# 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 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](#)
[アカウンティング](#)
[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 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 ヘッダ
- メッセージ オーセンティケータ 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 ヘッダ

RFC 2865 ごとに、オーセンティケータ 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 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ッフィ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、アカウンティング（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 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）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 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。

## アカウンティング

オーセンティケータ ヘッダの意味は Access-Request およびアカウンティ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。

アカウンティング要求のために、オーセンティケータはランダムではないですが、それは計算されます（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
```

これのための一般的 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.
```

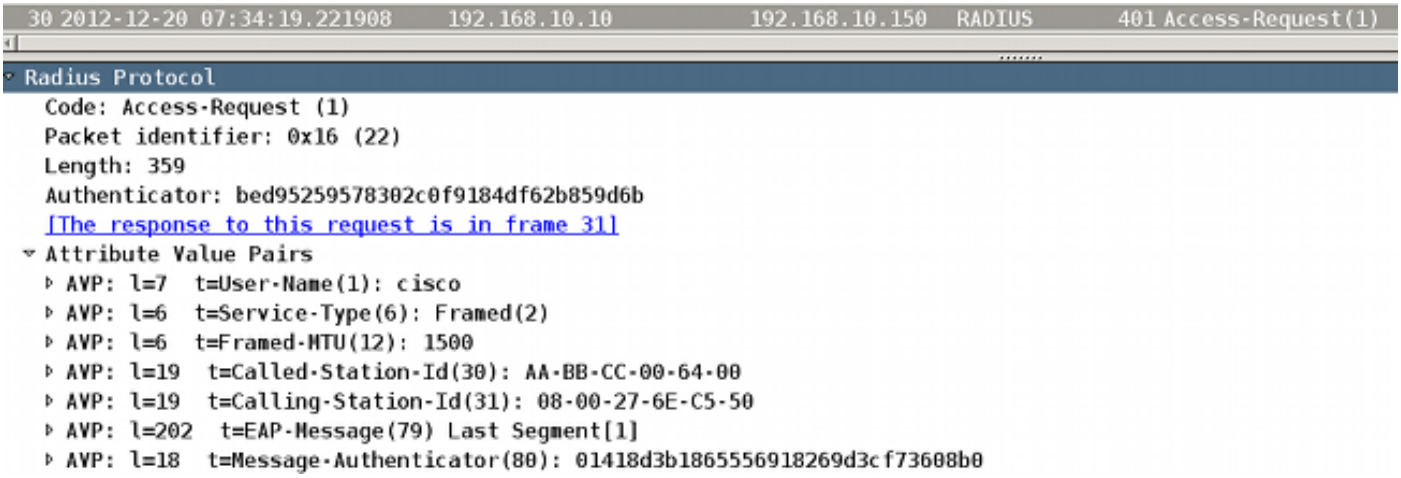
これは通常 EAP メッセージを添付する後期に発生します。 802.1X セッションの最初の

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:

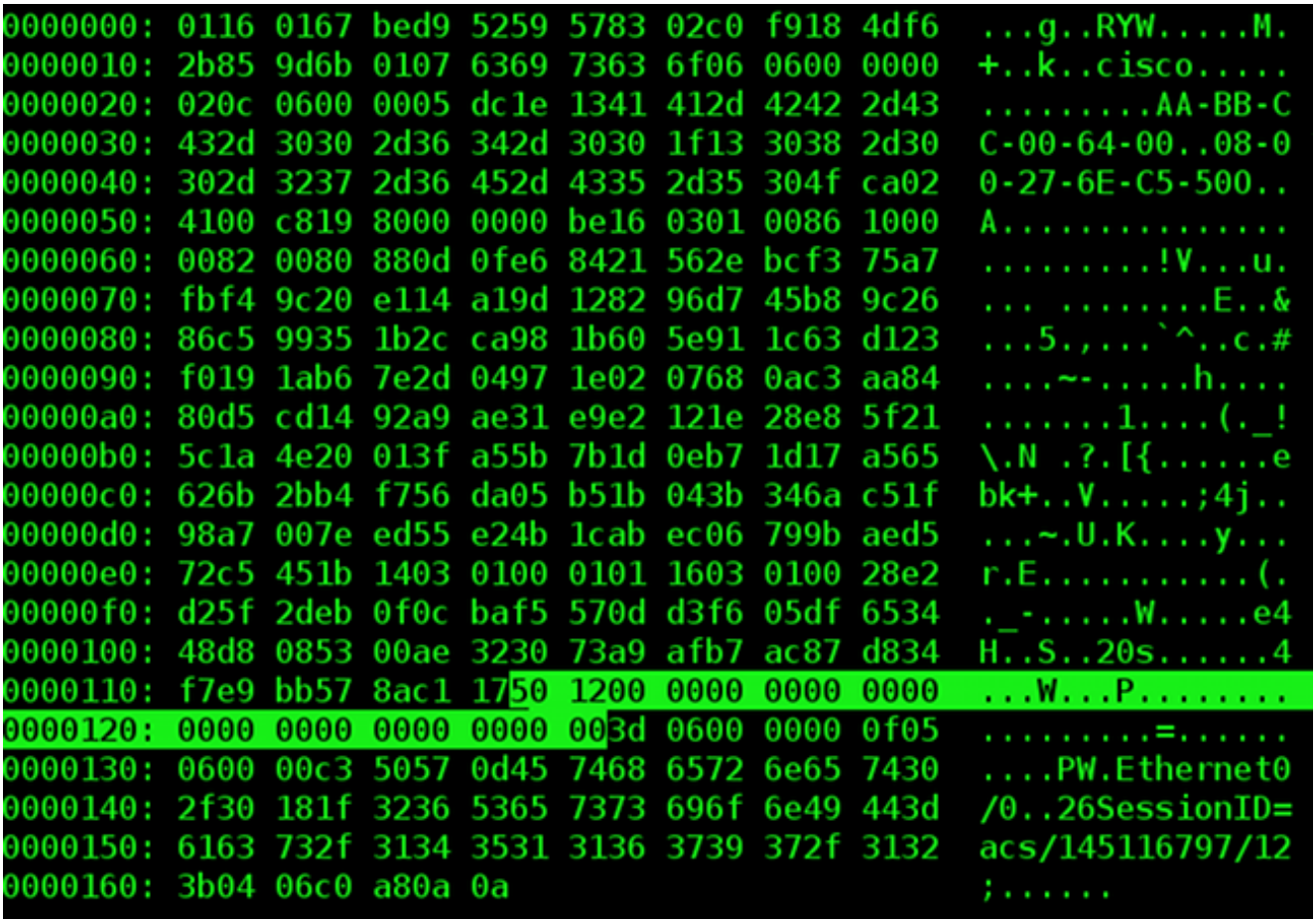



1. **RADIUSプロトコルを**右クリックし、**指定パケット バイトを**『Export』 を選択 して下さい。

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/バイナリー工ディターを、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:




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
```

HMAD-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 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