

RADIUS 無効な認証者およびメッセージ認証者のトラブルシューティング ガイド

内容

[概要](#)

[オーセンティケーターヘッダー](#)

[応答の認証](#)

[検証の失敗はいつ予想されますか。](#)

[パスワード非表示](#)

[再送信](#)

[アカウントティング](#)

[Message-Authenticator属性](#)

[Message-Authenticatorはいつ使用すべきですか。](#)

[検証の失敗はいつ予想されますか。](#)

[Message-Authenticator属性の検証](#)

[関連情報](#)

概要

このドキュメントでは、次に示す 2 つの RADIUS セキュリティ メカニズムについて説明します。

- オーセンティケーターヘッダー
- Message-Authenticator属性

このドキュメントでは、これらのセキュリティメカニズムの概要、使用方法、および検証の失敗が予想されるタイミングについて説明します。

オーセンティケーターヘッダー

RFC 2865によると、オーセンティケーターヘッダーの長さは16バイトです。Access-Requestで使用される場合、Request Authenticatorと呼ばれます。どのような応答でも使用される場合は、応答オーセンティケーターと呼ばれます。次の用途に使用されます。

- 応答の認証
- パスワード隠蔽

応答の認証

サーバが正しい応答オーセンティケーターで応答すると、クライアントはその応答が有効な要求に

関連しているかどうかを計算できます。

クライアントは、ランダムなオーセンティケータヘッダーを含む要求を送信します。次に、応答を送信するサーバは、共有秘密とともに要求パケットを使用して応答オーセンティケータを計算します。

```
ResponseAuth = MD5(Code + ID + Length + RequestAuth + Attributes + Secret)
```

応答を受信したクライアントは、同じ操作を実行します。結果が同じ場合、パケットは正しいです。

注：秘密の値を知っている攻撃者は、要求をスニフリングできない限り、応答をスプーフィングできません。

検証の失敗はいつ予想されますか。

スイッチが要求をキャッシュしなくなると、検証エラーが発生します(タイムアウトなど)。共有秘密が無効な場合にも発生する可能性があります(yes - Access-Rejectにこのヘッダーも含まれます)。これにより、ネットワークアクセスデバイス(NAD)は共有秘密の不一致を検出できます。通常は、認証、認可、アカウントिंग(AAA)クライアント/サーバによって共有キーの不一致として報告されますが、詳細は明らかになりません。

パスワード非表示

オーセンティケータヘッダーは、プレーンテキストでUser-Password属性を送信しないようにするために使用されます。まず、Message Digest 5 (MD5 - シークレット、オーセンティケータ)が計算されます。その後、パスワードのチャンクを持つ複数のXOR演算が実行される。この方法は、MD5が強力な一方向アルゴリズムとして認識されなくなるため、オフライン攻撃(レインポテーブル)の影響を受けやすくなっています。

User-Passwordを計算するPythonスクリプトを次に示します。

```
def Encrypt_Pass(password, authenticator, secret):
    m = md5()
    m.update(secret+authenticator)
    return "".join(chr(ord(x) ^ ord(y)) for x, y in zip(password.ljust(16, '\0')[:16], m.digest()[:16]))
```

再送信

RADIUS Access-Requestの属性 (RADIUS ID、ユーザ名など) が変更された場合、新しいAuthenticatorフィールドが生成され、それに依存する他のすべてのフィールドが再計算されます。再送信の場合は、何も変更しないでください。

アカウントिंग

オーセンティケータヘッダーの意味は、アクセス要求とアカウントिंग要求では異なります。

Access-Requestでは、オーセンティケータはランダムに生成され、ResponseAuthenticatorが正しく計算された応答を受信することが予想されます。これは、応答がその特定の要求に関連していたことを証明します。

アカウント要求の場合、オーセンティケータはランダムではありませんが、計算されます (RFC 2866に従います)。

```
RequestAuth = MD5(Code + ID + Length + 16 zero octets + Attributes + Secret)
```

