

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

目次

[概要](#)

[オーセンティケーター ヘッダ](#)

[応答の認証](#)

[いつ検証エラーを待つ必要がありますか。](#)

[パスワード隠れること](#)

[再送信](#)

[アカウントिंग](#)

[メッセージ オーセンティケーター アトリビュート](#)

[メッセージ オーセンティケーターはいつ使用する必要がありますか。](#)

[いつ検証エラーを待つ必要がありますか。](#)

[メッセージ オーセンティケーター アトリビュートを検証して下さい](#)

[関連情報](#)

概要

この資料は 2 つの RADIUS セキュリティ機構を解説したものです:

- オーセンティケーター ヘッダ
- メッセージ オーセンティケーター アトリビュート

この資料はどのように使用される、そしてとき検証エラーを待つ必要があるかものこれらのセキュリティ機構がである取り扱っています。

オーセンティケーター ヘッダ

RFC 2865 ごとに、オーセンティケーター ヘッダは長く 16 バイトです。それは Access-Request で使用されるとき、要求 オーセンティケーターと呼ばれます。それはあらゆる種類の応答で使用されるとき、応答オーセンティケーターと呼ばれます。それはのために使用されます:

- 応答の認証
- パスワード隠れること

応答の認証

サーバが正しい応答オーセンティケーターと応答する場合、クライアントはその応答が有効な要求と関連していた場合計算できます。

クライアントはランダム オーセンティケーター ヘッダとの要求を送信します。それから、サーバは共有秘密と共に応答を返す要求パケットの使用の応答オーセンティケーターを計算します:

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

応答を受け取るクライアントは同じオペレーションを行います。結果が同じである場合、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それが要求をスニффイングことはできなければ応答をスプーフィングできません。

いつ検証エラーを待つ必要がありますか。

検証エラーはスイッチが要求をもうキャッシュしない場合発生します (たとえば、タイムアウトが理由で)。共有秘密が無効 なときまたそれを経験するかもしれません (はい- Access-Reject はまたこのヘッダが含まれています)。こうすればは、ネットワーク アクセス デバイス (NAD) 共有秘密 ミスマッチを検出することができます。通常それは共有鍵 ミスマッチとして認証、許可、アカウントイング (AAA) クライアント/サーバによって報告されますが、詳細を明らかにしません。

パスワード隠れること

オーセンティケーター ヘッダも平文のユーザパスワード アトリビュートを送信 することを避けるために使用されます。最初に MD5 (MD5 -シークレット、オーセンティケーター) は計算されます。それからパスワードのチャンクとの複数の XOR オペレーションは実行されます。この方式は MD5 が強い一方向アルゴリズムとしてもう感知されないのでオフ・ライン不正侵入 (虹表) のために敏感です。

ユーザパスワードを計算する大蛇スクリプトはここにあります:

```
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 の属性のうちのどれかが (RADIUS ID のように、ユーザ名、等) 変更したら、新しいオーセンティケーター フィールドは生成し、それによって決まる他のフィールドはすべて再評価する必要があります。これが再送信である場合、何も変更する必要がありません。

アカウントイング

オーセンティケーター ヘッダの意味は Access-Request およびアカウントイング要求のために異なっています。

Access-Request に関しては、オーセンティケーターはランダムに生成され、正しく計算される応答はその特定の要求と関連していたと証明する ResponseAuthenticator の応答を受け取ることを期

待します。

アカウントリング要求のために、オーセンティケータはランダムではないですが、それは計算されず (RFC 2866 によって) :

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

こうすればは、サーバ計算し直された値がオーセンティケータ値を一致する場合会計 メッセージをすぐにチェックし、パケットを廃棄できます。 Identity Services Engine (ISE) 戻り:

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

このための一般的な原因は不正確な共有秘密 キーです。

メッセージ オーセンティケータ アトリビュート

メッセージ オーセンティケータ アトリビュートは RFC 3579 で定義される RADIUS 特性です。それは同じような目的で使用されます: 署名し、検証するため。応答要求を検証することをしかし今回、使用しません。

Access-Request (それをサーバである場合もあるアクセス チャレンジと応答する) 計算します Hash-Based Message Authentication Code を送信する クライアントはまた (自身のパケットからの HMAC)-MD5 はシグニチャとして、それからメッセージ オーセンティケータ アトリビュートを追加し。それから、サーバは確認できます同じオペレーションを行うことを。

数式はオーセンティケータ ヘッダに類似したに検知します:

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

HMAC-MD5 機能は 2 つの引数で奪取します:

- 16 バイト メッセージ オーセンティケータ フィールドが含まれているパケットのペイロードはゼロで、一杯になりました
- 共有秘密鍵

メッセージ オーセンティケータはいつ使用する必要がありますか。

メッセージ オーセンティケータは各パケットに使用する必要があります Extensible Authentication Protocol (EAP) メッセージ (RFC 3579) が含まれている。これにはアクセス チャレンジと応答するサーバおよび Access-Request を送信する クライアントが両方含まれています。反対側は検証が失敗した場合無言でパケットを廃棄する必要があります。

いつ検証エラーを待つ必要がありますか。

検証エラーは共有秘密が無効 なとき発生します。それから、AAAサーバは要求を検証できません。

ISE レポート:

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

これは通常 EAP メッセージを添付する後期に発生します。 802.1X セッションの最初の

RADIUSパケットは EAP メッセージが含まれていません; メッセージ オーセンティケータ フィールドがないし、要求を確認することはできませんがそのステージで、クライアントはオーセンティケータ フィールドの使用の応答を検証できます。

メッセージ オーセンティケータ アトリビュートを検証して下さい

確かめるために手動で値をどのように数えるか説明する例はここにあります正しく計算されることを。

パケット第 30 (Access-Request) は選択されました。それは EAP セッションの最中にあり、パケットはメッセージ オーセンティケータ フィールドが含まれています。目標はメッセージ オーセンティケータ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プロトコルを右クリックし、指定パケット バイトを『Export』 を選択 して下さい。
2. ファイル (バイナリーデータ) にその RADIUS ペイロードを書いて下さい。
3. メッセージ オーセンティケータ フィールドを計算するために、ゼロをそこに置き、HMAC-MD5 を計算して下さい。

たとえば、hex/バイナリーエディターを、vim のような使用する時、「入力した後: %! xxd は」"5012" 後十六進モードおよびゼロに開始する 16 バイト切り替える (50hex はメッセージ オーセンティケータ型の、12 属性値ペア (AVP) ヘッダを含んで 18 の) サイズです DEC の 80 であり:

```

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 ;.....

```

後その修正は、ペイロード準備ができています。hex/バイナリモード (型に戻ることは必要です: 「: %! xxd は -r」) ファイルを保存し、(「: wq」)。

4. HMAC-MD5 を計算するために OpenSSL を使用して下さい:

```

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

```

HMAC-MD5 機能は 2 つの引数を奪取します: 標準入力 (stdin) からの最初の 1 つはメッセージ自体および第 2 1 です共有秘密 (この例の Cisco) です。結果は RADIUS アクセス要求パケットに接続されるメッセージ オーセンティケータと丁度同じ値です。

同じは大蛇スクリプトの使用と計算することができます:

```

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

```

前例は Access-Request からのメッセージ オーセンティケーター フィールドを計算する方法を示します。アクセス チャレンジ、Access-Accept および Access-Reject に関しては、ロジックは丁度同じですが、前のアクセス要求パケットで提供される要求 オーセンティケーターが使用する必要があることを覚えておくことは重要です。

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](#)