WHITE PAPER

Detailing the Message-Authenticator Attribute

TABLE OF CONTENTS

Overview

The BLAST-RADIUS cyber attack caused a wave of updates to RADIUS handling. Most implementations picked an off-the-shelf spec for more secure RADIUS handling, adding the `Message-Authenticator` attribute; additional details are available in the following memos: RFC 3579, section 3.2 and RFC 2869, page 33. This white paper examines the `Message-Authenticator` attribute in detail, providing functional examples and illumination.

Baseline: Access-Accept, Access-Reject

In an authentication scenario, a user will need to authenticate to some application. That application, in turn, may communicate via RADIUS (using an `Access-Request` RADIUS message) as a RADIUS client to some RADIUS server. That RADIUS server will check the credentials supplied and will typically render a final response: Let the user in (`Access-Accept`) or don't let the user in (`Access-Reject`). In order to get a picture of the relevance of `Message-Authenticator`, we need to see what baseline is—what normal processing looks like prior to `Message-Authenticator`.

Authentication Success

The following Wireshark screen captures show a successful authentication.

```
▼ RADIUS Protocol
    Code: Access-Request (1)
    Packet identifier: 0xa5 (165)
    Length: 74
    Authenticator: 896308b3cabded84959b629eb6c294ba
    [The response to this request is in frame 1224]
  ▼ Attribute Value Pairs
    › AVP: t=User-Name(1) l=8 val=bsmith
    › AVP: t=User-Password(2) l=18 val=Encrypted
    › AVP: t=NAS-Identifier(32) l=22 val=OpenVPN.d-openvpn-vm
    › AVP: t=Service-Type(6) l=6 val=Authenticate-Only(8)

▼ RADIUS Protocol
    Code: Access-Accept (2)
    Packet identifier: 0xa5 (165)
    Length: 104
    Authenticator: 8de1627aeb22a088000ae38ba14faf2f
    [This is a response to a request in frame 1113]
    [Time from request: 4.462580000 seconds]
  ▼ Attribute Value Pairs
    › AVP: t=NAS-Filter-Rule(92) l=44 val=ACCESS_CHALLENGE_FOR_NUMBER_CHALLENGE_PUSH
    › AVP: t=Reply-Message(18) l=40 val=reason=0&tokenid=SYMC47572971; Success
```

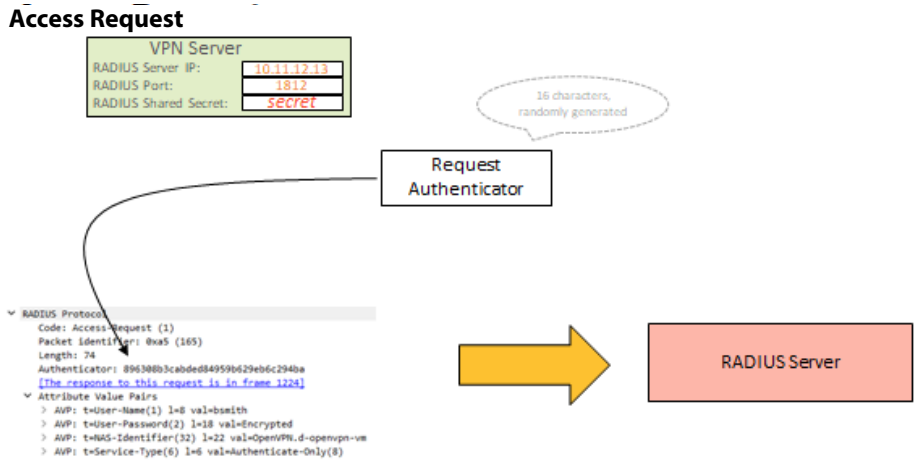
Authentication Failure

A failed authentication displays as follows.