このようにして、サーバはアカウントメッセージを即座にチェックし、再計算された値がオーセンティケータの値と一致しない場合はパケットをドロップできます。Identity Services Engine(ISE)は次を返します。

```
11038 RADIUS Accounting-Request header contains invalid Authenticator field
```

この一般的な理由は、誤った共有秘密キーです。

Message-Authenticator属性

Message-Authenticator属性は、RFC 3579で定義されているRADIUS属性です。同様の目的で使用されます。署名して検証します。ただし、今回は応答の検証ではなく要求の検証には使用されません。

Access-Requestを送信するクライアント (Access-Challengeで応答するサーバでもある) は、自身のパケットからHash-Based Message Authentication Code(HMAC)-MD5を計算し、Message-Authenticator属性をシグニチャとして追加します。その後、サーバは同じ動作を実行していることを確認できます。

式は、オーセンティケータヘッダーに似ています。

```
Message-Authenticator = HMAC-MD5 (Type, Identifier, Length, Request Authenticator, Attributes)
```

HMAC-MD5関数には、次の2つの引数があります。

- パケットのペイロード。16バイトのMessage-Authenticatorフィールドにゼロが埋め込まれている
- 共有秘密鍵

Message-Authenticatorはいつ使用すべきですか。

Message-Authenticator(MUST)は、Extensible Authentication Protocol(EAP)メッセージ(RFC 3579)を含むすべてのパケットに使用する必要があります。これには、Access-Requestを送信するクライアントと、Access-Challengeで応答するサーバの両方が含まれます。検証が失敗した場合は、相手側が何も通知せずにパケットをドロップします。

検証の失敗はいつ予想されますか。

共有秘密が無効な場合、検証エラーが発生します。その後、AAAサーバは要求を検証できません

。

ISEは次のように報告します。

```
11036 The Message-Authenticator Radius Attribute is invalid.
```

これは通常、EAPメッセージが添付された後の段階で発生します。802.1xセッションの最初のRADIUSパケットには、EAPメッセージは含まれません。Message-Authenticatorフィールドはなく、要求を確認することはできませんが、その段階では、クライアントはAuthenticatorフィールドを使用して応答を検証できます。

Message-Authenticator属性の検証

次に、値が正しく計算されていることを確認するために、手動で値をカウントする方法を示す例を示します。

パケット番号30(Access-Request)が選択されました。EAPセッションの途中にあり、パケットにはMessage-Authenticatorフィールドが含まれています。目的は、Message-Authenticatorが正しいことを確認することです。

```
30 2012-12-20 07:34:19.221908 192.168.10.10 192.168.10.150 RADIUS 401 Access-Request(1)
|
+-----+
| Radius Protocol |
+-----+
Code: Access-Request (1)
Packet identifier: 0x16 (22)
Length: 359
Authenticator: bed95259578302c0f9184df62b859d6b
[The response to this request is in frame 31]
+ Attribute Value Pairs
+ AVP: l=7 t=User-Name(1): cisco
+ AVP: l=6 t=Service-Type(6): Framed(2)
+ AVP: l=6 t=Framed-MTU(12): 1500
+ AVP: l=19 t=Called-Station-Id(30): AA-BB-CC-00-64-00
+ AVP: l=19 t=Calling-Station-Id(31): 08-00-27-6E-C5-50
+ AVP: l=202 t=EAP-Message(79) Last Segment[1]
+ AVP: l=18 t=Message-Authenticator(80): 01418d3b1865556918269d3c f73608b0
```

1. [Radius Protocol]を右クリックし、[Export selected packet bytes]を選択します。
2. そのRADIUSペイロードをファイル (バイナリデータ) に書き込みます。
3. Message-Authenticatorフィールドを計算するには、そこにゼロを入れてHMAC-MD5を計算する必要があります。

