Workgroup: RADEXT Working Group
Internet-Draft:
draft-dekok-radext-request-authenticator-04
Published: 1 October 2022
Intended Status: Standards Track
Expires: 4 April 2023
Authors: A. DeKok
FreeRADIUS

Correlating requests and replies in the Remote Authentication Dial In User Service (RADIUS) Protocol via the Request Authenticator.

Abstract

RADIUS uses a one-octet Identifier field to correlate requests and responses, which limits clients to 256 "in flight" packets per connection. This document removes that limitation by allowing Request Authenticator to be used as an additional field for identifying packets. The capability is negotiated on a per-connection basis, and requires no changes to the RADIUS packet format, attribute encoding, or data types.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-dekok-radext-request-authenticator/.

Discussion of this document takes place on the RADEXT Working Group mailing list (mailto:radext@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/radext/.

Source for this draft and an issue tracker can be found at https://github.com/freeradius/extended-id.git.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 April 2023.

Copyright Notice

Table of Contents

1. Introduction

The Remote Authentication Dial In User Service (RADIUS) protocol contains an Identifier field, defined in [RFC2865] Section 5 as:

> The Identifier field is one octet, and aids in matching requests and replies. The RADIUS server can detect a duplicate request if it has the same client source IP address and source UDP port and Identifier within a short span of time.

The small size of the field allows for only 256 outstanding requests without responses. If a client requires more than 256 packets to be outstanding to the RADIUS server, it must open a new connection, with a new source port.

This limitation does not severely impact low-load RADIUS systems. However, it is an issue for high-load systems. Opening new sockets is more expensive than tracking requests inside of an application, and is generally unnecessary in other UDP protocols.

For very high load systems, this "new socket" requirement can result in a client opening hundreds or thousands of connections. There are a number of problems with this approach:

* RADIUS is connection oriented, and each connection operates independently of all other connections.

* each connection created by the client must independently discover server availability. i.e. the connections can have different views of the status of the server, leading to packet loss and network instability.

* The small size of RADIUS packets means that UDP traffic can reach Ethernet rate limits long before bandwidth limits are reached for the same network. This limitation prevents high-load systems from fully using available network bandwidth.

* The limit of 256 outstanding requests means that RADIUS over TCP [RFC6613] is also limited to a small amount of traffic per connection, and thus will rarely achieve the full benefit of TCP transport.

* the existence of hundreds of simultaneous connections can impose significant management overhead on clients and servers.

* network stacks will generally operate more efficiently with a larger amount of data over one connection, instead of small amounts of data split over.in -0.2i

For these reasons, it is beneficial to extend RADIUS to allow more than 256 outstanding requests per connection.

1.1. Compatability with Existing RADIUS

It is difficult in practice to extend RADIUS. Any proposal must not only explain why it cannot use Diameter, but it also must fit within the technical limitations of RADIUS.

We believe that this specification is appropriate for RADIUS, due to the following reasons:

* this specification makes no change to the RADIUS packet format;

* this specification makes no change to RADIUS security;

* this specification adds no new RADIUS data types;

* this specification uses standard RADIUS attribute formats;

* this specification uses standard RADIUS data types;

* this specification adds a single attribute to the RADIUS Attribute Type registry;

* all existing RADIUS clients and servers will accept packets following this specification as valid RADIUS packets;

* due to negotiation of capabilities, implementations of this specification are fully compatible with all existing RADIUS implementations;

* clients implementing this specification act as traditional RADIUS clients until they see a matching response from a server, so errors due to client misconfiguration is impossible.

* clients implementing this specification can fall back to standard RADIUS in the event of misbehavior by a server;

* servers implementing this specification use standard RADIUS unless this functionality has been explicitly negotiated;

* low-load RADIUS systems do not need to implement this specification;

* high-load systems can use this specification to remove all RADIUS-specific limits on filling available network bandwidth;

* this specification allows for effectively unlimited numbers of RADIUS packets over one connection, removing almost all issues related to connection management from client and server;

* as a negative, implementations of this specification must have code paths this specification, as well as for standard RADIUS.

In short, this specification is largely limited to changing the way that clients and server implementations internally match requests and responses.

We believe that the benefits of this specification outweigh the costs

1.2. Outline of the document

The document gives a high level overview of proposal. It then describes how the functionality is signaled in a Status-Server [RFC5997] It then defines the Original-Request-Authenticator attribute. It then describes how this change to RADIUS affects each type of packet (ordered by Code). Finally, it describes how the change affects transports such as UDP, TCP, and TLS.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following terms:

ID tracking

: The traditional RADIUS method of tracking request / response packets by use of the Identifier field. Many implementations also use a socket descriptor, and/or src/dst IP/port as part of the tuple used to track packets.

ORA tracking

: The method defined here which allows request / response packets extends ID tracking, in order to add Request Authenticator or

Original-Request-Authenticator as part of the tuple used to track packets.

Network Access Server (NAS)

The device providing access to the network. Also known as the Authenticator (in IEEE 802.1X terminology) or RADIUS client.

RADIUS Proxy

In order to provide for the routing of RADIUS authentication and accounting requests, a RADIUS proxy can be employed. To the NAS, the RADIUS proxy appears to act as a RADIUS server, and to the RADIUS server, the proxy appears to act as a RADIUS client.

