

Network Working Group
Internet Draft
Intended status: Experimental
Expires: November 2013

P. Deacon
IEA Software, Inc
May 1, 2013

RADIUS Extended Request
draft-deacon-radext-extended-request-01.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on August 1, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document describes methods for RADIUS servers to communicate optional extended abilities to RADIUS clients. The abilities

described provide for exchange of RADIUS packets where total packet length exceeds 4096 bytes.

Table of Contents

1. Introduction.....	3
1.1. Terminology.....	3
2. Conventions used in this document.....	4
3. Extended-Request.....	4
3.1. Packet Format.....	5
4. Extended Request head attributes.....	7
4.1. Fragment-Data.....	7
4.1.1. Client to server packet fragmentation.....	7
4.1.1.1. Prerequisites.....	7
4.1.1.2. Preparing inner request packet.....	7
4.1.1.3. Generating outer request packet.....	8
4.1.1.4. Transmitting fragments to RADIUS server.....	9
4.1.1.5. Server state and fragment validation.....	9
4.1.2. Server to client packet fragmentation.....	10
4.1.2.1. Prerequisites.....	10
4.1.2.2. Preparing inner response packet.....	10
4.1.2.3. Generating outer response packet.....	11
4.1.2.4. Transmitting fragments to RADIUS client.....	12
4.1.2.5. Client state and fragment validation.....	12
4.1.3. Inner packet reassembly.....	13
4.1.4. Request response options.....	13
4.1.4.1. Fragmented response to non-fragmented request..	13
4.1.4.2. Fragmented response to fragmented request.....	14
4.1.5. Fragment-Data head attribute format.....	14
4.1.6. Server state management.....	15
4.1.7. Per-fragment and inner packet sizing.....	15
4.2. Fragment-Inquire.....	16
4.2.1. Request.....	16
4.2.2. Response.....	17
5. Compatibility.....	17
6. Attributes.....	18
6.1. Fragment-Reply-Supported.....	18
6.2. Fragment-Reply-Allowed.....	18
6.3. Fragment-Stream-Limit.....	19
6.4. Fragment-Limit.....	20
6.5. Fragment-Inquire-Interval.....	21
6.6. Framed-MTU.....	22
6.7. Event-Timestamp.....	22
7. Table of Attributes.....	22
8. Security Considerations.....	23
9. IANA Considerations.....	23

10. References.....	23
10.1. Normative References.....	23
11. Acknowledgments.....	24

1. Introduction

Historically RADIUS packets transporting Authentication Authorization and Accounting (AAA) information required a small fraction of 4096 byte message limit allotted by [RFC2865]. Today need for larger packets driven by progressively complex security and configuration requirements has increased pressure for RADIUS beyond 4096 bytes.

This text describes methods for enabling RADIUS clients and servers to effectively exchange large RADIUS packets above limits prescribed by [RFC2865].

To maintain compatibility with existing RADIUS infrastructure a protocol is defined such that large packets shall be permitted by RADIUS server or client only after support for large packets has been established. This is achieved automatically using the protocol defined in this text or by non-default administrative settings.

Two methods for supporting RADIUS packets beyond 4096 bytes are described.

Switching to TCP - Clients normally using UDP may elect to use TCP [RFC6614] always or only while large packets shall be exchanged. Since RADIUS over TCP is also limited to 4096 bytes procedures are described for establishing availability of TCP to UDP clients as well as TCP support for large packets to UDP and TCP clients. TCP is the recommended method for transmitting large RADIUS packets.

Fragmentation - Intended for UDP interoperability with TCP. This approach is based on fragmenting large packets into a series of smaller RADIUS packets, transmitting and finally reassembling all packet fragments. Procedures to signal support for fragmentation to client and server are described.

1.1. Terminology

This document uses these terms:

Head Attribute Attribute immediately following header of an Extended-Request packet.

Outer Packet	When a packet encapsulates another packet the encapsulating packet is known as the outer packet.
Inner Packet	When a packet encapsulates another packet the encapsulated packet is known as the inner packet.
RADIUS Client	For purposes of this document a RADIUS client is that which initiates a RADIUS request packet.
RADIUS Server	For purposes of this document a RADIUS server is that which responds to a RADIUS request packet.
Attribute	Usage of the word attribute in this document does not include "long attributes" or similar concepts where a logical attribute is created by aggregation of multiple Type-Length-Value fields.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Extended-Request

An Extended-Request packet is sent to the RADIUS server requesting an action whose purpose is determined by an attribute present immediately after RADIUS header within Extended-Request packet. This attribute is known as the "head attribute". All subsequent attributes are evaluated exclusively within defined context of the head attribute. Subsequent attributes have no meaning or purpose outside of those explicitly defined within the head attributes specification. Section 4 provides descriptions of each head attribute defined by this text.

In response to an Extended-Request packet sent to RADIUS server an Extended-Response or Extended-Reject packet is returned to the client indicating result of Extended-Request.

Extended-Response packets may include one or more response attributes. Extended-Reject packets indicate failure. Extended-Reject packets include failure attributes to communicate diagnostic information to the RADIUS client.

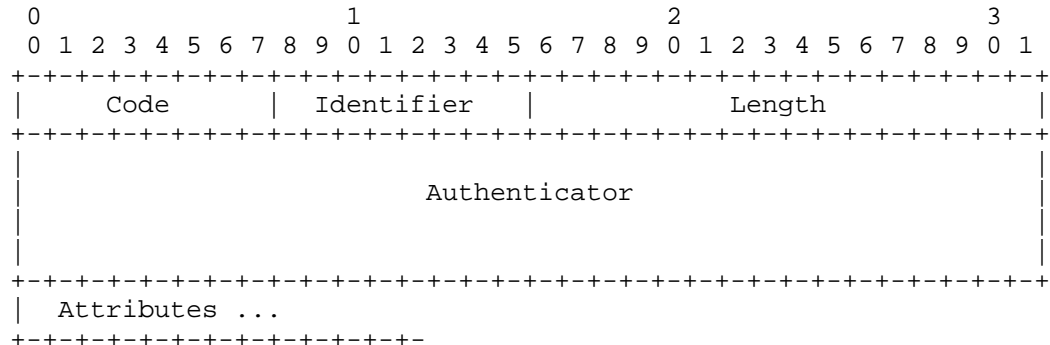
A RADIUS server not supporting a head attribute responds with Extended-Reject containing attribute Error-Cause = "Unsupported Extension" [RFC5176]. RADIUS servers MUST NOT make any attempt to use or interpret attributes subsequent to the head attribute in the event head attribute is unknown or not supported. All such attributes MUST be ignored.

Should RADIUS server receive an Extended-Request packet with no attributes or receive a request containing Missing attributes per the specification of a supported head attribute an Extended-Reject message is returned containing attribute Error-Cause = "Missing Attribute"

3.1. Packet Format

Packet format consists of the fields: Code, Identifier, Length, Authenticator and optional Attributes. All fields hold the same meaning as those described in RADIUS [RFC2865].

Authenticator field is calculated using same method specified for Accounting-Request and Accounting-Response packets [RFC2866].



Code

The Code field is one octet, and identifies the type of RADIUS packet. RADIUS codes described in this document are assigned as follows:

TBD - Extended-Request
TBD - Extended-Response
TBD - Extended-Reject

Identifier

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.

Length

The Length field is two octets. It indicates the length of the packet including the Code, Identifier, Length, Authenticator and Attribute fields. Octets outside the range of the Length field MUST be treated as padding and ignored on reception. If the packet is shorter than the Length field indicates, it MUST be silently discarded. The minimum length is 20 and maximum length is 4096 when transmitted over UDP.

Authenticator

The Authenticator field is sixteen (16) octets. This value is used to authenticate packets between the RADIUS server and client.

Request Authenticator

The Authenticator field in a Request packet (e.g. Extended-Request) is called the Request Authenticator. The Request Authenticator is calculated the same way as for an Accounting-Request packet specified in [RFC2866].

Response Authenticator

The Authenticator field in a Response packet (e.g. Extended-Response or Extended-Reject) is called the Response Authenticator. This field is calculated the same way as for an Accounting-Response packet specified in [RFC2866].

Attributes

Attributes may have none or multiple instances.

4. Extended Request head attributes

Head attributes always appear immediately after RADIUS header within an Extended-Request and conditionally within Extended-Response packets. Each head attribute defines purpose and usage of Extended-Request and Extended-Response packets.

4.1. Fragment-Data

Fragment-Data head attribute describes a method for encapsulating a large RADIUS packet within series of smaller Extended-Request packets and later reassembling the encapsulated packet.

In this section "outer packet" refers to the Extended-Request or Extended-Response RADIUS packet acting as an envelope for the encapsulated RADIUS packet. "Inner packet" refers to the encapsulated packet.

4.1.1. Client to server packet fragmentation

This section describes method for RADIUS client to fragment and transmit request packets (e.g. Access-Request) to server.

4.1.1.1. Prerequisites

RADIUS client SHALL meet one or more of the following requirements before sending a fragmented request:

1. Administrative setting indicating RADIUS server support for fragments. If client has previously received successful Fragment-Inquire response either omitting Fragment-Limit or consisting of value ≤ 4096 bytes the administrative setting is ignored, a fragmented request MUST NOT be sent.
2. Fragment-Inquire response containing Fragment-Limit having value > 4096 .

4.1.1.2. Preparing inner request packet

RADIUS client generates a complete RADIUS request packet in a storage buffer rather than transmitting to RADIUS server. Prior to generating request packet following changes to normal processing SHALL be observed:

1. Identifier field of RADIUS request is set 0. This field is unused while inner packet is encapsulated in outer packet.

2. 4096 byte maximum packet length [RFC2865] limit is replaced with lower of following three constraints:
 - A. Maximum Length RADIUS header field can accommodate. (e.g. 65535 bytes)
 - B. Value of Fragment-Limit obtained by prior Fragment-Inquire request.
 - C. Administrative limit.

