验证Cisco Intersight Webhook:基于逻辑指南

目录

简介

本文档介绍如何在intersight中验证webhook。

先决条件

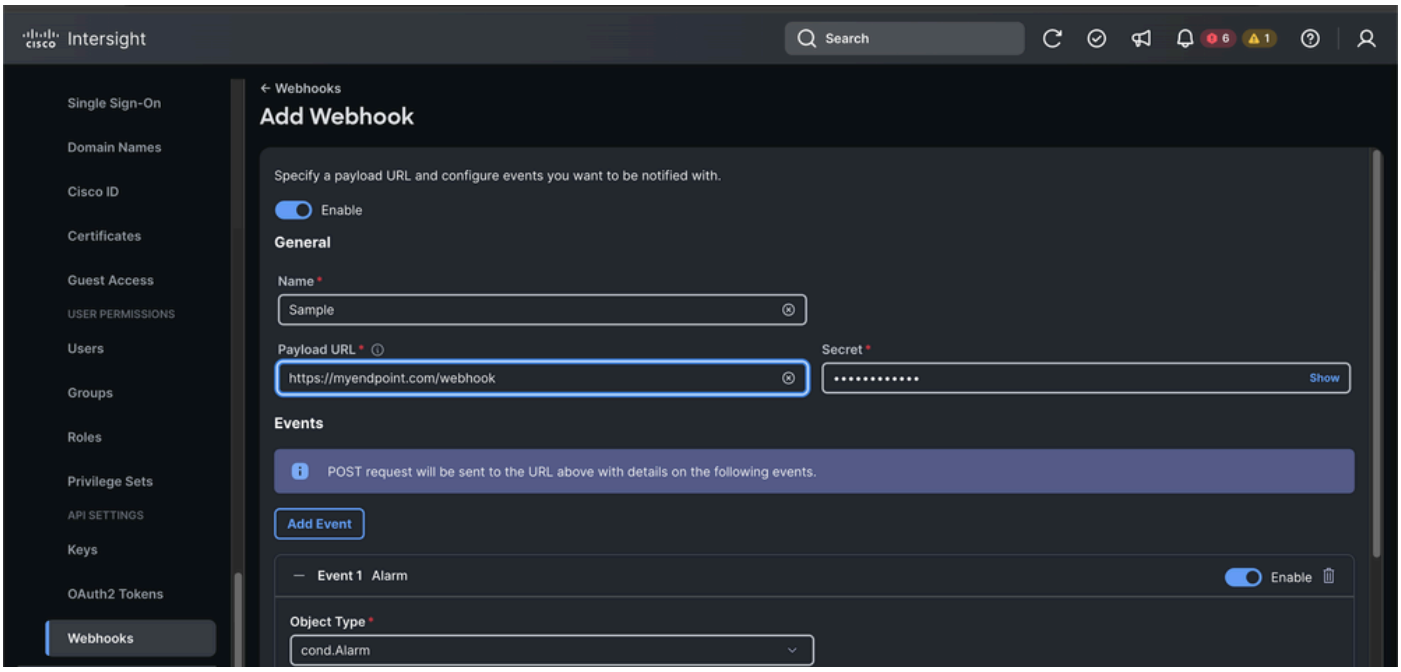
当Cisco Intersight向您的应用发送Webhook时，它实际上是在发送事件（如服务器警报）。 但是，您如何知道该消息实际上来自思科，而不是由试图在您的系统中触发虚假操作的其他人发送的？

为了解决这个问题，Intersight使用Webhook密钥。把它想象成信封上的封条：如果封条破损或看上去与预期不同，您不信任此信件。

设置密钥

第一步是在Intersight中配置Webhook并建立共享密钥。

1. 登录Cisco Intersight。
2. 导航到设置> Webhooks。
3. 创建或编辑Webhook时，可以看到标记为Secret的字段。
4. 自行定义此字符串（例如ecret）。 保存后，Intersight会使用此信息对发送的每条消息进行签名。
5. 重要信息：以安全方式保存此密钥，不要公开共享。



验证逻辑

步骤 1：检验密封（正文摘要）

首先要检查的是邮件正文是否在其传输过程中发生了更改。我们使用Hash(特别是SHA-256)执行此操作。

什么是Hash?

就像指纹。即使您在10页的文档中更改一个逗号，指纹也会完全更改。

逻辑：

1. 采用Raw Request Body（JSON文本与它到达的完全相同）。
2. 通过aSHA-256散列函数运行它。
3. 使用Base64编码将该二进制指纹转换为可读字符串。
4. 将您的结果与Intersight发送的Digestheader进行比较。
5. 它必须如下所示:SHA-256=your_calculated_string。

步骤 2：准备请求目标

Intersight在其签名中包含消息的目标以防止重播攻击（在此处，有人窃取有效消息并将其发送到其他终端）。

逻辑：创建一个将HTTP方法和路径结合在一起的字符串。

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

步骤 3：创建签名字符串

必须严格遵照执行。

这是大多数开发者遇到麻烦的地方。Intersight对用于签名的信头的顺序、区分大小写和格式设置极其严格。您必须构建一个文本块，其中每行均为header-name:价值.