'request packet' or 'request'

One of the allowed packets sent by a client to a server. e.g. Access-Request, Accounting-Request, etc.

'response packet' or 'response'

One of the allowed packets sent by a server to a client, in response to a request packet. e.g. Access-Accept, Accounting-Response, etc.

2. Review of Current Behavior

In this section, we give a short review of how clients and servers currently operate. This review is necessary to help contextualize the subsequent discussion.

2.1. Client Behavior

When a client sends a request to a server, it allocates a unique Identifier for that packet, signs the packet, and sends it to the server over a particular network connection. When a client receives a response from a server over that same network connection, the client uses the Identifier to find the original request. The client then uses the original Request Authenticator to verify the Response Authenticator of the response.

We call this behavior "ID tracking". It is the traditional method used in RADIUS to track requests and responses. The term "ID tracking" is used in preference to "traditional RADIUS".

This tracking behavior is similar across all packet types. However, the management of the ID field is largely unspecified. For example. [RFC2865] Section 5 says

> The Identifier field is one octet, and aids in matching requests and replies. The RADIUS server can detect a duplicate request if it has the same client source IP address and source UDP port and Identifier within a short span of time.

Similar text is contained in [RFC2866] Section 3, while [RFC5176] Section 2.3 has a more extensive discussion of the use of the Identifier field. However, all three documents are silent on the topic of how Identifiers are managed.

This silence means that there are no guidelines in the specifications for how a client can re-use an Identifier value. For example, can a client re-use an Identifier after a response is received? Can a client re-use an Identifier after a timeout, when no response has been received? If the client sends multiple requests and finally receives a response, can the Identifier be re-used immediately, or should the client wait until it receives all duplicate responses to its duplicate requests?

There are no clear answers to any of these questions.

The specifications are not much clearer on the subject of response packets. For example, [RFC2865] Section 4.2 has the following text for Access-Accept responses:

> On reception of an Access-Accept, the Identifier field is matched with a pending Access-Request. The Response Authenticator field MUST contain the correct response for the pending Access-Request. Invalid packets are silently discarded.

[RFC2866] Section 4.2 has similar text, while [RFC5176] has no such text, and assumes that the reader knows that Identifiers are used to match a CoA-ACK packet to a CoA-Request packet.

While these issues are undefined, they nevertheless have to be addressed in practice. Client implementations have therefore chosen some custom behavior for Identifier management. This behavior has proven to be inter-operable, despite being poorly defined.

We bring this issue up solely to note that much of the RADIUS protocol is undefined or implementation-defined. As a result, it should be possible to define behavior which was previously left open to implementation interpretation. Even better, this new behavior can be defined in such a way as to obtain new and useful features in RADIUS.

2.2. Server Behavior

Server implementations have similar issues related to managing Identifiers for request packets. For example, while clients are permitted to send duplicate requests, [RFC2865] is not clear on how servers should handle those duplicates.

That issue was addressed in [RFC5080] Section 2.2.2, which defines how servers should perform duplicate detection. This duplicate detection aids in lowering the load on servers by allowing them to send cached responses to duplicates, instead of re-processing the request.

However, the issue of duplicate packets and retransmissions by the clients can result in another situation which is not discussed in any RADIUS specification. This topic deserves more discussion, because as we will see below, this topic motivates this specification.

The specifications do not describe what a server should do if it receives a new packet which is on the same connection as a previous one, and shares the Code and Identifier fields, but for which the Request Authenticator is different. That is, a client may send a request using an Identifier, not get a response, then time out that request. The client is then implicitly allowed by the specifications to re-use that Identifier when sending a new request.

That new request will be sent on the same connection as a previous request, using the same Code and Identifiers as a previous request, but will have a different Request Authenticator.

When the server receives this new packet, it has no knowledge that the client has timed out the original request. The server may still be processing the original request. If the server responds to the original request, the response will likely get ignored by the client, as it has timed out that original request. If the server responds to the new request, the client will probably accept the response, but the server must then not respond to the original request.

The server now has two "live" requests from a client, but it can respond only to one. These request are "conflicting packets", and the process used to detect them is conflict detection and conflict management.

While the concept of "conflicting packets" is not defined in the RADIUS specifications, it nevertheless has had to be addressed in practice. Server implementations have each chosen some behavior for conflict detection and management. This implementation-defined

behavior has proven to be inter-operable in practice, which allows RADIUS to operate in the face of conflicts.

The concept conflict detection is the key concept driving this specification. If servers were instead allowed to process and reply to both requests, then the limitations of the Identifier field would be entirely bypassed.

The rest of this specification describes the impact of allowing these otherwise "conflicting packets" to be processed. We discuss a framework required to negotiate this functionality in an inter-operable manner. We define a new method of tracking packets, called "ORA tracking", which is discussed further below.

3. Alternative Approaches

There are other ways that the per-connection Identifier limitation could have been addressed. As extending RADIUS is a controversial topic, we believe it is useful to discuss some alternative approaches, and to explain their costs and benefits. This discussion ensures that readers of this specification are fully informed as to why this design is the preferred approach.

We finish this section by explaining why this solution was chosen.