4.1.1.3. Generating outer request packet

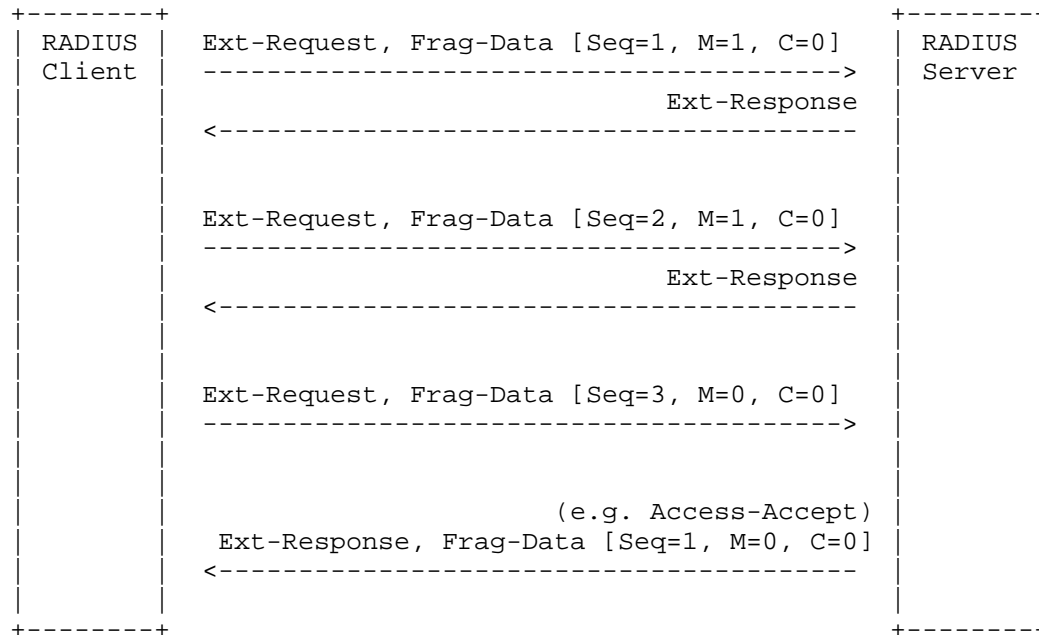
Once inner packet is generated it is fragmented and transmitted to RADIUS server in a series of outer Extension-Request/Extension-Response packet exchanges.

For each fragment following method is used to construct outer packet sent to the RADIUS server:

1. Fragment-Data head attribute added to Extension-Request packet.
2. Inner packet header authenticator field copied to Fragment-Data "Inner Authenticator" field. See section 4.1.5 for Fragment-Data format.
3. Inner packets header code field copied to Fragment-Data "Inner Code" field.
4. Fragment-Data Sequence field is incremented. First fragment starts at 1.
5. If there are more fragments to transmit Fragment-Data M bit is set 1. M bit is set 0 if current fragment is last.
6. Attributes are appended to outer packet from inner packet until either per-fragment length limit is reached or there are no more attributes remaining within inner packet. Attributes are copied in order they appear within inner packet without modification. An attribute is copied to outer packet only while there is enough space remaining to accommodate the full attribute. If sufficient space is unavailable attribute is sent with the next fragment.
7. Request authenticator of outer packet calculated normally per [RFC2866]

4.1.1.4. Transmitting fragments to RADIUS server

Once an outer packet is sent server responds with Extended-Response acknowledging receipt. The process is repeated until all fragments are delivered. Receipt of final fragment is acknowledged by RADIUS Servers response to assembled inner packet rather than a final Extended-Response.



4.1.1.5. Server state and fragment validation

RADIUS servers processing fragmented requests from clients SHALL perform the following validation steps:

1. Request authenticator of outer packet validated per [RFC2866]. If validation fails the outer packet is silently discarded.
2. Attributes contained within outer packet must be well formed otherwise outer packet is silently discarded.
3. Upon receipt of first fragment Fragment-Data head attribute sequence field shall be 1. In this case server allocates state necessary to uniquely track and assemble this and all subsequent fragments using Fragment-Data excluding Sequence and Flag fields as a unique identifier. If a request is received

in which sequence number is greater than 1 and for which no state exists the outer packet responds with Extended-Reject packet containing attribute Error-Cause = "Invalid Request"

4. If Continuation bit is 1 or Sequence number is 0 within Fragment-Data head attribute outer packet is silently discarded.
5. If "Inner code" field is Extended-Request, Extended-Response or Extended-Reject the outer packet is silently discarded.
6. If a request is received in which Fragment-Data sequence number is less than sequence of previously accepted fragment or does not contain the next expected sequential value the outer packet is silently discarded.
7. If outer packet is less than 400 bytes and More bit is 1 or if storing this fragment would cause total inner packet length to exceed the set inner packet limit then state for this request is removed and outer packet responds with Extended-Reject packet containing attribute Error-Cause = "Administratively Prohibited"

In any case described above where outer packet is silently discarded this MUST NOT trigger release of stored state for fragment assembly.

4.1.2. Server to client packet fragmentation

This section describes method for RADIUS server to fragment and transmit response packets (e.g. Access-Accept) to client.

4.1.2.1. Prerequisites

A RADIUS server SHALL meet one or more of the following requirements before sending a fragmented response.

1. Administrative option indicating client support for fragments.
2. Request packet from RADIUS client delivered using fragments.
3. Request contained Fragment-Reply-Supported attribute having value of Yes (4000).

4.1.2.2. Preparing inner response packet

RADIUS server generates a complete RADIUS response packet in a storage buffer rather than transmitting to RADIUS client. Prior to

generating response packet following changes to normal processing SHALL be observed:

1. Identifier field of RADIUS response is set 0. This field is unused while inner packet is encapsulated in outer packet.
2. 4096 byte maximum packet length [RFC2865] limit is replaced with lower of following two constraints:
 - A. Maximum Length RADIUS header field can accommodate. (e.g. 65535 bytes)
 - B. Administrative limit.

4.1.2.3. Generating outer response packet

Once inner packet is generated it is fragmented and transmitted to RADIUS client in a series of outer Extension-Response/Extension-Request packet exchanges.

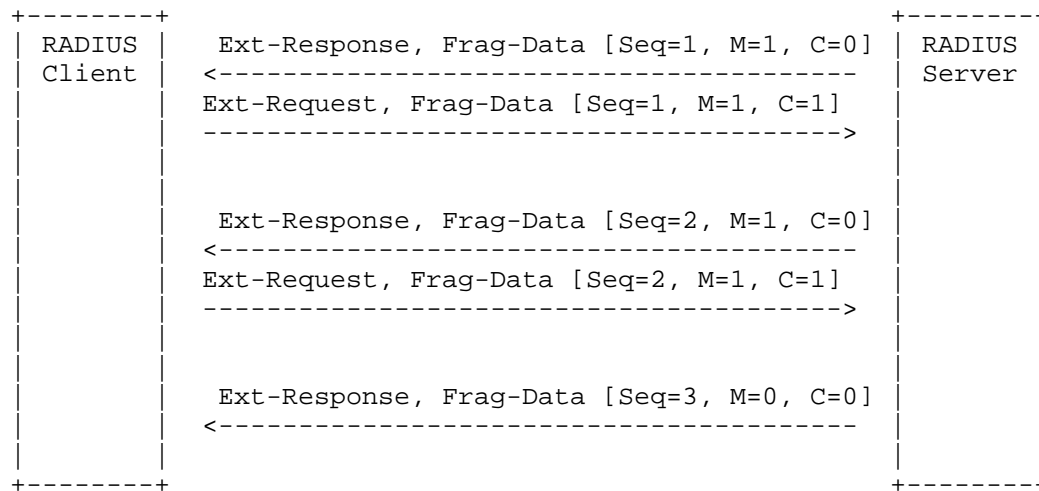
For each fragment following method is used to construct outer packet sent to the RADIUS client:

1. Fragment-Data head attribute added to Extension-Response packet.
2. Inner packet header authenticator field copied to Fragment-Data "Inner Authenticator" field. See section 4.1.5 for Fragment-Data format.
3. Inner packets header code field copied to Fragment-Data "Inner Code" field.
4. Fragment-Data Sequence field is incremented. First fragment starts at 1.
5. If there are more fragments to transmit Fragment-Data M bit is set 1. M bit is set 0 if current fragment is last.
6. Attributes are appended to outer packet from inner packet until either per-fragment length limit is reached or there are no more attributes remaining within inner packet. Attributes are copied in order they appear within inner packet without modification. An attribute is copied to outer packet only while there is enough space remaining to accommodate the full attribute. If sufficient space is unavailable attribute is sent with the next fragment.

7. Response authenticator of outer packet calculated normally per [RFC2866]

4.1.2.4. Transmitting fragments to RADIUS client

Once an outer packet (Extended-Response) is sent RADIUS client responds by issuing an Extended-Request containing Fragment-Data head attribute copied from response with Continuation bit set to 1. The process is repeated until all fragments are delivered. Receipt of final fragment is unacknowledged.



4.1.2.5. Client state and fragment validation

RADIUS clients processing fragmented responses from servers SHALL perform the following validation steps:

1. Response authenticator of outer packet validated per [RFC2866]. If validation fails outer packet is silently discarded.
2. Attributes contained within outer packet must be well formed otherwise outer packet is silently discarded.
3. Upon receipt of first response fragment Fragment-Data head attribute sequence field shall be 1 otherwise the outer packet is silently discarded.
4. If Continuation bit is 1 within Fragment-Data head attribute outer packet is silently discarded.

5. If "Inner code" field is Extended-Request, Extended-Response or Extended-Reject the outer packet is silently discarded.
6. If a response is received in which Fragment-Data sequence number is less than sequence of previously accepted fragment or does not contain the next expected sequence value outer packet is silently discarded.

4.1.3. Inner packet reassembly

Once all fragments are received then inner packet is assembled by reversing fragmentation process.

Inner packet RADIUS header is generated as follows:

1. Code = Fragment-Data "Inner Code" field
2. Identifier = 0
3. Length = RADIUS header length (e.g. 20) + sum of all attributes within all fragments excluding Fragment-Data head attribute of each fragment.
4. Authenticator = Fragment-Data "Inner Authenticator" field

Attributes are appended to packet in order received excluding Fragment-Data head attribute of each fragment.

Fully assembled inner packet is processed normally as if packet were received from the network including validation of applicable authenticator fields.

4.1.4. Request response options

RADIUS clients supporting fragmentation MUST be capable of accepting a fragmented response in reply to a non-fragmented request.

If a request is fragmented response SHALL also be fragmented.

