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

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ヘッダー

RFC 2865によると、オーセンティケータヘッダーの長さは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、要求をスニフィングできない限り、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、アカウンティング(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も使用されます。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（レインボテーブル）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。

アカウンティング

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ティング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。

アカウンティング要求の場合、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、サーバはアカウンティング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。

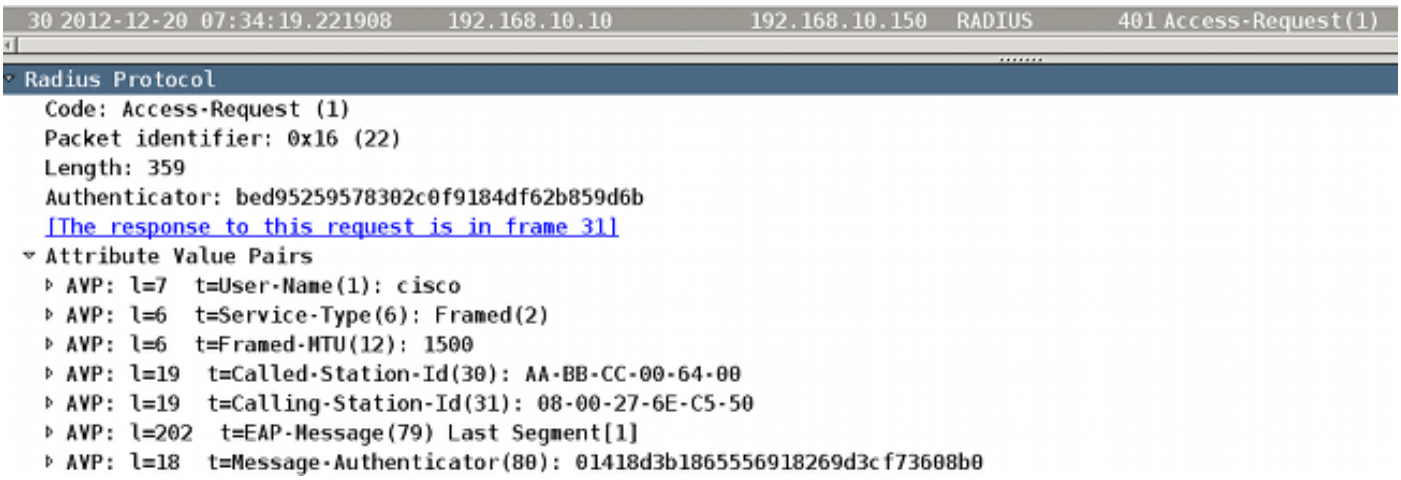
```
11036 The Message-Authenticator Radius Attribute is invalid.
```

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。

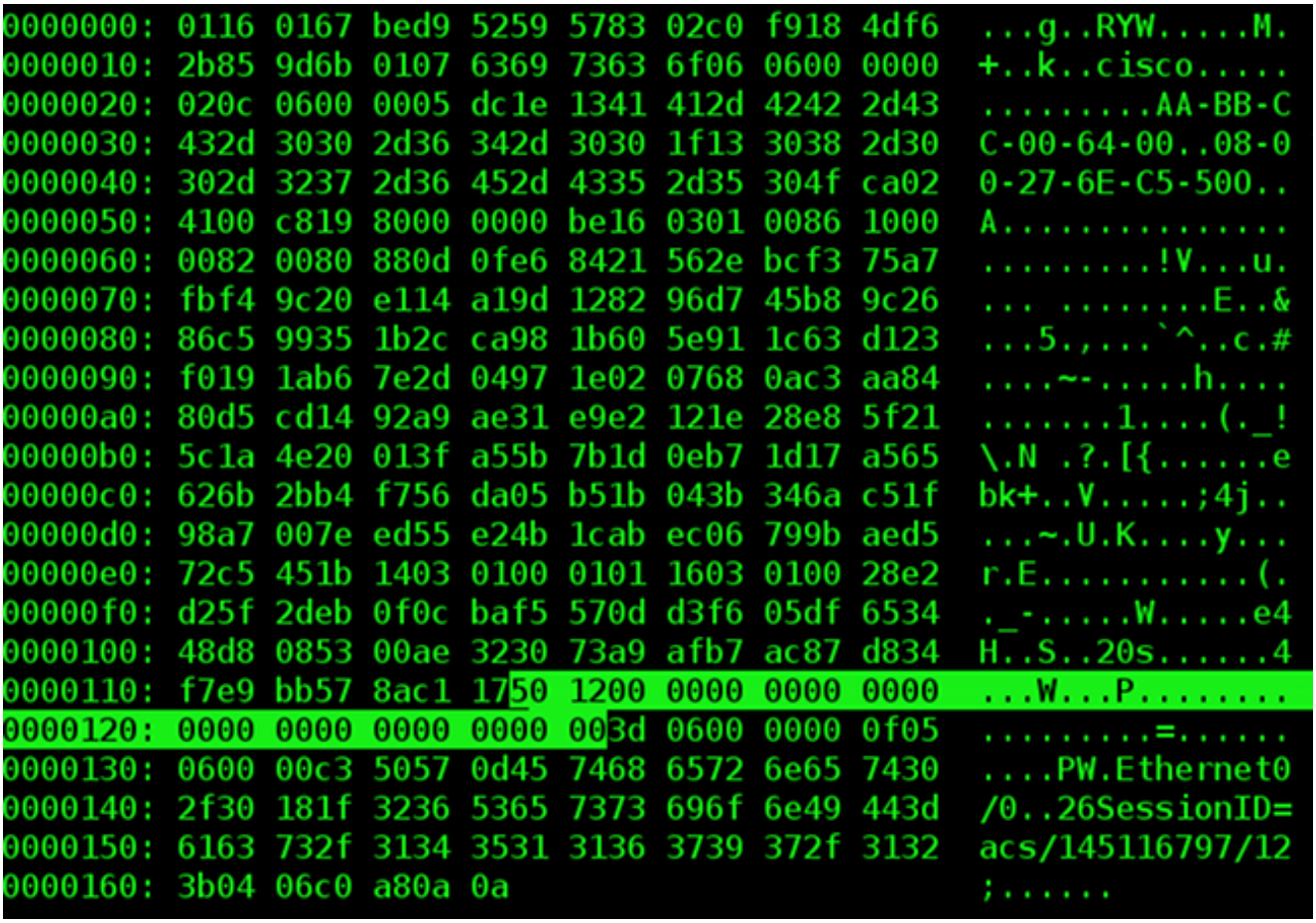


1. [Radius Protocol]を右クリックし、[Export selected packet bytes]を選択します。

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/バイナリ工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):



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
```

HMAD-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