3.1. Multiple Source Ports

One approach is to follow the suggestion of [RFC2865] Section 5, which says:

> ... If a NAS needs more than 256 IDs for outstanding requests, it MAY use additional source ports to send requests from, and keep track of IDs for each source port. This allows up to 16 million or so outstanding requests at one time to a single server

This suggestion has been widely implemented. However, practice shows that the number of open ports has a practical limitation much lower than 65535. This limitation is due both to ports being used by other applications on the same system, and application and OS-specific complexities related to opening thousands of ports.

The "multiple source port" approach is workable for low-load systems. However, implementors of high-load systems have requested an alternative to that approach. The justification is that this approach has proven to be problematic in practice for them.

3.2. Diameter

Another common approach is to suggest that implementors switch to using Diameter instead of RADIUS. We believe that the Diameter

protocol does not satisfy the requirements of the implementors who are requesting extensions to RADIUS.

The short summary is that implementors have significant investment in a RADIUS infrastructure. Switching to Diameter would involve either deprecating or throwing away all of that infrastructure. This approach is simply not technically or economically feasible.

Ad hoc surveys indicate that the majority of the implementations that will use this specification do not have pre-existing Diameter code bases. We suggest that it is simpler for implementors to make minor changes to their RADIUS systems than it is to implement a full Diameter stack. This reality is a major consideration when creating new specifications.

For these reasons, switching to Diameter is not a useful approach.

3.3. Multiple RADIUS packets in UDP

Another approach would be to allow multiple RADIUS packets in one UDP packet. This method would allow an increased amount of RADIUS traffic to be sent in the same number of UDP packets.

However, this approach still has the limit of 256 outstanding requests, which means that implementations have extra work of juggling RADIUS packets, UDP packets, and UDP connections. In addition, this approach does not address the issues discussed above for RADIUS over TCP.

As a result, this approach is not suitable as a solution to the ID limit problem.

3.4. An Extended ID field

Another approach it to use an attribute which contained an "extended" ID, typically one which is 32 bits in size. That approach has been used in at least one commercial RADIUS server for years, via a Vendor-Specific Attribute.

The benefits of this approach is that it makes no change to the RADIUS protocol format, attribute encoding, data types, security, etc. It just adds one "integer" attribute, as an "extended ID" field. Client implementations add the "extended ID" attribute to requests, and server implementations echo it back in responses. The "extended ID" field is also used in doing duplicate detection, finding conflicting packets, etc.

Implementations using this approach have generally chosen to not perform negotiation of this functionality. Instead, they require

both client and server to be statically configured to enable the "extended ID".

Clients implementing the "extended ID" must, of course, manage this new identifier. As the identfier is still local to a connection, it is possible to simply take the "extended ID" from a counter which is incremented for every request packet. There is no need to mark these identifiers as unused, as the 32-bit counter space is enough to ensure that re-use happens rarely. i.e. at 1 million packets per second, the counter is enough to last for over an hour, which is a time frame much larger than the lifetime of any individual packet.

This approach is compatible with all RADIUS transports, which is a major point in its favor.

An implementation of this "extended ID" approach is less than 200 lines of C code [PATCH]. That patch includes capability negotiation via Status-Server, and full client / server / proxy support.

This approach has therefore been proven to be workable in practice, and simple to implement.

3.5. This Specification

The approach discussed here was chosen after looking at the handling of packet conflicts, as discussed above in Section X. The conclusion was that since the behavior around "conflicting packets" was entirely implementation-defined, then changing the behavior would involve minor changes to the RADIUS specifications.

This specification suggests that clients and servers can choose to use the Request Authenticator as a an additional field to uniquely identify request packets. This choice is entirely local to the client or server implementation, and involves no changes to the wire format or wire protocol. There are additional considerations which are outlined below.

The approach outlined in this specification is largely similar to the "extended ID" approach, except that it leverages a pre-existing field as an identifier, instead of creating a new one. This re-use means that the client does not need to track or allocate new identifiers.

The use of the Request Authenticator as an identifier means that there are 128 bits of space available for identifiers. This large space means that the "conflicting packet" problem is avoided almost entirely, as the Request Authenticator is either random data (Access-Request), or an MD5 signature (other packets).

The subject of MD5 collisions is addressed below, in Section X.

As with the "extended ID" approach. this approach is compatible with all transports.

We believe that this approach is slightly simpler than the next best approach ("extended ID"), while at the same time providing more benefits. As a result, it is the recommended approach for allowing more than 256 outstanding packets on one connection.

4. Protocol Overview

We extend RADIUS to allow the use of the Request Authenticator field as an additional identifier. Subject to certain caveats outlined below, the Request Authenticator can be treated as being temporally and globally unique. This uniqueness is what makes it a useful identifier field.

The capability is first negotiated via Status-Server, as outlined below. Once both client and server agree on the use of this capability, the client can start sending normal request packets.

The client creates a request packet as usual. When the client needs to store the request packet, it adds the Request Authenticator as part of the unique key which tracks the request. The packet is then sent to the server.

The server receives the request, and uses the Request Authenticator to track the request, in addition to any previously used information. When the server sends a reply, it copies the Request Authenticator to the response packet, and places the 16 octet value into the Original-Request-Authenticator attribute. The response is then sent as before.

When the client receives the response, it uses the Original-Request-Authenticator attribute to create the lookup key, in order to find the original request. Once the original request is found, packet verification and processing continues as before.