4.1.4.1. Fragmented response to non-fragmented request

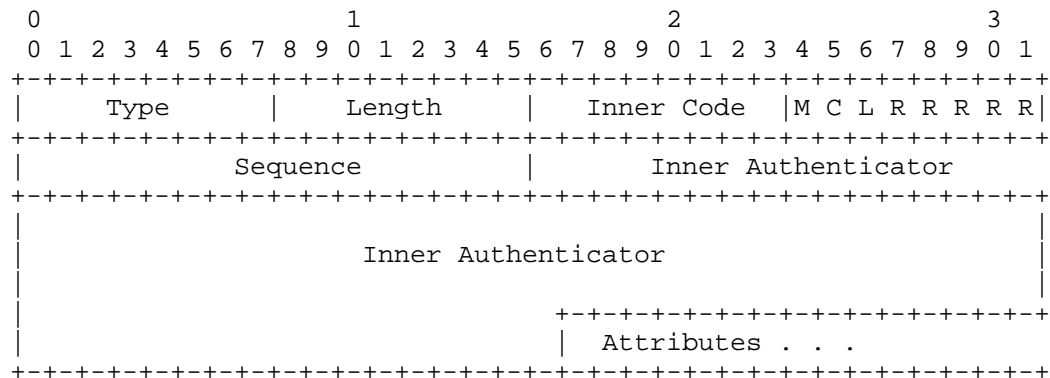
The non-fragmented requests authenticator is used both during production of the inner RADIUS response packet and production of the first outer packet carrying first fragmented response.

When client reassembles fragmented response to obtain inner packet original request authenticator from non-fragmented request is used to validate the inner packets response authenticator.

4.1.4.2. Fragmented response to fragmented request

Inner and outer packets authenticators are separate.

4.1.5. Fragment-Data head attribute format



Type

TBD for Fragment-Data

Length

24

Inner Code

Represents inner packet header code field

Flags

M - More

1 = Additional fragments pending

0 = Final fragment

C - Continuation

1 = RADIUS client requests next response fragment

0 = Packet contains inner packet fragments

L - Reserved

This field is reserved for future use and MUST be set 0.

R - Reserved

These fields are reserved for future use and MUST be set 0.

Sequence

The 16-bit unsigned fragment sequence number in network byte order.

Inner Authenticator

Represents inner packet header authenticator field

4.1.6. Server state management

RADIUS servers are required to keep state to facilitate assembly of fragments into completed inner packets. This process occurs between client and server and cannot be proxied. RADIUS servers may do no processing of a request until after the completed inner packet has been transmitted. These constraints allow selection of a low threshold for retention of state to account for worse case round trip delays, server delay and client retransmission policy. It is RECOMMENDED state be kept for no longer than 30 seconds after last successfully received fragment.

It is recommended RADIUS servers develop a robust policy for managing state to guard against resource exhaustion. One method of accomplishing this is to alter state retention period proportionally to pressure on state resources until a lower threshold is reached. Beyond this if state is exhausted the RADIUS server MAY respond to new fragment requests by sending Extended-Reject containing attribute Error-Cause = "Resources Unavailable". Upon receipt clients MAY attempt to contact an alternate RADIUS server if available.

4.1.7. Per-fragment and inner packet sizing

Minimum length of an outer packet shall be 400 bytes when the More bit is set 1.

Maximum acceptable length of an outer packet varies between 400 and 4096 bytes inclusive for UDP or 65535 bytes for TCP. Using Fragment-Inquire request described in section 4.2 clients MAY obtain

Framed-MTU hint from RADIUS server to size request fragments on IP packet boundaries.

For clients to signal MTU to server client includes Framed-MTU hint in fragmented or non-fragmented Access-Request packet.

RADIUS packets SHOULD only be fragmented if they would exceed 4096 bytes in length. It is not recommended MTU hinting be used as a threshold to fragment attributes.

It is RECOMMENDED by default servers support inner packets up to 65535 bytes in length. Administrative settings SHOULD be available to allow operators to change the limit.

4.2. Fragment-Inquire

Fragment-Inquire requests optional fragment related capabilities and parameters from the RADIUS server.

4.2.1. Request

When RADIUS is used over a connection based transport (e.g. TLS/DTLS) Fragment-Inquire requests are sent upon initial connection. When used over a connectionless transport (e.g. UDP) Fragment-Inquire requests are sent upon startup or before issuing first RADIUS request.

If RADIUS is used over a connection based transport following additional attributes MAY be included after the Fragment-Inquire head attribute.

Fragment-Stream-Limit (Section 6.4)

Fragment-Reply-Supported (Section 6.1)

If RADIUS is used over a connectionless transport no additional attributes SHALL be sent.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type           |  Length       |  Extended-Type  |  Value           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                                         |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type

TBD

Length

7

Extended-Type

TBD

Value (Integer)

1

4.2.2. Response

In response to Fragment-Inquire the following attributes MAY be present within Extended-Response packet.

Fragment-Reply-Allowed	Section 6.2
Fragment-Stream-Limit	Section 6.3
Fragment-Limit	Section 6.4
Fragment-Inquire-Interval	Section 6.5
Framed-MTU	Section 6.6 [RFC2865]
Event-Timestamp	Section 6.7 [RFC2869]

5. Compatibility

While procedures described in this text provide a means to manage compatibility amongst servers and clients capable systems are still required to exchange RADIUS packets beyond 4096 bytes. Clients and Servers may not send or process large packets unless both client and server support large packets. Likewise proxy systems may not process large packets unless they are capable of processing them.

If a RADIUS server is part of a proxy network without support for packets larger than 4096 bytes it SHOULD NOT advertise support for fragmentation or TCP large packet support via Fragment-Inquire. In this case clients and servers should seek to minimize the need for large packets by other means (e.g. limiting capability, out-of-band signals or compression) until such time remaining systems can be upgraded.

6. Attributes

6.1. Fragment-Reply-Supported

When sent within an Access-Request packet this attribute signals client support for fragmented response of packets beyond 4096 bytes. (e.g. Access-Accept).

This attribute MUST only be transmitted from client after a successful Fragment-Inquire request contains attribute Fragment-Reply-Allowed=1 within its response. This attribute MUST NOT be administratively configured nor SHALL it be forwarded for proxy unless the same procedure is followed.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Type										Length										Value																			
Value (cont.)																																							

Type

TBD

Length

6

Value (Integer)

4000 = Yes

All other values = No

6.2. Fragment-Reply-Allowed

If RADIUS server supports Fragment-Reply-Supported attribute described in section 6.1 then Fragment-Reply-Allowed is sent within Extended-Response to Fragment-Inquire head attribute.

The server MUST meet following requirements before this attribute can be sent.

1. Server supports Fragment-Data head attribute or is using TCP transport supporting RADIUS packets exceeding 4096 bytes.

2. Server will not forward Fragment-Reply-Supported attribute for proxy unless same client procedure described in section 6.1 has been followed toward forward proxy destination.