たとえば、vimなどの16進数/バイナリエディタを使用する場合、「:%!xxd」と入力すると、16バイトが「5012」の後に切り替わり、16バイトが16バイトになります(50hexは80、12はAttribute Value Pairs (AVP)ヘッダー):

```
0000000: 0116 0167 bed9 5259 5783 02c0 f918 4df6 ...g..RYW.....M.
0000010: 2b85 9d6b 0107 6369 7363 6f06 0600 0000 +..k..cisco.....
0000020: 020c 0600 0005 dc1e 1341 412d 4242 2d43 .....AA-BB-C
0000030: 432d 3030 2d36 342d 3030 1f13 3038 2d30 C-00-64-00..08-0
0000040: 302d 3237 2d36 452d 4335 2d35 304f ca02 0-27-6E-C5-500..
0000050: 4100 c819 8000 0000 be16 0301 0086 1000 A.....
0000060: 0082 0080 880d 0fe6 8421 562e bcf3 75a7 .....!V...u.
0000070: fbf4 9c20 e114 a19d 1282 96d7 45b8 9c26 ... ..E..&
0000080: 86c5 9935 1b2c ca98 1b60 5e91 1c63 d123 ...5.,...^..c.#
0000090: f019 1ab6 7e2d 0497 1e02 0768 0ac3 aa84 ....~.....h...
00000a0: 80d5 cd14 92a9 ae31 e9e2 121e 28e8 5f21 .....1....(._!
00000b0: 5c1a 4e20 013f a55b 7b1d 0eb7 1d17 a565 \.N .?.[ {...e
00000c0: 626b 2bb4 f756 da05 b51b 043b 346a c51f bk+..V.....;4j..
00000d0: 98a7 007e ed55 e24b 1cab ec06 799b aed5 ...~.U.K....y...
00000e0: 72c5 451b 1403 0100 0101 1603 0100 28e2 r.E.....(
00000f0: d25f 2deb 0f0c baf5 570d d3f6 05df 6534 ._-.....W.....e4
0000100: 48d8 0853 00ae 3230 73a9 afb7 ac87 d834 H..S..20s.....4
0000110: f7e9 bb57 8ac1 1750 1200 0000 0000 0000 ...W...P.....
0000120: 0000 0000 0000 0000 003d 0600 0000 0f05 .....=.....
0000130: 0600 00c3 5057 0d45 7468 6572 6e65 7430 ...PW.Ethernet0
0000140: 2f30 181f 3236 5365 7373 696f 6e49 443d /0..26SessionID=
0000150: 6163 732f 3134 3531 3136 3739 372f 3132 acs/145116797/12
0000160: 3b04 06c0 a80a 0a ;.....
```

その変更後、ペイロードの準備が整います。16進数/バイナリモードに戻る必要があります (タイプ: "%!xxd -r")ファイル(":wq")を保存します。

4. HMAC-MD5を計算するには、OpenSSLを使用します。

```
pluton # cat packet30-clear-msgauth.bin | openssl dgst -md5 -hmac 'cisco'
(stdin)= 01418d3b1865556918269d3cf73608b0
```

HMAC-MD5関数は2つの引数を取ります。標準入力(stdin)からの最初の入力はメッセージ自体で、2番目の入力は共有秘密 (この例ではCisco) です。結果は、RADIUS Access-Requestパケットに接続されたMessage-Authenticatorとまったく同じ値になります。

Pythonスクリプトを使用しても同じことが計算できます。

```
pluton # cat hmac.py
#!/usr/bin/env python

import base64
import hmac
import hashlib

f = open('packet30-clear-msgauth.bin', 'rb')
try:
    body = f.read()
finally:
    f.close()

digest = hmac.new('cisco', body, hashlib.md5)
d=digest.hexdigest()
print d
```

```
pluton # python hmac.py  
01418d3b1865556918269d3cf73608b0
```

前の例では、Access-RequestからMessage-Authenticatorフィールドを計算する方法を示しています。Access-Challenge、Access-Accept、およびAccess-Rejectでは、ロジックはまったく同じですが、前のAccess-Requestパケットで提供されるRequest Authenticatorを使用する必要があることに注意してください。

関連情報

- [RFC 2865](#)
- [RFC 2866](#)
- [RFC 3579](#)
- [テクニカル サポートとドキュメント - Cisco Systems](#)