We note again that the Identifier field is still used to correlate requests and responses, along with any other information that the implementation deems necessary. (e.g. Code, socket descriptor, src/dst IP/port, etc.) The only change is that the Request Authenticator is added to that tuple.

In short, the "ID tracking" method of tracking requests and responses is extended to allow the use of the Request Authenticator as part of the tuple used to track requests and responses. This new tracking method is called "ORA tracking".

Further details and implementation considerations are provided below.

for clients: If ORA has not been negotiated, assign IDs based on old-style tree. IF ORA has been negotiated, assign IDs based on ORA tree, generally just a sequential counter.

IF the response does not contain ORA, lookup packets based on old-style tree. IF the response does contain ORA, lookup packets based on new-style tree. If it's not found there, look it up in the old-stlye tree. This behavior ensures that clients can send both kinds of packets simultaneously with no conflict.

There is no need for any additional tracking / timer on the client... if the old-style tree is empty, there is essentially no cost to lookup packets up in it. The alterntive would be to track which packets were sent before / after negotiation, and do all kinds of additional magic.

The server has then to figure out WTF is going on... since packets aren't ordered, there's no way to tell which ones were sent *before* negotiation, and which ones *after*. In contrast, the extended-ID is easier, as the request contains the attribute. The solution is for the server to always send back ORA after negotiation has succeeded, even if the packet was received prior to negotiation happening.

Since the Request Authenticator field is sixteen octets in size, this process allows an essentially unlimited number of requests to be outstanding on any one connection. This capability allows clients to open only one connection to a server, and to send all data over that connection. As noted above, using fewer connections will increase the clients ability to perform dead server detection, do fail-over, and will result in increased network stability.

4.1. Why this works

In this section, we explain why the Request Authenticator makes a good packet identifier.

For Access-Request packets, [RFC2865] Section 3 defines the Request Authenticator to contain random values. Further, it is suggested that these values "SHOULD exhibit global and temporal uniqueness". The same definition is used in [RFC5997] Section 3, for Status-Server packets. Experience shows that implementations follow this suggestion. As a result, the Request Authenticator is a good identifier which uniquely identifies packets.

Other request packets create the Request Authenticator as an MD5 calculation over the packet and shared secret. i.e. MD5(packet + secret). The MD5 digest algorithm was designed to be strongly dependent on input data, and to have half of the output bits change if one bit changed in the input. As a result, the Request

Authenticator is a good hash which can distinguish different packets.

The question is whether or not the Request Authenticator is a good identifier. The following discussion make this case.

One argument is that MD5 has low collision rates. In the absence of an explicit attack, there should be one collision every 2e64 packets. Since the packet lifetime is small (typically 30 seconds maximum), we can expect a collision only if more than 2e59 packets are sent during that time frame. For more typical use-cases, the packet rate is low enough (i.e. even 2e20 packets per second), that there is a one in 2e39 chance of a collision every 30 seconds.

We believe that such collision rates are acceptibly low. Explicit attacks are discussed in the Security Considerations section, below.

The one case where collisions will occur naturally is when the packet contents are identical. For example, a transmission of a second Access-Request after the first one times out. In this situation, though, there is in fact no collision, as the input data is identical. Both requests are entirely identical, and any response to one is a response to both.

For non-identical requests, the packet typically contains the Identifier and Length fields, along with counters, timestamps, etc. These values change on a per-packet basis, making the Request Authenticator also change.

As a result, the MD5 signature of a request is appropriate to use as a packet identifier. In all cases (barring attacks), it will contain a globally and temporally unique identifier for the request.

5. Signaling via Status-Server

When a client supports this functionality, it sends a Status-Server packet to the server, containing an Original-Request-Authenticator attribute. See Section X, below, for the definition of the Original-Request-Authenticator attribute.

The contents of the Original-Request-Authenticator attribute in the Status-Server packet MUST be zero. The Original-Request-Authenticator attribute MUST NOT appear in any request packet other than Status-Server. If a server does see this attribute in a request packet, the attribute MUST be treated as an "invalid attribute", and ignored as per [RFC6929] Section 2.8.

A server which supports this functionality SHOULD signal that capability to the client by sending a response packet which contains an Original-Request-Authenticator attribute. That attribute MUST

contain an identical copy of the Request Authenticator from the original Status-Server packet.

When a client sends an Original-Request-Authenticator attribute in a Status-Server and does not receive that attribute in the response, the client MUST NOT use "ORA tracking" for requests and responses. The client MUST then behave as a normal RADIUS client, and use "ID tracking" for requests and response.

If a server does not support this functionality, it MUST NOT place an Original-Request-Authenticator attribute in the response packet. As the default behavior of existing RADIUS servers is to not place this attribute in the response to a Status-Server, negotiation will continue to use traditional RADIUS, and "ID tracking".

As the response to a Status-Server can use one of many RADIUS Codes, we use a generic "response" name above. See following sections for how to handle specific types of responses.

We note that "ORA tracking" negotation SHOULD be done per connection. i.e. per combination of (transport, src/dst ip/port). Section X, below, discusses additional issues related to client/server connections. In this section, when we refer to a client and server performing negotiation, we mean that negotiation to be specific to a particular connection.