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type      |  Length    | Extended-Type |  Value      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                                     |
+-----+-----+-----+-----+-----+-----+

```

Type

TBD

Length

7

Extended-Type

TBD

Value (Integer)

1

6.3. Fragment-Stream-Limit

When sent in response to Fragment-Inquire head attribute this attribute signals to RADIUS clients maximum (non-fragmented) RADIUS packet length in bytes accepted by server over a stream (e.g. TCP) transport.

This attribute MUST NOT be sent if RADIUS server does not support any stream transports or is otherwise unable to offer support to the client. Attribute MUST NOT be sent if maximum packet length supported by RADIUS server over stream transport is 4096 bytes or less.

When sent to UDP clients this provides a hint client MAY establish a stream connection while there is a need to send requests larger than 4096 bytes or where a large response is anticipated.

When sent to non-UDP clients the client MUST NOT attempt to send a request packet larger than indicated by Fragment-Stream-Limit.

A RADIUS client SHOULD only switch from UDP if protocol being switched to offers the same or higher level of security.

When sent within a Fragment-Inquire head attribute this signals to server that client is capable of receiving response packets up to length indicated by Fragment-Stream-Limit. If this attribute was sent by a UDP client it MUST be ignored.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type           |  Length       |  Extended-Type  |  Value           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                                         |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type

TBD

Length

7

Extended-Type

TBD

Value (Integer)

> 4096 (bytes)

6.4. Fragment-Limit

When sent in response to Fragment-Inquire head attribute this signals maximum fragmented inner packet length in bytes server is willing to accept and advertises to clients that fragmentation is supported. If Fragment-Limit is sent it MUST be greater than 4096 bytes.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type           |  Length       |  Extended-Type  |  Value           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                                         |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type

TBD

Length

7

Extended-Type

TBD

Value (Integer)

>4096 bytes

6.5. Fragment-Inquire-Interval

When sent in response to Fragment-Inquire head attribute this attribute controls interval in seconds at which additional Fragment-Inquire requests should be sent in order for clients to be made aware of configuration changes. If the attribute is not set clients are not expected to check periodically for configuration change.

Fragment-Inquire-Interval SHOULD be no less than 60 seconds.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										Length										Extended-Type										Value									

Type

TBD

Length

7

Extended-Type

TBD

Value (Integer)

>= 60

6.6. Framed-MTU

When sent in response to Fragment-Inquire head attribute Framed-MTU [RFC2865] provides client a hint as to server MTU. (See also section 4.1.7)

6.7. Event-Timestamp

When sent in response to Fragment-Inquire head attribute Event-Timestamp [RFC2869] reflects time at which server generated Extended-Response packet.

7. Table of Attributes

The following table provides a guide to which attributes may be found in which kinds of packets and in what quantity.

Auth	Acct	Disconnect	CoA	Extended-Req
REQ ACK NAK	REQ ACK	REQ ACK NAK	REQ ACK NAK	REQ ACK NAK
Fragment-Data	0 0 0	0 0 0	0 0 0	0-1 0-1 0
Fragment-Inquire	0 0 0	0 0 0	0 0 0	0-1 0 0
Fragment-Reply-Supported	0-1 0 0	0 0 0	0 0 0	0-1 0 0
Fragment-Reply-Allowed	0 0 0	0 0 0	0 0 0	0 0-1 0
Fragment-Stream-Limit	0 0 0	0 0 0	0 0 0	0-1 0-1 0
Fragment-Limit	0 0 0	0 0 0	0 0 0	0 0-1 0
Fragment-Inquire-Interval	0 0 0	0 0 0	0 0 0	0 0-1 0

0 This attribute MUST NOT be present in packet.

0-1 Zero or one instance of this attribute MAY be present in packet.

8. Security Considerations

An attacker can collect and replay previous fragment exchanges in a bid to overwhelm server resources or cause previous packets to be reprocessed.

All security considerations which apply to confidentiality, integrity and availability of RADIUS Accounting [RFC2866] packets apply to this document.

9. IANA Considerations

This document defines following RADIUS attribute types and packet codes.

Protocols / RADIUS Types / RADIUS Packet Type Codes

- TBD - Extended-Request
- TBD - Extended-Response
- TBD - Extended-Reject

Protocols / RADIUS Types / RADIUS Attribute Types (Standard Space)

- TBD - Fragment-Data
- TBD - Fragment-Reply-Supported

Protocols / RADIUS Types / RADIUS Attribute Types (Extended Space)

- TBD - Fragment-Inquire
- TBD - Fragment-Reply-Allowed
- TBD - Fragment-Stream-Limit
- TBD - Fragment-Limit
- TBD - Fragment-Inquire-Interval

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC2869] RADIUS Extensions C. Rigney, W. Willats, P. Calhoun June 2000

[RFC5176] Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS) M. Chiba, G. Dommety, M. Eklund, D. Mitton, B. Aboba, January 2008

[RFC6613] RADIUS over TCP A. DeKok, May 2012

[RFC6614] Transport Layer Security (TLS) Encryption for RADIUS S. Winter, M. McCauley, S. Venaas, K. Wierenga, May 2012

[RFC6929] Remote Authentication Dial In User Service (RADIUS) Protocol Extensions. A. DeKok, A. Lior. April 2013.

11. Acknowledgments

Author would like to thank Sam Hartman for valuable ideas.

Authors' Addresses

Peter Deacon
IEA Software, Inc.
P.O. Box 1170
Veradale, WA 99037
USA

Email: peterd@iea-software.com

Network Working Group
INTERNET-DRAFT
Updates: 2865,3162,6158,6572
Category: Standards Track
<draft-dekok-radext-datatypes-06.txt>
1 April 2015

DeKok, Alan
FreeRADIUS

Data Types in the Remote Authentication
Dial-In User Service Protocol (RADIUS)
draft-dekok-radext-datatypes-06.txt

Abstract

RADIUS specifications have used data types for two decades without defining them as managed entities. During this time, RADIUS implementations have named the data types, and have used them in attribute definitions. This document updates the specifications to better follow established practice. We do this by naming the data types defined in RFC 6158, which have been used since at least RFC 2865. We provide an IANA registry for the data types, and update the RADIUS Attribute Type registry to include a "Data Type" field for each attribute. Finally, we recommend that authors of RADIUS specifications use these types in preference to existing practice.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on October 1, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info/>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Specification use of Data Types	4
1.2. Implementation use of Data Types	4
1.3. Requirements Language	5
2. Data Type Definitions	6
2.1. integer	7
2.2. enum	8
2.3. ipv4addr	8
2.4. time	9
2.5. text	10
2.6. string	10
2.7. concat	11
2.8. ifid	12
2.9. ipv6addr	13
2.10. ipv6prefix	14
2.11. ipv4prefix	15
2.12. integer64	16
2.13. tlv	16
2.14. vsa	18
2.15. extended	19
2.16. long-extended	20
2.17. evs	22
3. Updated Registries	24
3.1. Create a Data Type Registry	24
3.2. Updates to the Attribute Type Registry	25
4. Suggestions for Specifications	30
5. Security Considerations	31
6. IANA Considerations	31
7. References	31
7.1. Normative References	31
7.2. Informative References	32

1. Introduction

RADIUS specifications have historically defined attributes in terms of name, type value, and data type. Of these three pieces of information, only the type value is managed by IANA. There is no management of, or restriction on, the attribute name, as discussed in [RFC6929] Section 2.7.1. There is no management of data type name or definition. This document defines an IANA registry for data types, and updates the RADIUS Attribute Type registry to use those newly defined data types.

In this section, we review the use of data types in specifications and implementations. We highlight ambiguities and inconsistencies. The rest of this document is devoted to resolving those problems.

1.1. Specification use of Data Types

A number of data type names and definitions are given in [RFC2865] Section 5, at the bottom of page 25. These data types are named and clearly defined. However, this practice was not continued in later specifications.

Specifically, [RFC2865] defines attributes of data type "address" to carry IPv4 addresses. Despite this definition, [RFC3162] defines attributes of data type "Address" to carry IPv6 addresses. We suggest that the use of the word "address" to refer to disparate data types is problematic.

Other failures are that [RFC3162] does not give a data type name and definition for the data types IPv6 address, Interface-Id, or IPv6 prefix. [RFC2869] defines Event-Timestamp to carry a time, but does not re-use the "time" data type defined in [RFC2865]. Instead, it just repeats the "time" definition. [RFC6572] defines multiple attributes which carry IPv4 prefixes. However, an "IPv4 prefix" data type is not named, defined as a data type, or called out as an addition to RADIUS. Further, [RFC6572] does not follow the recommendations of [RFC6158], and does not explain why it fails to follow those recommendations.

These ambiguities and inconsistencies need to be resolved.

1.2. Implementation use of Data Types

RADIUS implementations often use "dictionaries" to map attribute names to type values, and to define data types for each attribute. The data types in the dictionaries are defined by each implementation, but correspond to the "ad hoc" data types used in the specifications.

In effect, implementations have seen the need for well-defined data types, and have created them. It is time for RADIUS specifications to follow this practice.

This document requires no changes to any RADIUS implementation, past, present, or future. It instead documents existing practice, in order to simplify the process of writing RADIUS specifications, to clarify the interpretation of RADIUS standards, and to improve the communication between specification authors and IANA.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Data Type Definitions

This section defines the new data types. For each data type, it gives a definition, a name, a number, a length, and an encoding format. Where relevant, it describes subfields contained within the data type. These definitions have no impact on existing RADIUS implementations. There is no requirement that implementations use these names.

Where possible, the name of each data type has been taken from previous specifications. In some cases, a different name has been chosen. The change of name is sometimes required to avoid ambiguity (i.e. "address" versus "Address"). Otherwise, the new name has been chosen to be compatible with [RFC2865], or with use in common implementations. In some cases, new names are chosen to clarify the interpretation of the data type.

The numbers assigned herein for the data types have no meaning other than to permit them to be tracked by IANA. As RADIUS does not encode information about data types in a packet, the numbers assigned to a data type will never occur in a packet.

The encoding of each data type is taken from previous specifications. The fields are transmitted from left to right.

Where the data types have inter-dependencies, the simplest data type is given first, and dependent ones are given later.

We do not create specific data types for the "tagged" attributes, as discussed in [RFC2868]. That specification defines the "tagged" attributes as being backwards compatible with pre-existing data types. In addition, [RFC6158] Section 2.1 says that "tagged" attributes should not be used. There is therefore no benefit to defining additional data types for these attributes.

Similarly, we do not create data types for some attributes having complex structure, such as CHAP-Password, ARAP-Features, or Location-Capable. We need to strike a balance between correcting earlier mistakes, and making this document more complex. In some cases, it is better to treat complex attributes as being of type "string", even though they need to be interpreted by RADIUS implementations.

Implementations not supporting a particular data type MUST treat attributes of that data type as being of data type "string". See Section 2.6, below for a definition of the "string" data type.

The definitions below use specialized names for various fields of attributes and data types. These names serve to address ambiguity of

the field names in previous specifications. For example, the term "Value" is used in [RFC2865] Section 5 to define a field which carries the contents of attribute. It is then used in later sections as the sub-field of attribute contents. The result is that the field is defined as recursively containing itself. Similarly, "String" is used both as a data type, and as a sub-field of other data types.

This document uses slightly different terminology than previous specifications, in order to be avoid ambiguity. The first addition is the following term:

Attr-Data

The "Value" field of an Attribute as defined in [RFC2865] Section 5. The contents of this field MUST be a valid data type as defined in the RADIUS Data Type registry.

In this document, we use the term "Value" only to refer to the contents of a data type, where that data type cannot carry other data types. In other cases, we refer to the contents of a data type as "Type-Data", to distinguish it from data of other types. For example, a data type "vsa" will contain a data field called "VSA-Data".

These terms are used in preference to the term "String", which was used in multiple incompatible ways. It is RECOMMENDED that future specifications use the new terms in order to maintain consistent definitions, and to avoid ambiguities.

2.1. integer

The "integer" data type encodes a 32-bit unsigned integer in network byte order. Where the range of values for a particular attribute is limited to a sub-set of the values, specifications MUST define the valid range. Values outside of the allowed ranges SHOULD be treated as invalid.

Name

integer

Number

1

Length

Four octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Value                                                    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.2. enum

The "enum" data type encodes a 32-bit unsigned integer in network byte order. It differs from the "integer" data type only in that it is used to define enumerated types, such as Service-Type. Specifications MUST define a valid set of enumerated values, along with a unique name for each value.

Name

enum

Number

2

Length

Four octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Value                                                    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.3. ipv4addr

The "ipv4addr" data type encodes an IPv4 address in network byte order. Where the range of address for a particular attribute is limited to a sub-set of possible addresses, specifications MUST define the valid range(s). Values outside of the allowed range SHOULD be treated as invalid.

Name

ipv4addr

Number

3

Length

Four octets

Format

```

      0                               1                               2                               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Address           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.4. time

The "time" data type encodes time as a 32-bit unsigned value in network byte order and in seconds since 00:00:00 UTC, January 1, 1970. We note that dates before the year 2013 are likely to be erroneous.

Note that the "time" attribute is defined to be unsigned, which means it is not subject to a signed integer overflow in the year 2038.

Name

time

Number

4

Length

Four octets

Format

```

      0                               1                               2                               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Time           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.5. text

The "text" data type encodes UTF-8 text [RFC3629]. The maximum length of the text is given by the encapsulating attribute. Where the range of lengths for a particular attribute is limited to a sub-set of possible lengths, specifications MUST define the valid range(s).

Note that the "text" type does not terminate with a NUL octet (hex 00). The Attribute has a Length field and does not use a terminator. Texts of length zero (0) MUST NOT be sent; omit the entire attribute instead.

Name

text

Number

5

Length

One or more octets.

Format

```

0
0 1 2 3 4 5 6 7
+-----+
| Value   ...
+-----+
```

2.6. string

The "string" data type encodes binary data, as a sequence of undistinguished octets. Where the range of lengths for a particular attribute is limited to a sub-set of possible lengths, specifications MUST define the valid range(s).

Note that the "string" data type does not terminate with a NUL octet (hex 00). The Attribute has a Length field and does not use a terminator. Strings of length zero (0) MUST NOT be sent; omit the entire attribute instead.

Where there is a need to encapsulate complex data structures, and

TLVs cannot be used, the "string" data type MUST be used. This requirement include encapsulation of data structures defined outside of RADIUS, which are opaque to the RADIUS infrastructure. It also includes encapsulation of some data structures which are not opaque to RADIUS, such as the contents of CHAP-Password.

There is little reason to define a new RADIUS data type for only one attribute. However, where the complex data type cannot be represented as TLVs, and is expected to be used in many attributes, a new data type SHOULD be defined.

These requirements are stronger than [RFC6158], which makes the above encapsulation a "SHOULD". This document defines data types for use in RADIUS, so there are few reasons to avoid using them.

Name

string

Number

6

Length

One or more octets.

Format

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+
| Octets    ...
+---+---+---+---+

```

2.7. concat

The "concat" data type permits the transport of more than 253 octets of data in a "standard space" [RFC6929] attribute. It is otherwise identical to the "string" data type.