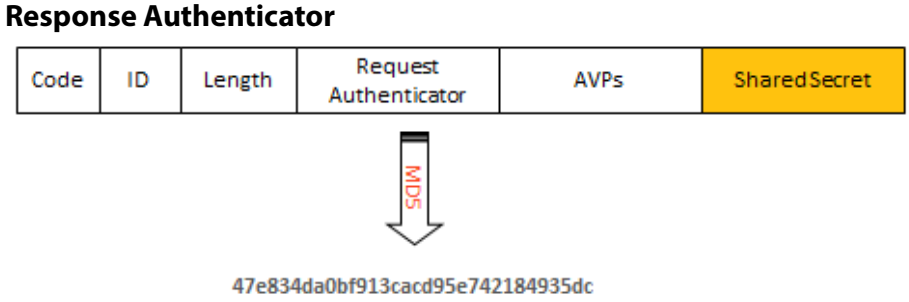
```
RADIUS Protocol
    Code: Access-Request (1)
    Packet identifier: 0xa7 (167)
    Length: 74
    Authenticator: c26c91f9ee9f55c9fdf55120f09b716d
  Attribute Value Pairs
    > AVP: t=User-Name(1) l=8 val=bsmith
    > AVP: t=User-Password(2) l=18 val=Encrypted
    > AVP: t=NAS-Identifier(32) l=22 val=OpenVPN.d-openvpn-vm
    > AVP: t=Service-Type(6) l=6 val=Authenticate-Only(8)

RADIUS Protocol
    Code: Access-Reject (3)
    Packet identifier: 0xa7 (167)
    Length: 100
    Authenticator: 47e834da0bf913cacd95e742184935dc
    [This is a response to a request in frame 875]
    [Time from request: 0.056733000 seconds]
  Attribute Value Pairs
    > AVP: t=NAS-Filter-Rule(92) l=44 val=ACCESS_CHALLENGE_FOR_NUMBER_CHALLENGE_PUSH
    > AVP: t=Reply-Message(18) l=36 val=reason=3; Incorrect LDAP Password.
```

The following illustration explains how those messages are handled.

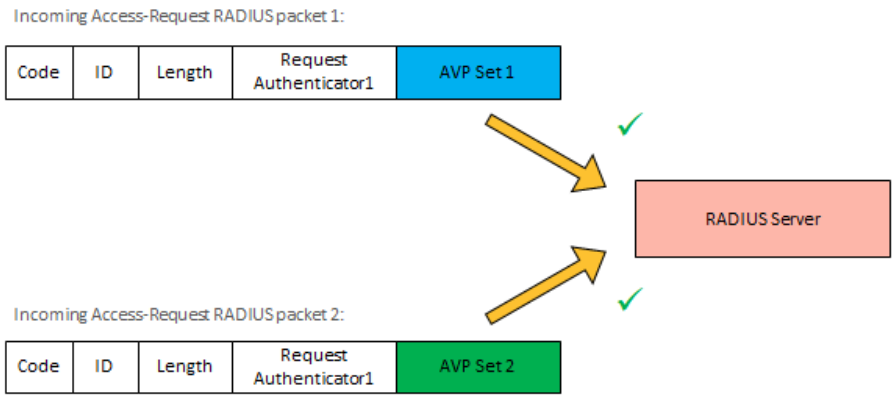


Note that the Authenticator attribute (Request Authenticator) shown above is a 16-octet random value and is not based on other values; additional information is available in the RADIUS RFC. The `Access-Accept` and `Access-Reject` responses use that Request Authenticator as input when generating a Response Authenticator, as detailed in the RADIUS RFC as well as in the following image.