Once the "ORA tracking" has been negotiated on a connect, then all packets for that connection MUST use it, no matter what values they allow for the Code field. For example, [RFC6613] permits multiple Codes on one connection.

Even if a client and server negotiate "ORA tracking", the client can still fall back to "ID tracking". There is no need for the client to signal the server that this change has happened. The client can use "ID tracking" while the server uses "ORA tracking", as the two systems are entirely compatible from the client side.

The situation is a bit different for a server. Once "ORA tracking" has been negotiated, a server MUST use that method, and MUST include the Original-Request-Authenticator attribute in all response packets. If a client negotiates "ORA tracking" on a connection and later sees response packets which do not contain an Original-Request-Authenticator attribute, the client SHOULD discard those non-compliant packets. For connection-oriented protocols, the client SHOULD close the connection.

There is a time frame during this failure process during which outstanding requests on that connection may not receive responses. This situation will result in packet loss, which will be corrected once the new connection is used. The possibility of such problems

should be used instead by implementors as incentive to ensure that they do not create invalid Original-Request-Authenticator attributes. Implementing the specification correctly will prevent this packet loss from occuring.

The negotiation outlined here ensures that RADIUS clients and servers supporting this functionality are entirely backwards compatible with existing RADIUS clients and servers.

5.1. Static Configuration

As an alternative to using Status-Server, clients and servers MAY be administratively configured with a flag which indicates that the other party supports this functionality. Such a flag can be used where the parties are known to each other. Such a flag is not appropriate for dynamic peer discovery [RFC7585], as there are no provisions for encoding the flag in the DNS queries or responses.

When a client is administratively configured to know that a server supports this functionality, it SHOULD NOT do negotiation via Status-Server.

The client MUST behave as a normal RADIUS client (i.e. send only 256 requests on any one connection) until such time as it receives an Original-Request-Authenticator attribute in a response. Only then can the client send more packets on one connection. See Section X ("Connection Issues") below, for a larger discussion of this topic.

If a client is administratively configured to believe that a server supports the Original-Request-Authenticator attribute, but the response packets do not contain an Original-Request-Authenticator attribute, the client MUST update its configuration to mark the server as not supporting this functionality.

This process allows for relatively simple downgrade negotiation in the event of misconfiguration on either the client or the server.

6. Original-Request-Authenticator Attribute

We define a new attribute, called Original-Request-Authenticator. It is intended to be used in response packets, where it contains an exact copy of the Request Authenticator field from the original request that elicited the response.

As per the suggestions of [RFC8044], we describe the attribute using a data type defined therein, and without the use of ASCII art.

Type

TBD - IANA allocation from the "extended" type space

Length

   19 - TBD double-check this after IANA allocation

Data Type

   octets

Value

   MUST be 16 octets in length. For Status-Server packets, the
   contents of the Value field MUST be zero. For response packets,
   the contents of the Value field MUST be a copy of the Request
   Authenticator from the original packet that elicited the
   response.

The Original-Request-Authenticator attribute can be used in a Status-Server packet.

The Original-Request-Authenticator attribute can be used in a response packet. For example, it can be used in an Access-Accept, Accounting-Response, CoA-ACK, CoA-NAK, etc.

Note that this document updates multiple previous specifications, in order to allow this attribute in responses.

  *[RFC2865] Section 5.44 is updated to allow Original-Request-Authenticator in Access-Accept, Access-Reject, and Access-Challenge responses

  *[RFC2866] Section 5.13 is updated to allow Original-Request-Authenticator in Accounting-Response responses.

  *[RFC5176] Section 3.6 is updated to allow Original-Request-Authenticator in CoA-ACK, CoA-NAK, Disconnect-Ack, and Disconnect-NAK responses.

The Original-Request-Authenticator attribute MUST NOT be used in any request packet. That is, it MUST NOT be used in an Access-Request, Accounting-Request, CoA-Request, or Disconnect-Request packets.

When it is permitted in a packet, the Original-Request-Authenticator attribute MUST exist either zero or one times in that packet. There MUST NOT be multiple occurances of the attribute in a packet.

The contents of the Original-Request-Authenticator attribute MUST be an exact copy of the Request Authenticator field of a request packet sent by a client. As with "ID tracking", the Identifier field in the response MUST match the Identifier field in a request.

Where the format and/or contents of the Original-Request-Authenticator attribute does not meet these criteria, the received attribute MUST be treated as an "invalid attribute" as per [RFC6929], Section 2.8. That is, when an invalid Original-Request-Authenticator attribute is seen by either a client or server, their behavior is to behave as if the attribute did not exist.

7. Transport Considerations

This section describes transport considerations for this specification.

The considerations for DTLS are largely the same as for UDP. The considerations for TLS are largely the same as for TCP. We therefore do not have different sections herein for the TLS-enabled portion of the protocols.

7.1. UDP

RADIUS over UDP is defined in [RFC2866]. RADIUS over DTLS is defined in [RFC7360].

When negotiated by both peers, this proposal changes the number of requests which can be outstanding over a UDP connection.

Where clients are sending RADIUS packets over UDP, they SHOULD include the Original-Request-Authenticator attribute in all Status-Server messages to a server, even if the functionality has been previously negotiatied. While the client can generally assume that a continual flow of packets means that the server has not been changed, this assumption is not true when the server is unresponsive, and the client decides it needs to send Status-Server packets.