If multiple attributes of this data type are contained in a packet, all attributes of the same type code MUST be in order and they MUST be consecutive attributes in the packet.

The amount of data transported in a "concat" data type can be no more than the RADIUS packet size. In practice, the requirement to

transport multiple attributes means that the limit may be substantially smaller than one RADIUS packet. As a rough guide, is RECOMMENDED that this data type transport no more than 2048 octets of data.

The "concat" data type MAY be used for "standard space" attributes. It MUST NOT be used for attributes in the "short extended space" or the "long extended space". It MUST NOT be used in any field or subfields of the following data types: "tlv", "vsa", "extended", "long-extended", or "evs".

Name

concat

Number

7

Length

One or more octets.

Format

```

0
0 1 2 3 4 5 6 7
+-----+
| Octets   ...
+-----+
```

2.8. ifid

The "ifid" data type encodes an Interface-Id as an 8-octet string in network byte order.

Name

ifid

Number

8

Length

Eight octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Interface-ID ...
+-----+-----+-----+-----+-----+-----+-----+-----+
... Interface-ID
+-----+-----+-----+-----+-----+-----+-----+-----+

```

2.9. ipv6addr

The "ipv6addr" data type encodes an IPv6 address in network byte order. Where the range of address for a particular attribute is limited to a sub-set of possible addresses, specifications MUST define the valid range(s).

Name

ipv6addr

Number

9

Length

Sixteen octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Address ...
+-----+-----+-----+-----+-----+-----+-----+-----+
... Address ...
+-----+-----+-----+-----+-----+-----+-----+-----+
... Address ...
+-----+-----+-----+-----+-----+-----+-----+-----+
... Address
+-----+-----+-----+-----+-----+-----+-----+-----+

```

2.10. ipv6prefix

The "ipv6prefix" data type encodes an IPv6 prefix, using both a prefix length and an IPv6 address in network byte order.

Name

ipv6prefix

Number

10

Length

At least two, and no more than eighteen octets.

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Reserved   | Prefix-Length | Prefix ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
... Prefix ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
... Prefix ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
... Prefix
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Subfields

Reserved

This field, which is reserved and MUST be present, is always set to zero.

Prefix-Length

The length of the prefix, in bits. At least 0 and no larger than 128.

Prefix

The Prefix field is up to 16 octets in length. Bits outside of the Prefix-Length, if included, must be zero.

2.11. ipv4prefix

The "ipv4prefix" data type encodes an IPv4 prefix, using both a prefix length and an IPv4 address in network byte order.

Name

ipv4prefix

Number

11

Length

At least two, and no more than eighteen octets.

Format

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Reserved   | Prefix-Len | Prefix ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   ... Prefix   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Subfields

Reserved

This field, which is reserved and MUST be present, is always set to zero.

Prefix-Length

A 6-bit unsigned integer containing the length of the prefix, in bits. The values MUST be no larger than 32.

Prefix

The Prefix field is 4 octets in length. Bits outside of the Prefix-Length must be zero. Unlike the "ipv6prefix" data type, this field is fixed length. If the address is all zeros (i.e. "0.0.0.0", then the Prefix-Length MUST be set to 32.

2.12. integer64

The "integer64" data type encodes a 64-bit unsigned integer in network byte order. Where the range of values for a particular attribute is limited to a sub-set of the values, specifications **MUST** define the valid range(s).

Name

integer64

Number

12

Length

Eight octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Value ...
+-----+-----+-----+-----+-----+-----+-----+-----+
|      ... Value
+-----+-----+-----+-----+-----+-----+-----+-----+

```

2.13. tlv

The "tlv" data type encodes a type-length-value, as defined in [RFC6929] Section 2.3.

Name

tlv

Number

13

Length

Three or more octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  TLV-Type   |  TLV-Length   |  TLV-Data ...  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Subfields

TLV-Type

This field is one octet. Up-to-date values of this field are specified according to the policies and rules described in [RFC6929] Section 10. Values 254-255 are "Reserved" for use by future extensions to RADIUS. The value 26 has no special meaning, and MUST NOT be treated as a Vendor Specific attribute.

The TLV-Type is meaningful only within the context defined by "Type" fields of the encapsulating Attributes, using the dotted-number notation introduced in [RFC6929].

A RADIUS server MAY ignore Attributes with an unknown "TLV-Type".

A RADIUS client MAY ignore Attributes with an unknown "TLV-Type".

A RADIUS proxy SHOULD forward Attributes with an unknown "TLV-Type" verbatim.

TLV-Length

The TLV-Length field is one octet, and indicates the length of this TLV including the TLV-Type, TLV-Length and TLV-Value fields. It MUST have a value between 3 and 255. If a client or server receives a TLV with an invalid TLV-Length, then the attribute which encapsulates that TLV MUST be considered to be an "invalid attribute", and handled as per [RFC6929] Section 2.8.

TLVs having TLV-Length of zero (0) MUST NOT be sent; omit the entire TLV instead.

TLV-Data

The TLV-Data field is one or more octets and contains information specific to the Attribute. The format and length of the TLV-Data field is determined by the TLV-Type and TLV-

Length fields.

The TLV-Data field MUST contain only known RADIUS data types. The TLV-Data field MUST NOT contain any of the following data types: "concat", "vsa", "extended", "long-extended", or "evs".

2.14. vsa

The "vsa" data type encodes Vendor-Specific data, as given in [RFC2865] Section 5.26. It is used only in the Attr-Data field of a Vendor-Specific Attribute. It MUST NOT appear in the contents of any other data type.

Name

vsa

Number

14

Length

Five or more octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Vendor-Id                             |
+-----+-----+-----+-----+-----+-----+-----+-----+
| VSA-Data ....                                                                |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Subfields

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

VSA-Data

The VSA-Data field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished

octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

It SHOULD be encoded as a sequence of "tlv" fields. The interpretation of the TLV-Type and TLV-Data fields are dependent on the vendor's definition of that attribute.

The "vsa" data type MUST be used as contents of the Attr-Data field of the Vendor-Specific attribute. The "vsa" data type MUST NOT appear in the contents of any other data type.

2.15. extended

The "extended" data type encodes the "Extended Type" format, as given in [RFC6929] Section 2.1. It is used only in the Attr-Data field of an Attribute allocated from the "standard space". It MUST NOT appear in the contents of any other data type.

Name

extended

Number

15

Length

Two or more octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Extended-Type | Ext-Data ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Subfields

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified according to the policies and rules described in [RFC6929] Section 10. Unlike the Type field

defined in [RFC2865] Section 5, no values are allocated for experimental or implementation-specific use. Values 241-255 are reserved and MUST NOT be used.

The Extended-Type is meaningful only within a context defined by the Type field. That is, this field may be thought of as defining a new type space of the form "Type.Extended-Type". See [RFC6929] Section 2.5 for additional discussion.

A RADIUS server MAY ignore Attributes with an unknown "Type.Extended-Type".

A RADIUS client MAY ignore Attributes with an unknown "Type.Extended-Type".

Ext-Data

The contents of this field MUST be a valid data type as defined in the RADIUS Data Type registry. The Ext-Data field MUST NOT contain any of the following data types: "concat", "vsa", "extended", "long-extended", or "evs".

The Ext-Data field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Ext-Data field.

2.16. long-extended

The "long-extended" data type encodes the "Long Extended Type" format, as given in [RFC6929] Section 2.2. It is used only in the Attr-Data field of an Attribute. It MUST NOT appear in the contents of any other data type.

Name

long-extended

Number

16

Length

Three or more octets

Format

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
| Extended-Type |M| Reserved   | Ext-Data ...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Subfields

Extended-Type

This field is identical to the Extended-Type field defined above in Section 2.13.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. The More field **MUST** be clear (0) if the Length field has value less than 255. The More field **MAY** be set (1) if the Length field has value of 255.

If the More field is set (1), it indicates that the Ext-Data field has been fragmented across multiple RADIUS attributes. When the More field is set (1), the attribute **MUST** have a Length field of value 255; there **MUST** be an attribute following this one; and the next attribute **MUST** have both the same Type and Extended Type. That is, multiple fragments of the same value **MUST** be in order and **MUST** be consecutive attributes in the packet, and the last attribute in a packet **MUST NOT** have the More field set (1).

That is, a packet containing a fragmented attribute needs to contain all fragments of the attribute, and those fragments need to be contiguous in the packet. RADIUS does not support inter-packet fragmentation, which means that fragmenting an attribute across multiple packets is impossible.

If a client or server receives an attribute fragment with the "More" field set (1), but for which no subsequent fragment can be found, then the fragmented attribute is considered to be an "invalid attribute", and handled as per [RFC6929] Section 2.8.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations **MUST** set it to zero (0) when encoding an

attribute for sending in a packet. The contents SHOULD be ignored on reception.

Future specifications may define additional meaning for this field. Implementations therefore MUST NOT treat this field as invalid if it is non-zero.

Ext-Data

The contents of this field MUST be a valid data type as defined in the RADIUS Data Type registry. The Ext-Data field MUST NOT contain any of the following data types: "concat", "vsa", "extended", "long-extended", or "evs".

The Ext-Data field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Ext-Data field.

The length of the data MUST be taken as the sum of the lengths of the fragments (i.e. Ext-Data fields) from which it is constructed. Any interpretation of the resulting data MUST occur after the fragments have been reassembled. If the reassembled data does not match the expected format, each fragment MUST be treated as an "invalid attribute", and the reassembled data MUST be discarded.

We note that the maximum size of a fragmented attribute is limited only by the RADIUS packet length limitation. Implementations MUST be able to handle the case where one fragmented attribute completely fills the packet.

2.17. evs

The "evs" data type encodes an "Extended Vendor-Specific" attribute, as given in [RFC6929] Section 2.4. The "evs" data type is used solely to extend the Vendor Specific space. It MAY appear inside of an "extended" or a "long-extended" data type. It MUST NOT appear in the contents of any other data type.

Name

evs

Number

17

Length

Six or more octets

Format

```

      0                               1                               2                               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               Vendor-Id                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Vendor-Type   | EVS-Data ....                                         |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Subfields

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

EVS-Data

The EVS-Data field is one or more octets. It SHOULD encapsulate a previously defined RADIUS data type. Non-standard data types SHOULD NOT be used. We note that the EVS-Data field may be of data type "tlv".

