RADIUS Invalid Authenticator and Message-Authenticator Troubleshoot Guide

Document ID: 118673

Contributed by Michal Garcarz, Cisco TAC Engineer.
Jan 20, 2016

Contents

Introduction

This document describes two RADIUS security mechanisms:

- Authenticator Header
- Message-Authenticator attribute

This document covers what these security mechanisms are, how they are used, and when you should expect validation failure.

Authenticator Header

Per RFC 2865, the Authenticator Header is 16 bytes long. When it is used in an Access-Request, it is called a Request Authenticator. When it is used in any kind of response, it is called a Response Authenticator. It is used for:

- Authentication of response
- Password hiding

Authentication of Response

If the server responds with the correct Response Authenticator, the client can compute if that response was related to a valid request.

The client sends the request with the random Authenticator Header. Then, the server that sends the response calculates the Response Authenticator with the use of the request packet along with the shared secret:

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

The client that receives the response performs the same operation. If the result is the same, the packet is correct.

Note: The attacker that knows the secret value is not able to spoof the response unless it is able to sniff the request.

When should you expect validation failure?

Validation failure occurs if the switch does not cache the request anymore (for example, because of timeout). You might also experience it when the shared secret is invalid (yes - Access-Reject also includes this header). This way, the Network Access Device (NAD) can detect the shared secret mismatch. Usually it is reported by Authentication, Authorization, and Accounting (AAA) clients/servers as a shared key mismatch, but it does not reveal the details.

Password Hiding

The Authenticator Header is also used in order to avoid sending the User-Password attribute in plain text. First the Message Digest 5 (MD5 - secret, authenticator) is computed. Then several XOR operations with the chunks of the password are executed. This method is susceptible for offline attacks (rainbow tables) because MD5 is not perceived as a strong one-way algorithm anymore.

Here is the Python script that computes the User-Password:

```
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]))
```

Retransmissions

If any of the attributes in the RADIUS Access-Request have changed (like the RADIUS ID, User-Name, and so on), the new Authenticator field should be generated and all other fields that depend on it should be recomputed. If this is a retransmission, nothing should change.

Accounting

The meaning of the Authenticator Header is different for an Access-Request and an Accounting-Request.

For an Access-Request, the Authenticator is generated randomly and it is expected to receive a response with the ResponseAuthenticator calculated correctly, which proves that the response was related to that specific request.

For an Accounting-Request, the Authenticator is not random, but it is calculated (as per RFC 2866):

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

This way, the server can check the accounting message immediately and drop the packet if the recalculated value does not match the Authenticator value. The Identity Services Engine (ISE) returns:

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

The typical reason for this is the incorrect shared secret key.

Message-Authenticator Attribute

The Message-Authenticator attribute is the RADIUS attribute defined in RFC 3579. It is used for a similar purpose: to sign and validate. But this time, it is not used in order to validate a response but a request.

The client that sends an Access-Request (it can also be a server that responds with an Access-Challenge) computes the Hash-Based Message Authentication Code (HMAC)-MD5 from its own packet, and then adds the Message-Authenticator attribute as a signature. Then, the server is able to verify it performs the same operation.

The formula looks similar to the Authenticator Header:

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

The HMAC-MD5 function takes in two arguments:

- The payload of the packet, which includes the 16 byte Message-Authenticator field filled with zeros
- The shared secret

When should the Message-Authenticator be used?

The Message-Authenticator MUST be used for every packet, which includes the Extensible Authentication Protocol (EAP) message (RFC 3579). This includes both the client that sends the Access-Request and the server that responds with the Access-Challenge. The other side should silently drop the packet if validation fails.

When should you expect validation failure?

Validation failure will occur when the shared secret is invalid. Then, the AAA server is not able to validate the request.

The ISE reports:

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

This usually occurs at the later stage when the EAP message is attached. The first RADIUS packet of the 802.1x session does not include the EAP message; there is no Message-Authenticator field and it is not possible to verify the request, but at that stage, the client is able to validate the response with the use of the Authenticator field.

Validate the Message-Authenticator Attribute

Here is an example to illustrate how you manually count the value in order to make sure it is computed correctly.

Packet number 30 (Access-Request) has been chosen. It is in the middle of the EAP session, and the packet includes the Message-Authenticator field. The aim is to verify that the Message-Authenticator is correct:

```
 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): 01418d3b1865556918269d3cf73608b0
```

1. Right-click **Radius Protocol** and choose **Export selected packet bytes**.

2. Write that RADIUS payload to a file (binary data).

3. In order to compute Message-Authenticator field, you must put zeros there and compute the HMAC-MD5.

   For example, when you use hex/binary editor, such as vim, after you type ":%!xxd", which switches to hex mode and zeroes 16 bytes starting after "5012" (50hex is 80 in dec which is Message-Authenticator type, and 12 is the size which is 18 including the Attribute Value Pairs (AVP) header):

```
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                        ;.......
```

   After that modification, the payload is ready. It is necessary to return back to hex/binary mode (type: ":%!xxd -r") and save the file (":wq").

4. Use OpenSSL in order to compute HMAC-MD5:

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

The HMAD-MD5 function takes two arguments: the first one from standard input (stdin) is the message itself and the second one is the shared secret (Cisco in this example). The result is exactly the same value as the Message-Authenticator attached to the RADIUS Access-Request packet.

The same can be computed with the use of the Python script:

```
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
```

The previous example presents how to calculate the Message-Authenticator field from the Access-Request. For Access-Challenge, Access-Accept, and Access-Reject, the logic is exactly the same, but it is important to remember that Request Authenticator should be used, which is provided in the previous Access-Request packet.

Related Information

- **RFC 2865**
- **RFC 2866**
- **RFC 3579**
- **Technical Support & Documentation - Cisco Systems**

Updated: Jan 20, 2016 Document ID: 118673