Similarly, the server cannot assume that it is respond to the same client on every packet. However, once Original-Request-Authenticator has been negotiasted, the server can safely include that attribute in all response packets to that client. If the client changes to not supporting the attribute, the attribute will be ignored by the client, and the behavior falls back to standard RADIUS.

Where clients are sending RADIUS packets over DTLS, there is an underlying TLS session context. The client can therefore assume that all packets for one TLS session are for the same server, with the same capabilities. The server can make the same assumption.

7.2. TCP

RADIUS over TCP is defined in [RFC6614]. RADIUS over TLS is defined in [RFC6614].

When negotiated by both peers, this proposal changes the number of requests which can be outstanding over a TCP connection.

Status-Server packets are also used by the application-layer watchdog, described in [RFC6614] Section 2.6. Where clients have previously negotiated Original-Request-Authenticator for a connection, they MUST continue to send that attribute in all Status-Server packets over that connection.

There are other consideratiosn with the use of Status-Server. Due to the limitations of the ID field, [RFC6613] Section 2.6.5 suggests:

.in +0.3i Implementations SHOULD reserve ID zero (0) on each TCP connection for Status-Server packets. This value was picked arbitrarily, as there is no reason to choose any one value over another for this use. .in -0.3i

This restriction can now be relaxed when both client and server have negotiated the use of the Original-Request-Authenticator attribute. Or, with no loss of generality, implementations can continue to use a fixed ID field for Status-Server application watchdog messages.

We also note that the next paragraph of [RFC6614] Section 2.6.5. says:

> Implementors may be tempted to extend RADIUS to permit more than 256 outstanding packets on one connection. However, doing so is a violation of a fundamental part of the protocol and MUST NOT be done. Making that extension here is outside of the scope of this specification.

This specification extends RADIUS in a standard way, making that recommendation no longer applicable.

[RFC6613] Section 2.5 describes congestion control issues which affect inter-transport proxies. If both inbound and outbound transports support this specification, those congestion issues no longer apply.

If however, a proxy supports this specification on inbound connections but does not support it on outbound connections, then congestion may occur. The only solution here is to ensure that the proxy is capable of opening multiple source ports, as per [RFC2865] Section 5.

7.3. Dynamic Discovery

The dynamic discovery of RADIUS servers is defined in [RFC7585].

This specification is compatible with [RFC7585], with the exception of the statically configured flag described in Section X, above. As the server is dynamically discovered, it is impossible to have a static flag describing the server capabilities.

The other considerations for dynamic discovery are the same as for RADIUS over TLS.

7.4. Connection Issues

Where clients start a new connection to a server (no matter what the transport), they SHOULD negotiate this functionality for the new connection, unless the ability has been statically configured. There is no guarantee that the new connection goes to the same server.

When a client has zero connections to a server, it MUST perform this negotiation for the new connection, prior to using this functionality, unless the ability has been statically configured. There is every reason to believe that server has remained the same over extended periods of time.

If a client has one or more connections open to a server, and wishes to open a new one, it may skip the renegotiation.

Each client and server MUST negotiate and track this capability on a per-connection basis. Implementations MUST be able to send packets to the same peer at the same time, using both this method, and the traditional RADIUS ID allocation.

A client may have a backlog of packets to send while negotiating this functionality. In the interests of efficiency, it SHOULD send packets from that backlog while negotiation is taking place. As negotiation has not finished, these packets and their responses MUST be managed as per standard RADIUS.

After this functionality has been negotiated, new packets from that connection MUST follow this specification. Reponses to earlier packets sent on that connection during the negotiation phase MUST be accepted and processed.

In short, the client MUST behave as a normal RADIUS client, until such time as it receives a response packet which contains a compliant Original-Request-Authenticator attribute. This requirement ensures complete compatibility with existing RADIUS, even in the event of client misconfiguration.

We recognize that this tracking may be complex, which is why this behavior is not mandatory. Clients may choose instead to wait until negotiation is complete before sending packets; or to assume that

the functionality of the server is the same across all connections to it, and therefore only do negotiations once.

8. System Considerations

This section describes implementation considerations for clients and servers.

8.1. Client Considerations

Clients SHOULD have an configuration flag which lets administators statically configure this behavior for a server. Clients MUST otherwise negotiate this functionality before using it.

If this functionality has been negotiated, clients MUST use the Request Authenticator as an part of the Key used to uniquely identify request packets. Clients MUST use the Original-Request-Authenticator attribute from response packets as part of the Key to find the original request packet.

The Original-Request-Authenticator attribute has been (or is likely to be) allocated from the "extended" attribute space. We note that despite this allocation, clients are not required to implement the full [RFC6929] specification. That is, clients may be able to originate and receive Original-Request-Authenticator attributes, while still being unable to originate or receive any other attribute in the "extended" attribute space.

The traditional behavior of clients is to track one or more connections, each of which has 256 IDs available for use. As requests are sent, IDs are marked "used". As responses are received, IDs are marked "free". IDs may also marked "free" when a request times out, and the client gives up on receiving a response.

If all of the IDs for a particular connection are marked "free", the client opens a new connection, as per the suggestion of [RFC2865] Section 5. This connection and any associated IDs are then made available for use by new requests.