The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets. We recognise that Vendors have complete control over the contents and format of the Ext-Data field, while at the same time recommending that good practices be followed.

Further codification of the range of allowed usage of this field is outside the scope of this specification.

3. Updated Registries

This section defines a new IANA registry for RADIUS data types, and updates the existing RADIUS Attribute Type registry.

3.1. Create a Data Type Registry

This section defines a new RADIUS registry, called "Data Type". Allocation in this registry requires IETF Review. The "Registration Procedures" for this registry are "Standards Action".

The registry contains three columns of data, as follows.

Value

The number of the data type. The value field is an artifact of the registry, and has no on-the-wire meaning.

Description

The name of the data type. The name field is used only for the registry, and has no on-the-wire meaning.

Reference

The specification where the data type was defined.

The initial contents of the registry are as follows.

Value	Description	Reference
-----	-----	-----
1	integer	[RFC2865], TBD
2	enum	[RFC2865], TBD
3	ipv4addr	[RFC2865], TBD
4	time	[RFC2865], TBD
5	text	[RFC2865], TBD
6	string	[RFC2865], TBD
7	concat	TBD
8	ifid	[RFC3162], TBD
9	ipv6addr	[RFC3162], TBD
10	ipv6prefix	[RFC3162], TBD
11	ipv4prefix	[RFC6572], TBD
12	integer64	[RFC6929], TBD
13	tlv	[RFC6929], TBD
14	evs	[RFC6929], TBD
15	extended	[RFC6929], TBD
16	long-extended	[RFC6929], TBD

3.2. Updates to the Attribute Type Registry

This section updates the RADIUS Attribute Type Registry to have a new column, which is inserted in between the existing "Description" and "Reference" columns. The new column is named "Data Type". The contents of that column are the name of a data type, corresponding to the attribute in that row, or blank if the attribute type is unassigned. The name of the data type is taken from the RADIUS Data Type registry, defined above.

The updated registry follows in CSV format.

```
Value,Description,Data Type,Reference
1,User-Name,string,[RFC2865]
2,User-Password,string,[RFC2865]
3,CHAP-Password,string,[RFC2865]
4,NAS-IP-Address,ipv4addr,[RFC2865]
5,NAS-Port,integer,[RFC2865]
6,Service-Type,enum,[RFC2865]
7,Framed-Protocol,enum,[RFC2865]
8,Framed-IP-Address,ipv4addr,[RFC2865]
9,Framed-IP-Netmask,ipv4addr,[RFC2865]
10,Framed-Routing,enum,[RFC2865]
11,Filter-Id,text,[RFC2865]
12,Framed-MTU,integer,[RFC2865]
13,Framed-Compression,enum,[RFC2865]
14>Login-IP-Host,ipv4addr,[RFC2865]
15>Login-Service,enum,[RFC2865]
16>Login-TCP-Port,integer,[RFC2865]
17,Unassigned,,
18,Reply-Message,text,[RFC2865]
19,Callback-Number,text,[RFC2865]
20,Callback-Id,text,[RFC2865]
21,Unassigned,,
22,Framed-Route,text,[RFC2865]
23,Framed-IPX-Network,ipv4addr,[RFC2865]
24,State,string,[RFC2865]
25,Class,string,[RFC2865]
26,Vendor-Specific,vsa,[RFC2865]
27,Session-Timeout,integer,[RFC2865]
28,Idle-Timeout,integer,[RFC2865]
29,Termination-Action,enum,[RFC2865]
30,Called-Station-Id,text,[RFC2865]
31,Calling-Station-Id,text,[RFC2865]
32,NAS-Identifier,text,[RFC2865]
33,Proxy-State,string,[RFC2865]
34>Login-LAT-Service,text,[RFC2865]
35>Login-LAT-Node,text,[RFC2865]
```

36, Login-LAT-Group, string, [RFC2865]
37, Framed-AppleTalk-Link, integer, [RFC2865]
38, Framed-AppleTalk-Network, integer, [RFC2865]
39, Framed-AppleTalk-Zone, text, [RFC2865]
40, Acct-Status-Type, enum, [RFC2866]
41, Acct-Delay-Time, integer, [RFC2866]
42, Acct-Input-Octets, integer, [RFC2866]
43, Acct-Output-Octets, integer, [RFC2866]
44, Acct-Session-Id, text, [RFC2866]
45, Acct-Authentic, enum, [RFC2866]
46, Acct-Session-Time, integer, [RFC2866]
47, Acct-Input-Packets, integer, [RFC2866]
48, Acct-Output-Packets, integer, [RFC2866]
49, Acct-Terminate-Cause, enum, [RFC2866]
50, Acct-Multi-Session-Id, text, [RFC2866]
51, Acct-Link-Count, integer, [RFC2866]
52, Acct-Input-Gigawords, integer, [RFC2869]
53, Acct-Output-Gigawords, integer, [RFC2869]
54, Unassigned, ,
55, Event-Timestamp, time, [RFC2869]
56, Egress-VLANID, integer, [RFC4675]
57, Ingress-Filters, enum, [RFC4675]
58, Egress-VLAN-Name, text, [RFC4675]
59, User-Priority-Table, string, [RFC4675]
60, CHAP-Challenge, string, [RFC2865]
61, NAS-Port-Type, enum, [RFC2865]
62, Port-Limit, integer, [RFC2865]
63, Login-LAT-Port, text, [RFC2865]
64, Tunnel-Type, enum, [RFC2868]
65, Tunnel-Medium-Type, enum, [RFC2868]
66, Tunnel-Client-Endpoint, text, [RFC2868]
67, Tunnel-Server-Endpoint, text, [RFC2868]
68, Acct-Tunnel-Connection, text, [RFC2867]
69, Tunnel-Password, text, [RFC2868]
70, ARAP-Password, string, [RFC2869]
71, ARAP-Features, string, [RFC2869]
72, ARAP-Zone-Access, enum, [RFC2869]
73, ARAP-Security, integer, [RFC2869]
74, ARAP-Security-Data, text, [RFC2869]
75, Password-Retry, integer, [RFC2869]
76, Prompt, enum, [RFC2869]
77, Connect-Info, text, [RFC2869]
78, Configuration-Token, text, [RFC2869]
79, EAP-Message, concat, [RFC2869]
80, Message-Authenticator, string, [RFC2869]
81, Tunnel-Private-Group-ID, text, [RFC2868]
82, Tunnel-Assignment-ID, text, [RFC2868]
83, Tunnel-Preference, integer, [RFC2868]

84, ARAP-Challenge-Response, string, [RFC2869]
85, Acct-Interim-Interval, integer, [RFC2869]
86, Acct-Tunnel-Packets-Lost, integer, [RFC2867]
87, NAS-Port-Id, text, [RFC2869]
88, Framed-Pool, text, [RFC2869]
89, CUI, string, [RFC4372]
90, Tunnel-Client-Auth-ID, text, [RFC2868]
91, Tunnel-Server-Auth-ID, text, [RFC2868]
92, NAS-Filter-Rule, text, [RFC4849]
93, Unassigned, ,
94, Originating-Line-Info, string, [RFC7155]
95, NAS-IPv6-Address, ipv6addr, [RFC3162]
96, Framed-Interface-Id, ifid, [RFC3162]
97, Framed-IPv6-Prefix, ipv6prefix, [RFC3162]
98, Login-IPv6-Host, ipv6addr, [RFC3162]
99, Framed-IPv6-Route, text, [RFC3162]
100, Framed-IPv6-Pool, text, [RFC3162]
101, Error-Cause Attribute, enum, [RFC3576]
102, EAP-Key-Name, string, [RFC4072] [RFC7268]
103, Digest-Response, text, [RFC5090]
104, Digest-Realm, text, [RFC5090]
105, Digest-Nonce, text, [RFC5090]
106, Digest-Response-Auth, text, [RFC5090]
107, Digest-Nextnonce, text, [RFC5090]
108, Digest-Method, text, [RFC5090]
109, Digest-URI, text, [RFC5090]
110, Digest-Qop, text, [RFC5090]
111, Digest-Algorithm, text, [RFC5090]
112, Digest-Entity-Body-Hash, text, [RFC5090]
113, Digest-CNonce, text, [RFC5090]
114, Digest-Nonce-Count, text, [RFC5090]
115, Digest-Username, text, [RFC5090]
116, Digest-Opaque, text, [RFC5090]
117, Digest-Auth-Param, text, [RFC5090]
118, Digest-AKA-Auts, text, [RFC5090]
119, Digest-Domain, text, [RFC5090]
120, Digest-Stale, text, [RFC5090]
121, Digest-HA1, text, [RFC5090]
122, SIP-AOR, text, [RFC5090]
123, Delegated-IPv6-Prefix, ipv6prefix, [RFC4818]
124, MIP6-Feature-Vector, string, [RFC5447]
125, MIP6-Home-Link-Prefix, ipv6prefix, [RFC5447]
126, Operator-Name, text, [RFC5580]
127, Location-Information, string, [RFC5580]
128, Location-Data, string, [RFC5580]
129, Basic-Location-Policy-Rules, string, [RFC5580]
130, Extended-Location-Policy-Rules, string, [RFC5580]
131, Location-Capable, enum, [RFC5580]