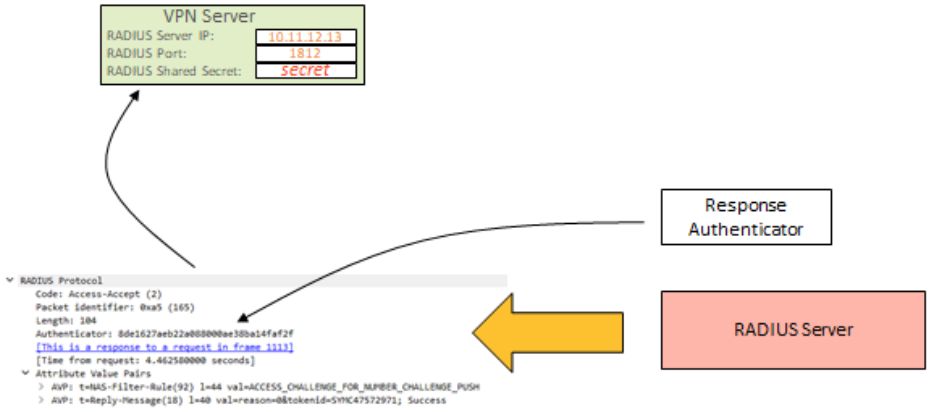
If a RADIUS server received two RADIUS packets from what appears to be the same IP address and with the same Request Authenticator value, but with different AVPs, it should process it normally. This is the entry point for BLAST-RADIUS that `Message-Authenticator` closes.

Access Request



Finally, the RADIUS server sends back the response:

RADIUS Server Response



The Message-Authenticator Attribute

When we get to the `Message-Authenticator` attribute, we see an additional Attribute Value Pair. The `Message-Authenticator` attribute itself is similar to the Response Authenticator:

Response Authenticator Calculation:

MD5 (Code + ID + Length + RequestAuth + Attributes + Secret)

The Response Authenticator calculation is straightforward: concatenate all these values together and use this as input to the MD5 function.

Message-Authenticator Calculation:

HMAC-MD5 (Type, Identifier, Length, Request Authenticator, Attributes), using the shared secret as the key

However, the HMAC-MD5 function is more complex:

- First, essentially all of the values in the UDP datagram are concatenated together.
- Then the shared secret is used as the key.

- Those values are entered into the HMAC-MD5 function, which is described in RFC 2104—the appendix there has a good illustration. *ipad* and *opad* values each modify the key and the text to provide some baseline unpredictability. We expect for any small change to any attribute that would necessarily cause a big and unpredictable change in the `Message-Authenticator` value.

Notably, the BLAST-RADIUS attack relied upon being able to predict or control values in the initial server response. It then proceeded to falsify those values by dropping stuff and making it look *close*, so the colliding MD5 values *checked out* against the rest of the packet. This yielded an elegant chosen-prefix attack that leveraged some unfortunately helpful RADIUS attributes and behavior.

THE BLAST-RADIUS ATTACK RELIED UPON BEING ABLE TO PREDICT OR CONTROL VALUES IN THE INITIAL SERVER RESPONSE.

BY EXTENDING PACKET CONTENT VALIDATION TO REQUESTS AND RESPONSES, WE'RE EFFECTIVELY PLACING A GREAT BIG BAND AID ON THIS ISSUE.

By extending the packet content validation to requests and responses, we're effectively placing a great big band aid on this issue.

The `Message-Authenticator` Calculation

Let's dissect a `Message-Authenticator` example:

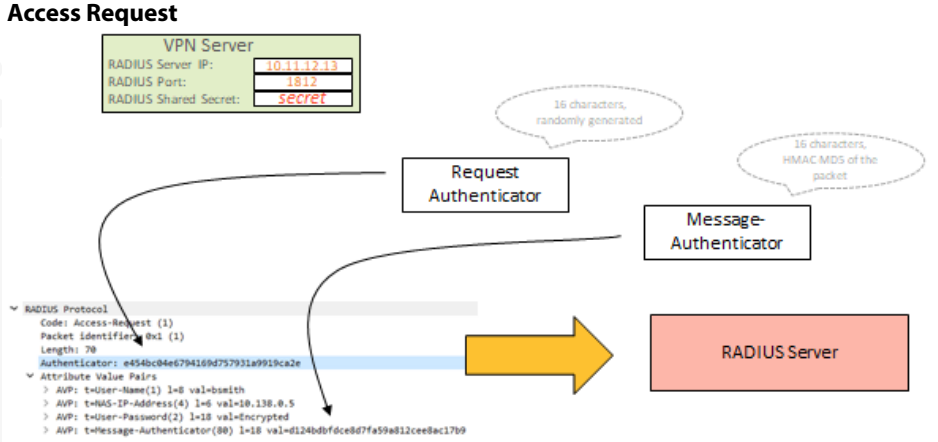
The `Message-Authenticator` Attribute Value Pair (AVP) in a successful authentication is displayed below.

```
▼ RADIUS Protocol
    Code: Access-Request (1)
    Packet identifier: 0x1 (1)
    Length: 70
    Authenticator: e454bc04e6794169d757931a9919ca2e
  ▼ Attribute Value Pairs
    › AVP: t=User-Name(1) l=8 val=bsmith
    › AVP: t=NAS-IP-Address(4) l=6 val=10.138.0.5
    › AVP: t=User-Password(2) l=18 val=Encrypted
    › AVP: t=Message-Authenticator(80) l=18 val=d124bdbfdce8d7fa59a812cee8ac17b9

▼ RADIUS Protocol
    Code: Access-Accept (2)
    Packet identifier: 0x1 (1)
    Length: 78
    Authenticator: 197d9324001947a67df1aab396df4ccf
    [This is a response to a request in frame 25]
    [Time from request: 4.246241000 seconds]
  ▼ Attribute Value Pairs
    › AVP: t=Message-Authenticator(80) l=18 val=efc2f30af5254ec325f77751b1b697cb
    › AVP: t=Reply-Message(18) l=40 val=reason=0&tokenid=SYMC47572971; Success
```