Similarly, when a client notices that all of the IDs for a connection are marked "free", it may close that connection, and remove the IDs from the ones available for use by new requests. The connections may have associated idle timeouts, maximum lifetimes, etc. to avoid "connection flapping".

All of this management is complex, and can be expensive for client implementations. While this management is still necessary for backwards compatibility, this specification allows for a significantly simpler process for ID allocation. There is no need

for the client to open multiple connections. Instead, all traffic can be sent over one connection.

In addition, there is no need to track "used" or "free" status for individual IDs. Instead, the client can re-use IDs at will, and can rely on the uniqueness of the Request Authenticator to disambiguate packets.

As there is no need to track ID status, the client may simply allocate IDs by incrementing a local counter.

With this specification, the client still needs to track all outgoing requests, but that work was already required in traditional RADIUS.

Client implementors may be tempted to require that the Original-Request-Authenticator be the first attribute after the RADIUS header. We state instead that clients implementing this specification MUST accept the Original-Request-Authenticator attribute, no matter where it is in the response. We remind implementors that this specification adds a new attribute, it does not change the RADIUS header.

Finally, we note that clients MUST NOT set the ID field to a fixed value for all packets. While it is beneficial to use the Request Authenticator as an identifier, removing the utility of an existing identifier is unwarranted.

8.2. Server Considerations

Servers SHOULD have an configuration flag which lets administators statically configure this behavior for a client. Servers MUST otherwise negotiate this functionality before using it.

If this functionality has been negotiated, servers MUST use the Request Authenticator as an part of the key used to uniquely identify request packets. Servers MUST use the Original-Request-Authenticator attribute from response packets as part of the Key to find the original request packet.

The Original-Request-Authenticator attribute has been (or is likely to be) allocated from the "extended" attribute space. We note that despite this allocation, servers are not required to implement the full [RFC6929] specification. That is, servers may be able to originate and receive Original-Request-Authenticator attributes, while still being unable to originate or receive any other attribute in the "extended" attribute space.

8.3. Proxy Considerations

There are additional considerations specific to proxies. [RFC6929] Section 5.2 says in part;

> Proxy servers SHOULD forward attributes, even attributes that they do not understand or that are not in a local dictionary. When forwarded, these attributes SHOULD be sent verbatim, with no modifications or changes. This requirement includes "invalid attributes", as there may be some other system in the network that understands them.

On its face, this recommendation applies to the Original-Request-Authenticator attribute. The caveast is that Section X, above, requires that servers do not send the Original-Request-Authenticator to clients unless the clients have first negotiated the use of that attribute. This requirement should ensure that proxies which are unaware of the Original-Request-Authenticator attribute will never receive it.

However, if a server has been administratively configured to send Original-Request-Authenticator to a client, that configuration may be in error. In which case a proxy or originating client may erroneously receive that attribute. If the proxy or server is unaware of Original-Request-Authenticator, then no harm is done.

It is possible for a proxy or client to be aware of Original-Request-Authenticator, and not negotiate it with a server, but that server (due to isses outlined above) still forwards the attribute to the proxy or client. In that case, the requirements of Section X, above, are that the client treat the received Original-Request-Authenticator attribure as an "invalid attribute", and ignore it.

The net effect of these requirements and cross-checks is that there are no interoperability issues between existing RADIUS implementations, and implementations of this specification.

9. Security Considerations

This proposal does not change the underlying RADIUS security model, which is poor.

The contents of the Original-Request-Authenticator attribute are the Request Authenticator, which is already public information for UDP or TCP transports.

The use of Original-Request-Authenticator is defined in such a way that all systems fall back gracefully to using standard RADIUS. As such, there are no interoperability issues between this specification and existing RADIUS implementations.

There are few, if any, security considerations related to implementations. Clients already must track the Request Authenticator, so matching it in a response packet is minimal extra work. Servers must also track and cache duplicate packets, as per [RFC5080] Section 2.2.2, so using the Request Authenticator as an additional identifier is minimal extra work.

The use (or not) of Original-Request-Authenticator has no other security considerations, as it is used solely as an identifier to match requests and responses. It has no other meaning or use.

9.1. Access-Request Forging

The Request Authenticator in Access-Request packets is defined to be a 16 octet random number [RFC2865] Section 3. As such, these packets can be trivially forged.

The Message-Authenticator attribute was defined in [RFC2869] Section 5.14 in order to address this issue. Further, [RFC5080] Section 2.2.2 suggests that client implementations SHOULD include a Message-Authenticator attribute in every Access-Request to further help mitigate this issue.

The Status-Server packets also have a Request Authenticator which is a 16-octet random number [RFC5997] Section 3. However, [RFC5997] Section Section 2 says that a Message-Authenticator attribute MUST be included in every Status-Server packet, which provides per-packet authentication and integrity protection.

We extend that suggestion for this specification. Where the transport does not provide for authentication or integrity protection (e.g. RADIUS over UDP or RADIUS over TCP), each Access-Request packet using this specification MUST include a Message-Authenticator attribute. This inclusion ensures that packets are accepted only from clients who know the RADIUS shared secret.