132, Requested-Location-Info, enum, [RFC5580]
133, Framed-Management-Protocol, enum, [RFC5607]
134, Management-Transport-Protection, enum, [RFC5607]
135, Management-Policy-Id, text, [RFC5607]
136, Management-Privilege-Level, integer, [RFC5607]
137, PKM-SS-Cert, concat, [RFC5904]
138, PKM-CA-Cert, concat, [RFC5904]
139, PKM-Config-Settings, string, [RFC5904]
140, PKM-Cryptosuite-List, string, [RFC5904]
141, PKM-SAID, text, [RFC5904]
142, PKM-SA-Descriptor, string, [RFC5904]
143, PKM-Auth-Key, string, [RFC5904]
144, DS-Lite-Tunnel-Name, text, [RFC6519]
145, Mobile-Node-Identifier, string, [RFC6572]
146, Service-Selection, text, [RFC6572]
147, PMIP6-Home-LMA-IPv6-Address, ipv6addr, [RFC6572]
148, PMIP6-Visited-LMA-IPv6-Address, ipv6addr, [RFC6572]
149, PMIP6-Home-LMA-IPv4-Address, ipv4addr, [RFC6572]
150, PMIP6-Visited-LMA-IPv4-Address, ipv4addr, [RFC6572]
151, PMIP6-Home-HN-Prefix, ipv6prefix, [RFC6572]
152, PMIP6-Visited-HN-Prefix, ipv6prefix, [RFC6572]
153, PMIP6-Home-Interface-ID, ifid, [RFC6572]
154, PMIP6-Visited-Interface-ID, ifid, [RFC6572]
155, PMIP6-Home-IPv4-HoA, ipv4prefix, [RFC6572]
156, PMIP6-Visited-IPv4-HoA, ipv4prefix, [RFC6572]
157, PMIP6-Home-DHCP4-Server-Address, ipv4addr, [RFC6572]
158, PMIP6-Visited-DHCP4-Server-Address, ipv4addr, [RFC6572]
159, PMIP6-Home-DHCP6-Server-Address, ipv6addr, [RFC6572]
160, PMIP6-Visited-DHCP6-Server-Address, ipv6addr, [RFC6572]
161, PMIP6-Home-IPv4-Gateway, ipv4addr, [RFC6572]
162, PMIP6-Visited-IPv4-Gateway, ipv4addr, [RFC6572]
163, EAP-Lower-Layer, enum, [RFC6677]
164, GSS-Acceptor-Service-Name, text, [RFC7055]
165, GSS-Acceptor-Host-Name, text, [RFC7055]
166, GSS-Acceptor-Service-Specifics, text, [RFC7055]
167, GSS-Acceptor-Realm-Name, text, [RFC7055]
168, Framed-IPv6-Address, ipv6addr, [RFC6911]
169, DNS-Server-IPv6-Address, ipv6addr, [RFC6911]
170, Route-IPv6-Information, ipv6prefix, [RFC6911]
171, Delegated-IPv6-Prefix-Pool, text, [RFC6911]
172, Stateful-IPv6-Address-Pool, text, [RFC6911]
173, IPv6-6rd-Configuration, tlv, [RFC6930]
174, Allowed-Called-Station-Id, text, [RFC7268]
175, EAP-Peer-Id, string, [RFC7268]
176, EAP-Server-Id, string, [RFC7268]
177, Mobility-Domain-Id, integer, [RFC7268]
178, Preauth-Timeout, integer, [RFC7268]
179, Network-Id-Name, string, [RFC7268]

180, EAPoL-Announcement, concat, [RFC7268]
181, WLAN-HESSID, text, [RFC7268]
182, WLAN-Venue-Info, integer, [RFC7268]
183, WLAN-Venue-Language, string, [RFC7268]
184, WLAN-Venue-Name, text, [RFC7268]
185, WLAN-Reason-Code, integer, [RFC7268]
186, WLAN-Pairwise-Cipher, integer, [RFC7268]
187, WLAN-Group-Cipher, integer, [RFC7268]
188, WLAN-AKM-Suite, integer, [RFC7268]
189, WLAN-Group-Mgmt-Cipher, integer, [RFC7268]
190, WLAN-RF-Band, integer, [RFC7268]
191, Unassigned, ,
192-223, Experimental Use, , [RFC3575]
224-240, Implementation Specific, , [RFC3575]
241, Extended-Attribute-1, extended, [RFC6929]
241. {1-25}, Unassigned, ,
241.26, Extended-Vendor-Specific-1, evs, [RFC6929]
241. {27-240}, Unassigned, ,
241. {241-255}, Reserved, , [RFC6929]
242, Extended-Attribute-2, extended, [RFC6929]
242. {1-25}, Unassigned, ,
242.26, Extended-Vendor-Specific-2, evs, [RFC6929]
242. {27-240}, Unassigned, ,
242. {241-255}, Reserved, , [RFC6929]
243, Extended-Attribute-3, extended, [RFC6929]
243. {1-25}, Unassigned, ,
243.26, Extended-Vendor-Specific-3, evs, [RFC6929]
243. {27-240}, Unassigned, ,
243. {241-255}, Reserved, , [RFC6929]
244, Extended-Attribute-4, extended, [RFC6929]
244. {1-25}, Unassigned, ,
244.26, Extended-Vendor-Specific-4, evs, [RFC6929]
244. {27-240}, Unassigned, ,
244. {241-255}, Reserved, , [RFC6929]
245, Extended-Attribute-5, long-extended, [RFC6929]
245. {1-25}, Unassigned, ,
245.26, Extended-Vendor-Specific-5, evs, [RFC6929]
245. {27-240}, Unassigned, ,
245. {241-255}, Reserved, , [RFC6929]
246, Extended-Attribute-6, long-extended, [RFC6929]
246. {1-25}, Unassigned, ,
246.26, Extended-Vendor-Specific-6, evs, [RFC6929]
246. {27-240}, Unassigned, ,
246. {241-255}, Reserved, , [RFC6929]
247-255, Reserved, , [RFC3575]

4. Suggestions for Specifications

We suggest that these data types be used in new RADIUS specifications. Attributes can usually be completely described through their Attribute Type code, name, and data type. The use of "ASCII art" is then limited only to the definition of new data types, and complex data types.

Use of the new extended attributes [RFC6929] makes ASCII art even more problematic. An attribute can be allocated from the standard space, or from one of the extended spaces. This allocation decision is made after the specification has been accepted for publication. That allocation strongly affects the format of the attribute header, making it nearly impossible to create the correct ASCII art prior to final publication. Allocation from the different spaces also changes the value of the Length field, also making it difficult to define it correctly prior to final publication of the document.

The following fields **SHOULD** be given when defining new attributes:

Description

A description of the meaning and interpretation of the attribute.

Type

The Attribute Type code, given in the "dotted number" notation from [RFC6929]. Specifications can often leave this as "TBD", and request that IANA fill in the allocated values.

Length

A description of the length of the attribute. For attributes of variable length, a maximum length **SHOULD** be given.

Data Type

One of the named data types from the RADIUS Data Type registry.

Value

A description of any attribute-specific limitations on the values carried by the specified data type. If there are no attribute-specific limitations, then the description of this field can be omitted.

Where the values are limited to a subset of the possible range, valid range(s) **MUST** be defined.

For attributes of data type "enum", a list of enumerated values and names MUST be given, as with [RFC2865] Section 5.6.

5. Security Considerations

This specification is concerned solely with updates to IANA registries. As such, there are no security considerations with the document itself.

However, the use of inconsistent names and poorly-defined entities in a protocol is problematic. Inconsistencies in specifications can lead to security and interoperability problems in implementations. Further, having one canonical source for the definition of data types means an implementor has fewer specifications to read. The implementation work is therefore simpler, and is more likely to be correct.

The goal of this specification is to reduce ambiguities in the RADIUS protocol, which we believe will lead to more robust and more secure implementations.

6. IANA Considerations

IANA is instructed to create one new registry as described above in Section 3.1. The "TBD" text in that section should be replaced with the RFC number of this document when it is published.

IANA is instructed to update the RADIUS Attribute Type registry, as described above in Section 3.2.

IANA is instructed to require that all allocation requests in the RADIUS Attribute Type Registry contain a "data type" field. That field is required to contain one of the "data type" names contained in the RADIUS Data Type registry.

7. References

7.1. Normative References

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997.

[RFC2865]

Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3629]

Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.

[RFC6158]

DeKok, A., and Weber, G., "RADIUS Design Guidelines", RFC 6158, March 2011.

[RFC6572]

Xia, F., et al, "RADIUS Support for Proxy Mobile IPv6", RFC 6572, June 2012.

7.2. Informative References

[RFC2868]

Zorn, G., Leifer, D., Rubens, A., Shriver, J., Holdrege, M., and I. Goyret, "RADIUS Attributes for Tunnel Protocol Support", RFC 2868, June 2000.

[RFC2869]

Rigney, C., et al, "RADIUS Extensions", RFC 2869, June 2000.

[RFC3162]

Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.

[RFC6929]

DeKok, A., and Lior, A., "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, April 2013.

[PEN]

<http://www.iana.org/assignments/enterprise-numbers>

Acknowledgments

Stuff

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org

Network Working Group
INTERNET-DRAFT
Category: Experimental
<draft-ietf-radext-dtls-13.txt>
Expires: January 4, 2015
3 July 2014

Alan DeKok
FreeRADIUS

DTLS as a Transport Layer for RADIUS
draft-ietf-radext-dtls-13

Abstract

The RADIUS protocol defined in RFC 2865 has limited support for authentication and encryption of RADIUS packets. The protocol transports data in the clear, although some parts of the packets can have obfuscated content. Packets may be replayed verbatim by an attacker, and client-server authentication is based on fixed shared secrets. This document specifies how the Datagram Transport Layer Security (DTLS) protocol may be used as a fix for these problems. It also describes how implementations of this proposal can co-exist with current RADIUS systems.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on November 8, 2014

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info/>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
1.1.	Terminology	4
1.2.	Requirements Language	5
1.3.	Document Status	5
2.	Building on Existing Foundations	7
2.1.	Changes to RADIUS	7
2.2.	Similarities with RADIUS/TLS	8
2.2.1.	Changes from RADIUS/TLS to RADIUS/DTLS	8
3.	Interaction with RADIUS/UDP	9
3.1.	DTLS Port and Packet Types	10
3.2.	Server Behavior	10
4.	Client Behavior	11
5.	Session Management	12
5.1.	Server Session Management	12
5.1.1.	Session Opening and Closing	13
5.2.	Client Session Management	15
6.	Implementation Guidelines	16
6.1.	Client Implementations	16
6.2.	Server Implementations	17
7.	Diameter Considerations	18
8.	IANA Considerations	18
9.	Implementation Status	18
9.1.	Radsecproxy	18
9.2.	jradius	19
10.	Security Considerations	19
10.1.	Crypto-Agility	20
10.2.	Legacy RADIUS Security	20
10.3.	Resource Exhaustion	21
10.4.	Client-Server Authentication with DTLS	22
10.5.	Network Address Translation	23
10.6.	Wildcard Clients	24
10.7.	Session Closing	24
10.8.	Client Subsystems	24
11.	References	25
11.1.	Normative references	25
11.2.	Informative references	26

1. Introduction

The RADIUS protocol as described in [RFC2865], [RFC2866], [RFC5176], and others has traditionally used methods based on MD5 [RFC1321] for per-packet authentication and integrity checks. However, the MD5 algorithm has known weaknesses such as [MD5Attack] and [MD5Break]. As a result, some specifications such as [RFC5176] have recommended using IPsec to secure RADIUS traffic.