Here is the formation of the `Message-Authenticator` at the RADIUS client (NAS or VPN Server):

Access Request



`Message-Authenticator` Example

The data fed in to the HMAC-MD5 function for the Access-Request is as follows:

Code/Type: **1** (this is a one-byte value)
Packet Identifier: **1** (this is a one-byte value)
Length: **70** (or 0x46 in hex, a two-byte value)
Request Authenticator: **e454bc04e6794169d757931a9919ca2e** (16 byte random data)

AVPs:
Type: **1** (one byte value)
Length: **8** (one byte value)
Value: **bsmith** (6 bytes for bsmith, one byte for the type and one byte for the length)

Type: **4** (one byte value)
Length: **6** (one byte value)
Value: **0x0a8a0005** (four bytes)

Type: **2** (one byte value)
Length: **18** (one byte value)
Value: **0x4a02e1d6 0xb14c2ee5 0x0ca12884 0x0440abe5** (16 bytes)

Type: **80** (one byte value)
Length: **18** (one byte value)
Value: **0x00000000 0x00000000 0x00000000 0x00000000** (16 bytes)
Note: The signature string referenced in the RFCs is the `Message-Authenticator` AVP, but with a value of all zeroes.

Concatenating these values provides the bytestream from the packet (in hex):

01010046e454bc04e6794169d757931a9919ca2e010862736d69746804060a8a000502124a02e1d6b14c2ee50ca128840440abe550120000000000000000000000000000000000

Now stepping through the HMAC steps:

1. Take the shared secret and add zeroes until it is 64 bytes long (== 512 bits == chunks that MD5 operates on). The shared secret is represented by question marks:
   ???????? ???????? ???????? ???????? 00000000 00000000 00000000 00000000
2. XOR **the above** with ipad (0x36 repeated):
   ???????? ???????? ???????? ???????? 36363636 36363636 36363636 36363636
3. Tack onto **the above** the bytestream from the packet:
   ???????? ???????? ???????? ???????? 36363636 36363636 36363636 36363636 01010046 e454bc04 e6794169 d757931a 9919ca2e 01086273 6d697468 04060a8a 00050212 4a02e1d6 b14c2ee5 0ca12884 0440abe5 50120000 00000000 00000000 00000000 0000
4. Throw that data into MD5:
   0f38c805be11ecebe8bb2a6730cb357d
5. XOR **the result in step 1** with opad (0x5C repeated):
   ???????? ???????? ???????? ???????? 5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C
6. Append **the result** from step 4 onto this:
   ???????? ???????? ???????? ???????? 5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C 0f38c805 be11eceb e8bb2a67 30cb357d
7. Take the above and feed it to MD5 slowly:
   d124bdbfdce8d7fa59a812cee8ac17b9

The result shown in Step 7 is the HMAC, and its value should—and does—match the `Message-Authenticator` value sent in the Access Request.