This protection is, of course, insufficient. Malicious or misbehaving clients can create Access-Request packets which re-use Request Authenticators. These clients can also create Request Authenticators which exploit implementation issues in servers, such as turning a simply binary lookup into a linked list lookup.

As a result, server implementations MUST NOT assume that the Request Authenticator is random. Server implementations MUST be able to detect re-use of Request Authenticators.

When a server detects that a Request Authenticator is re-used, it MUST replace the older request with the newer request. It MUST NOT respond to the older request. It SHOULD issue a warning message to the administrator that the client is malicious or misbehaving.

Server implementations SHOULD use data structures such as Red-Black trees, which are immune to maliciously crafted Request Authenticators.

9.2. MD5 Collisions

For other packet types (Accounting-Request, etc.), the Request Authenticator is the MD5 signature of the packet and the shared secret. Since this data is used directly as an identifier, we need to examine the security issues related to this practice.

We must note that MD5 has been broken, in that there is a published set of work which describes how to create two sets of input data which have the same MD5 hash. These attacks have been extended to create sets of data of arbitrary length, which differ only in 128 bytes, and have the same MD5 hash.

This attack is possible in RADIUS, as the protocol has the capability to transport opaque binary data in (for example) Vendor-Specific attributes. There is no need for the client or server to understand the data, it simply has to exist in the packet for the attack to succeed.

Another attack allows two sets of data to have the same MD5 hash, by appending thousands of bytes of carefully crafted data to the end of the file. This attack is also possible in RADIUS, as the maximum packet size for UDP is 4096 octets, and [RFC7930] permits packets up to 65535 octets in length.

However, as the packets are authenticated with the shared secret, these attacks can only be performed by clients who are in possession of the shared secret. That is, only trusted clients can create MD5 collisions.

We note that this specification requires server implementations to detect duplicates, and to process only one of the packets. This requirement could be exploited by a client to force a server to do large amounts of work, partially processing a packet which is then made obselete by a subsequent packet. This attack can be done in RADIUS today, so this specification adds no new security issues to the protocol.

In fact, this specification describes the problem of "conflicting packets" for the first time, and defines how they should be processed by servers. This addition to the RADIUS protocol in fact increases it's security, by specifying how this corner case should be handled. The fact that RADIUS has been widely implemented for almost 25 years without this issue being described shows that the protocol and implementations are robust.

We do not offer a technical solution to the problem of trusted parties misbehaving. Instead, the problem should be noted by the server which is being attacked, and administrative (i.e. human) intervention should take place.

10. IANA Considerations

This specification allocates one attribute in the RADIUS Attribute Type registry, as follows.

Name > Original-Request-Authenticator

Type > TBD - allocate from the "extended" space

Data Type > octets

11. Acknowledgements

In hindsight, the decision to retain MD5 for RADIUS/TLS was likely wrong. It was an easy decision in the short term, but it has caused ongoing problems which this document addresses.

12. Changelog

13. References

13.1. Normative References

[BCP14] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <https://www.rfc-editor.org/info/rfc2865>.

[RFC2869] Rigney, C., Willats, W., and P. Calhoun, "RADIUS Extensions", RFC 2869, DOI 10.17487/RFC2869, June 2000, <https://www.rfc-editor.org/info/rfc2869>.

[RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, DOI 10.17487/RFC5080, December 2007, <https://www.rfc-editor.org/info/rfc5080>.

[RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, DOI 10.17487/RFC6929, April 2013, <https://www.rfc-editor.org/info/rfc6929>.

[RFC8044] DeKok, A., "Data Types in RADIUS", RFC 8044, DOI 10.17487/RFC8044, January 2017, <https://www.rfc-editor.org/info/rfc8044>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[PATCH] Naiming, S., "https://mailarchive.ietf.org/arch/msg/radext/BkXgD68MASxqD4vjXV1M2CaRWAY", April 2017.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <https://www.rfc-editor.org/info/rfc2866>.

[RFC5176] Chiba, M., Dommety, G., Eklund, M., Mitton, D., and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, DOI 10.17487/RFC5176, January 2008, <https://www.rfc-editor.org/info/rfc5176>.

[RFC5997] DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", RFC 5997, DOI 10.17487/RFC5997, August 2010, <https://www.rfc-editor.org/info/rfc5997>.

[RFC6613] DeKok, A., "RADIUS over TCP", RFC 6613, DOI 10.17487/RFC6613, May 2012, <https://www.rfc-editor.org/info/rfc6613>.

[RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, DOI 10.17487/RFC6614, May 2012, <https://www.rfc-editor.org/info/rfc6614>.

[RFC7360] DeKok, A., "Datagram Transport Layer Security (DTLS) as a Transport Layer for RADIUS", RFC 7360, DOI 10.17487/RFC7360, September 2014, <https://www.rfc-editor.org/info/rfc7360>.

[RFC7585] Winter, S. and M. McCauley, "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access

Identifier (NAI)", RFC 7585, DOI 10.17487/RFC7585, October 2015, <https://www.rfc-editor.org/info/rfc7585>.

[RFC7930] Hartman, S., "Larger Packets for RADIUS over TCP", RFC 7930, DOI 10.17487/RFC7930, August 2016, <https://www.rfc-editor.org/info/rfc7930>.

Author's Address

Alan DeKok
FreeRADIUS

Email: aland@freeradius.org