所需的确切顺序：

1. (request-target)（来自第2步）
2. 主机
3. 日期
4. digest（步骤1中的摘要报头的完整值）
5. 内容类型
6. 内容长度

```
(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
```

步骤 4：生成签名(HMAC)

现在使用Secret Key（从Intersight UI）对我们刚才构建的字符串进行签名。我们使用称为HMAC-SHA256的方法。

什么是HMAC?

这是一种使用密钥对消息进行签名的方式。只有拥有相同密钥的人才能生成相同的签名。

逻辑：

1. 输入：步骤3中的签名字符串。
2. 密钥：您的Webhook密钥。
3. 流程：运行HMAC-SHA256算法。
4. 输出：取得生成的二进制数据和Base64编码。

步骤 5：最终比较

Intersight发送Authorization报头。您需要使用您刚才生成的签名重建期望信头的外观。

如果计算出的字符串与请求中提供的Authorization报头匹配，则消息是真实的。

重要注意事项

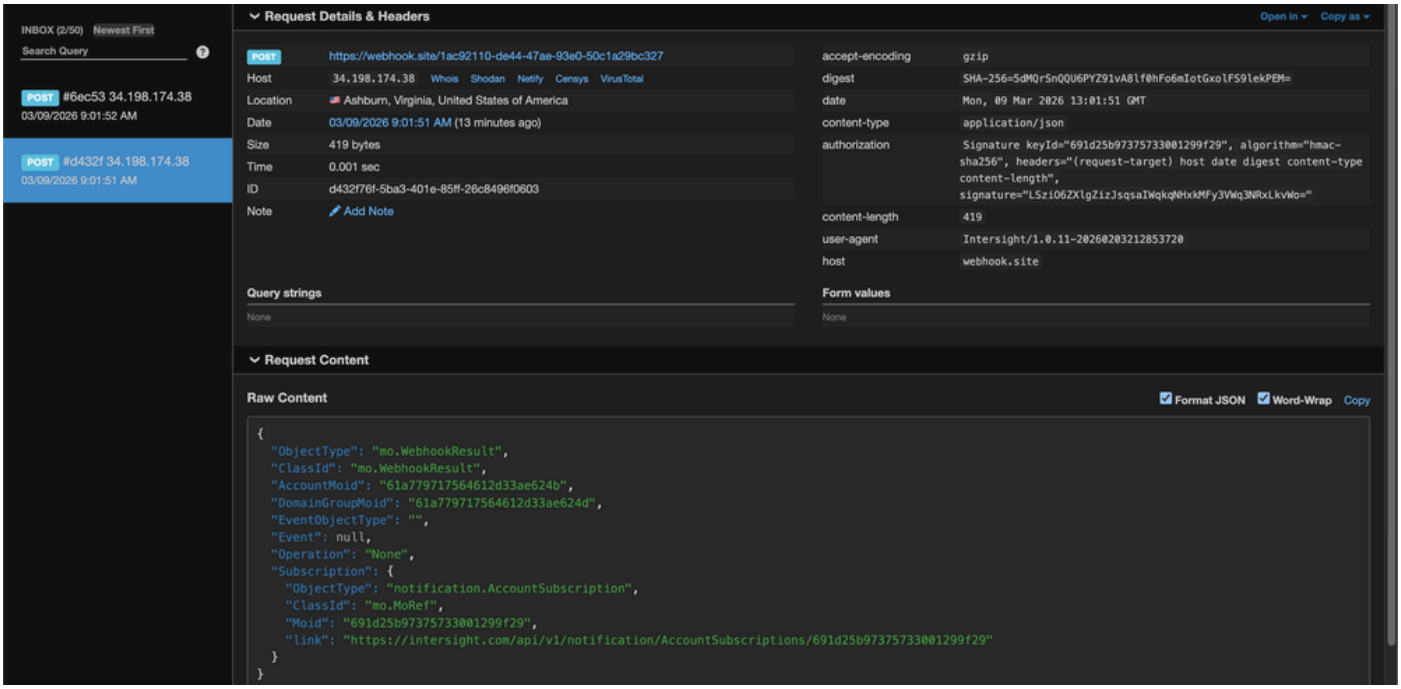
1. 时滞:请始终检查日期标题。如果与服务器时间相比，请求超过5分钟，则拒绝该请求以防止重放攻击。
2. 原始正文:在验证摘要之前，请勿分析JSON并重新对其进行结构化。不同的库会添加不同的间

距，从而打破哈希值。

3. 标题顺序:Intersight根据授权报头的headers="。.."部分中定义的顺序验证签名。确保String to Sign与该顺序完全匹配。

可验证的示例

为了帮助您测试代码，下面是一个基于Intersight发送的实际Webhook负载的示例。



测试参数

<#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和OpenSSL验证

```
#!/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验证

```
# 1. Setup the inputs
$Secret = "secret"
$ExpectedDigest = "5dMQrSnQQU6PYZ91vA8lfOhFo6mIotGxolFS9lekPEM="
$ExpectedSig    = "LSziO6ZXlgZizJsqsaIWqkqNHxkMFy3VWq3NRxLkvWo="

$Payload = '{"ObjectType":"mo.WebhookResult","ClassId":"mo.WebhookResult","AccountMoid":"61a7797175646

# 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
}
```

相关信息

- 调用Web API请求
- Cisco Intersight Webhook配置
- Webhook/终端

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。