ADDITIONAL HELPFUL TOOLS

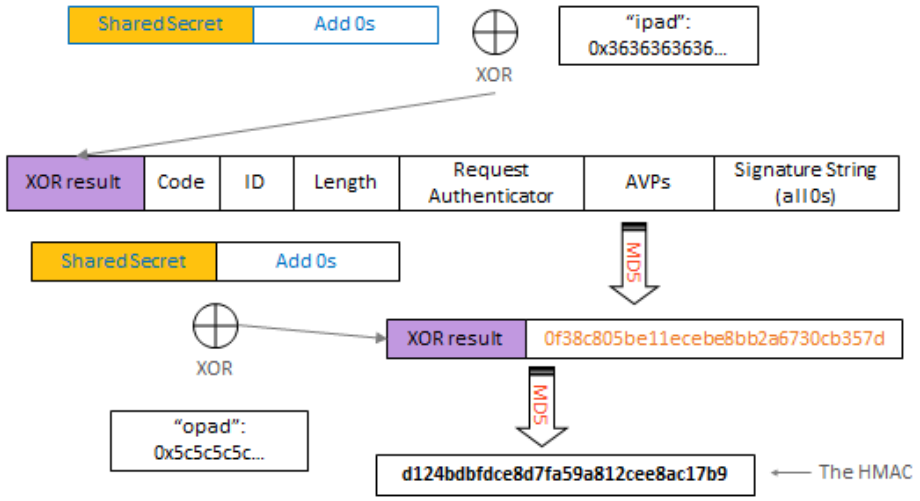
If you would like to reproduce some of these calculations, you can use command line tools such as `hexdump`, `od`, and `xxd -r`.

Online HMAC and md5 tools are also helpful:

- **Cryptii** - An online HMAC calculator that accepts hex as input
- **emn178** - An online md5 calculator that accepts hex as input

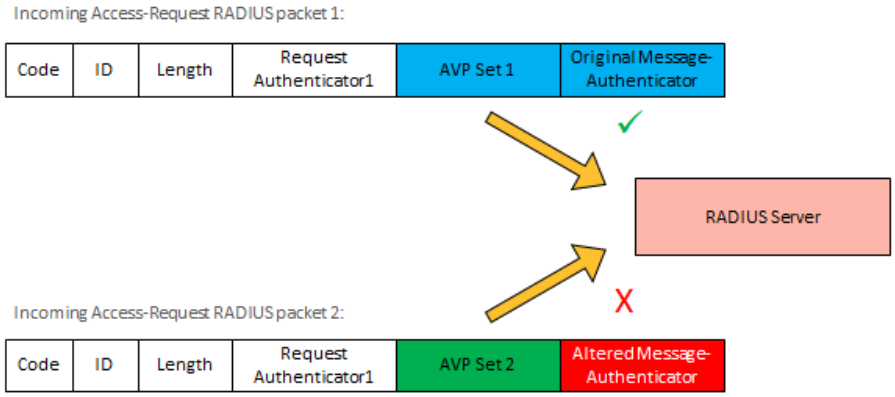
The following image illustrates the processing at work:

Message-Authenticator



A would-be attacker that simply modified values in the packet by adding or dropping AVPs, which was needed for the HASH-CLASH attack, would drastically alter the `Message-Authenticator` value. The RADIUS server would quietly drop that bad packet.

Access Request



The RADIUS client (a NAS—for example: a VPN Server) performs this calculation and places the HMAC into the `Message-Authenticator` attribute. The RADIUS server repeats this calculation and confirms that it gets the same answer. If it does, it's most likely from the actual NAS. If not, it's still most likely to be a random error so the RADIUS server won't reply—but we need to throw it on the floor silently, as it might be beneficial to have debug messages that log this. A RADIUS server that tried to provide a helpful reply may run us afoul of things like the Bleichenbacher chosen-ciphertext attack, which would be bad.

This process is repeated for the message from the RADIUS server to the RADIUS client.

VIP-BR-MA-WP100 September 24, 2024