキュメントで1つのカンマを変更しても、フィンガープリントは完全に変更されます。

ロジック：

1. Raw Request Body（JSONテキストが到達したとおりに）を取得します。
2. aSHA-256ハッシュ関数を使用します。
3. そのバイナリフィンガープリントを、Base64エンコーディングを使用して読み取り可能な文字列に変換します。
4. 結果をインターサイトから送信されたDigestheaderと比較します。
5. SHA-256=your_calculated_stringのような形式である必要があります。

ステップ2：要求ターゲットの準備

Intersightでは、リプレイアタック（有効なメッセージが盗まれ、別のエンドポイントに送信され

る攻撃）を防止するために、シグニチャにメッセージの宛先が含まれています。

Logic:HTTPメソッドとパスを組み合わせた文字列を作成します。

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

ステップ3：署名文字列の作成

厳密な順序で従う必要があります。

この場合、ほとんどの開発者が問題に直面します。Intersightは、シグニチャに使用されるヘッダーの順序、大文字と小文字の区別、および形式に関して非常に厳密です。各行がheader-name: valueである単一テキストブロックを作成する必要があります。

正確な順序は次のとおりです。

1. (request-target)（ステップ2から）
2. host
3. 日付
4. digest（ステップ1のDigestヘッダーの完全な値）
5. content-type
6. コンテンツ長

```
(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
```

ステップ4：シグニチャを生成する(HMAC)

次に、（Intersight UIの）秘密キーを使用して、作成したばかりの文字列に署名します。ここでは、HMAC-SHA256という方法を使用します。

HMACとは

これは、秘密キーを使用してメッセージに署名する方法です。同じ秘密鍵を持つユーザだけが、同じシグニチャを生成できます。

ロジック：

1. Input:ステップ3で取得した署名文字列。
2. キー：Webhookの秘密。
3. プロセス：HMAC-SHA256アルゴリズムを実行します。
4. 出力：生成されたバイナリデータとBase64 Encodeitを取得します。

ステップ5：最終比較

IntersightはAuthorizationヘッダーを送信します。生成したシグニチャを使用して、そのヘッダーがどのように見えるかを再構築する必要があります。

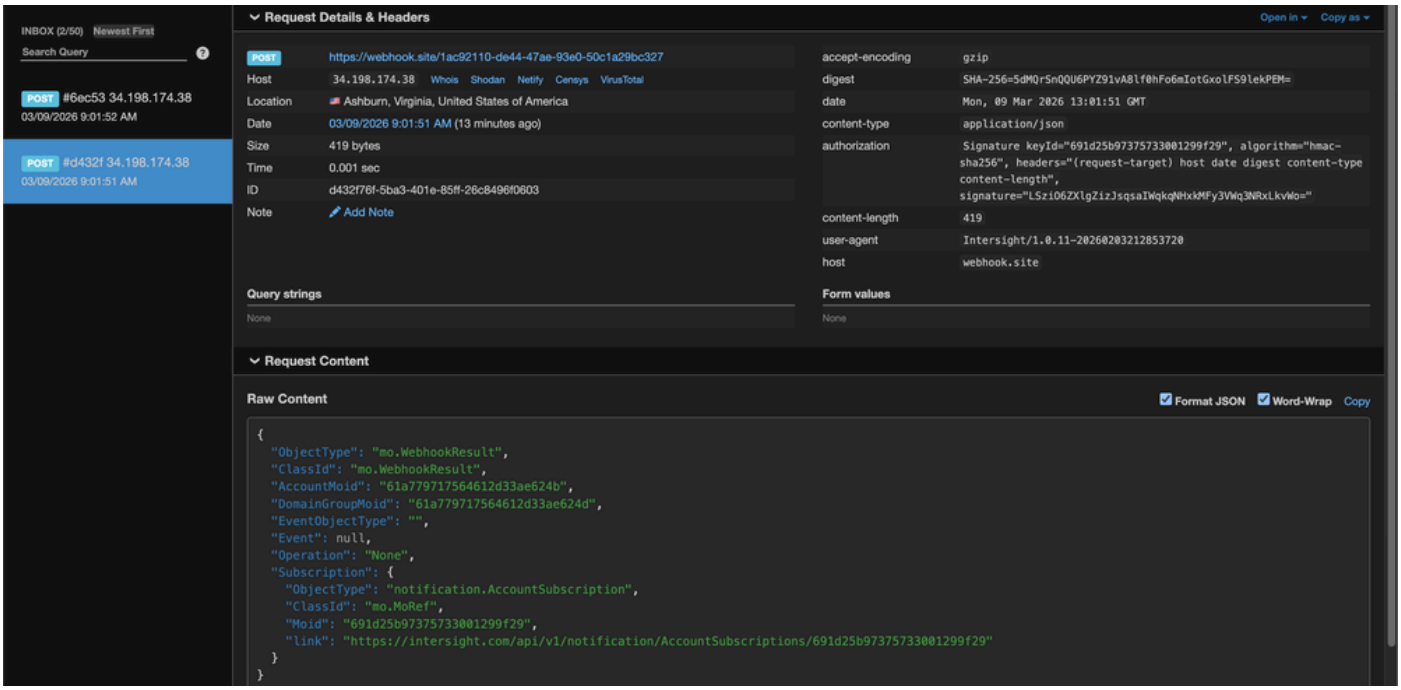
計算された文字列が、要求で提供されるAuthorizationヘッダーと一致する場合、メッセージは本物です。

重要な考慮事項

1. クロックスキュー:データヘッダーを常に確認します。要求がサーバ時間に比べて5分以上前である場合は、リプレイアタックを防ぐためにそれを拒否します。
2. Raw Body：ダイジェストを検証する前に、JSONを解析してから再度文字列を生成しないでください。ライブラリが異なると追加されるスペースも異なるため、ハッシュが分割されます。
3. ヘッダーの順序:Intersightは、許可ヘッダーのheaders="..."セクションで定義されている順序に基づいて署名を検証します。String to Signがその順序に正確に一致していることを確認します。

検証可能な例

コードをテストするために、Intersightから送信された実際のWebHookペイロードに基づく例を次に示します。



テストパラメータ

<#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およびOpenSSLの検証

```
#!/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の検証

```
# 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
}
```

関連情報

- Web API要求の呼び出し
- Cisco Intersight Webhookの設定
- webhook/エンドポイント

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。