While RADIUS over IPsec has been widely deployed, there are difficulties with this approach. The simplest point against IPsec is that there is no straightforward way for an application to control or monitor the network security policies. That is, the requirement that the RADIUS traffic be encrypted and/or authenticated is implicit in the network configuration, and cannot be enforced by the RADIUS application.

This specification takes a different approach. We define a method for using DTLS [RFC6347] as a RADIUS transport protocol. This approach has the benefit that the RADIUS application can directly monitor and control the security policies associated with the traffic that it processes.

Another benefit is that RADIUS over DTLS continues to be a User Datagram Protocol (UDP) based protocol. The change from RADIUS/UDP is largely to add DTLS support, and make any necessary related changes to RADIUS. This allows implementations to remain UDP based, without changing to a TCP architecture.

This specification does not, however, solve all of the problems associated with RADIUS/UDP. The DTLS protocol does not add reliable or in-order transport to RADIUS. DTLS also does not support fragmentation of application-layer messages, or of the DTLS messages themselves. This specification therefore shares with traditional RADIUS the issues of order, reliability, and fragmentation. These issues are dealt with in RADIUS/TCP [RFC6613] and RADIUS/TLS [RFC6614].

1.1. Terminology

This document uses the following terms:

RADIUS/DTLS

This term is a short-hand for "RADIUS over DTLS".

RADIUS/DTLS client

This term refers both to RADIUS clients as defined in [RFC2865], and to Dynamic Authorization clients as defined in [RFC5176], that

implement RADIUS/DTLS.

RADIUS/DTLS server

This term refers both to RADIUS servers as defined in [RFC2865], and to Dynamic Authorization servers as defined in [RFC5176], that implement RADIUS/DTLS.

RADIUS/UDP

RADIUS over UDP, as defined in [RFC2865].

RADIUS/TLS

RADIUS over TLS, as defined in [RFC6614].

silently discard

This means that the implementation discards the packet without further processing.

1.2. Requirements Language

In this document, several words are used to signify the requirements of the specification. 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 [RFC2119].

1.3. Document Status

This document is an Experimental RFC.

It is one out of several approaches to address known cryptographic weaknesses of the RADIUS protocol, such as [RFC6614]. This specification does not fulfill all recommendations on a AAA transport profile as per [RFC3539]; however unlike [RFC6614], it is based on UDP, does not have head-of-line blocking issues.

If this specification is indeed selected for advancement to Standards Track, certificate verification options ([RFC6614] Section 2.3, point 2) needs to be refined.

Another experimental characteristic of this specification is the question of key management between RADIUS/DTLS peers. RADIUS/UDP only allowed for manual key management, i.e., distribution of a shared secret between a client and a server. RADIUS/DTLS allows manual distribution of long-term proofs of peer identity, by using TLS-PSK ciphersuites. RADIUS/DTLS also allows the use of X.509 certificates in a PKIX infrastructure. It remains to be seen if one of these methods will prevail or if both will find their place in real-life deployments. The authors can imagine pre-shared keys (PSK)

to be popular in small-scale deployments (Small Office, Home Office (SOHO) or isolated enterprise deployments) where scalability is not an issue and the deployment of a Certification Authority (CA) is considered too much of a hassle; however, the authors can also imagine large roaming consortia to make use of PKIX. Readers of this specification are encouraged to read the discussion of key management issues within [RFC6421] as well as [RFC4107].

It has yet to be decided whether this approach is to be chosen for Standards Track. One key aspect to judge whether the approach is usable on a large scale is by observing the uptake, usability, and operational behavior of the protocol in large-scale, real-life deployments.

2. Building on Existing Foundations

Adding DTLS as a RADIUS transport protocol requires a number of changes to systems implementing standard RADIUS. This section outlines those changes, and defines new behaviors necessary to implement DTLS.

2.1. Changes to RADIUS

The RADIUS packet format is unchanged from [RFC2865], [RFC2866], and [RFC5176]. Specifically, all of the following portions of RADIUS MUST be unchanged when using RADIUS/DTLS:

- * Packet format
- * Permitted codes
- * Request Authenticator calculation
- * Response Authenticator calculation
- * Minimum packet length
- * Maximum packet length
- * Attribute format
- * Vendor-Specific Attribute (VSA) format
- * Permitted data types
- * Calculations of dynamic attributes such as CHAP-Challenge, or Message-Authenticator.
- * Calculation of "obfuscated" attributes such as User-Password and Tunnel-Password.

In short, the application creates a RADIUS packet via the usual methods, and then instead of sending it over a UDP socket, sends the packet to a DTLS layer for encapsulation. DTLS then acts as a transport layer for RADIUS, hence the names "RADIUS/UDP" and "RADIUS/DTLS".

The requirement that RADIUS remain largely unchanged ensures the simplest possible implementation and widest interoperability of this specification.

We note that the DTLS encapsulation of RADIUS means that RADIUS packets have an additional overhead due to DTLS. Implementations MUST support sending and receiving encapsulated RADIUS packets of 4096 octets in length, with a corresponding increase in the maximum size of the encapsulated DTLS packets. This larger packet size may cause the packet to be larger than the Path MTU (PMTU), where a RADIUS/UDP packet may be smaller. See Section 5.2, below, for more discussion.

The only changes made from RADIUS/UDP to RADIUS/DTLS are the following two items:

(1) The Length checks defined in [RFC2865] Section 3 MUST use the length of the decrypted DTLS data instead of the UDP packet length. They MUST treat any decrypted DTLS data octets outside the range of the Length field as padding, and ignore it on reception.

(2) The shared secret secret used to compute the MD5 integrity checks and the attribute encryption MUST be "radius/dtls".

All other aspects of RADIUS are unchanged.

2.2. Similarities with RADIUS/TLS

While this specification can be thought of as RADIUS/TLS over UDP instead of the Transmission Control Protocol (TCP), there are some differences between the two methods. The bulk of [RFC6614] applies to this specification, so we do not repeat it here.

This section explains the differences between RADIUS/TLS and RADIUS/DTLS, as semantic "patches" to [RFC6614]. The changes are as follows:

- * We replace references to "TCP" with "UDP"
- * We replace references to "RADIUS/TLS" with "RADIUS/DTLS"
- * We replace references to "TLS" with "DTLS"

Those changes are sufficient to cover the majority of the differences between the two specifications. The next section reviews some more detailed changes from [RFC6614], giving additional commentary only where necessary.

2.2.1. Changes from RADIUS/TLS to RADIUS/DTLS

This section describes where this specification is similar to [RFC6614], and where it differs.

Section 2.1 applies to RADIUS/DTLS, with the exception that the RADIUS/DTLS port is UDP/2083.

Section 2.2 applies to RADIUS/DTLS. Servers and clients need to be preconfigured to use RADIUS/DTLS for a given endpoint.

Most of Section 2.3 applies also to RADIUS/DTLS. Item (1) should be interpreted as applying to DTLS session initiation, instead of TCP connection establishment. Item (2) applies, except for the recommendation that implementations "SHOULD" support

TLS_RSA_WITH_RC4_128_SHA. This recommendation is a historical artifact of RADIUS/TLS, and does not apply to RADIUS/DTLS. Item (3) applies to RADIUS/DTLS. Item (4) applies, except that the fixed shared secret is "radius/dtls", as described above.

Section 2.4 applies to RADIUS/DTLS. Client identities SHOULD be determined from DTLS parameters, instead of relying solely on the source IP address of the packet.

Section 2.5 does not apply to RADIUS/DTLS. The relationship between RADIUS packet codes and UDP ports in RADIUS/DTLS is unchanged from RADIUS/UDP.

Sections 3.1, 3.2, and 3.3 apply to RADIUS/DTLS.

Section 3.4 item (1) does not apply to RADIUS/DTLS. Each RADIUS packet is encapsulated in one DTLS packet, and there is no "stream" of RADIUS packets inside of a TLS session. Implementors MUST enforce the requirements of [RFC2865] Section 3 for the RADIUS Length field, using the length of the decrypted DTLS data for the checks. This check replaces the RADIUS method of using the length field from the UDP packet.

Section 3.4 items (2), (3), (4), and (5) apply to RADIUS/DTLS.

Section 4 does not apply to RADIUS/DTLS. Protocol compatibility considerations are defined in this document.

Section 6 applies to RADIUS/DTLS.

3. Interaction with RADIUS/UDP

Transitioning to DTLS is a process which needs to be done carefully. A poorly handled transition is complex for administrators, and potentially subject to security downgrade attacks. It is not sufficient to just disable RADIUS/UDP and enable RADIUS/DTLS. RADIUS has no provisions for protocol negotiation, so simply disabling RADIUS/UDP would result in timeouts, lost traffic, and network instabilities.

The end result of this specification is that nearly all RADIUS/UDP implementations should transition to using a secure alternative. In some cases, RADIUS/UDP may remain where IPsec is used as a transport, or where implementation and/or business reasons preclude a change. However, we do not recommend long-term use of RADIUS/UDP outside of isolated and secure networks.

This section describes how clients and servers should use

RADIUS/DTLS, and how it interacts with RADIUS/UDP.

3.1. DTLS Port and Packet Types

The default destination port number for RADIUS/DTLS is UDP/2083. There are no separate ports for authentication, accounting, and dynamic authorization changes. The source port is arbitrary. The text above in [RFC6614] Section 3.4 describes issues surrounding the use of one port for multiple packet types. We recognize that implementations may allow the use of RADIUS/DTLS over non-standard ports. In that case, the references to UDP/2083 in this document should be read as applying to any port used for transport of RADIUS/DTLS traffic.

3.2. Server Behavior

When a server receives packets on UDP/2083, all packets **MUST** be treated as being DTLS. RADIUS/UDP packets **MUST NOT** be accepted on this port.

Servers **MUST NOT** accept DTLS packets on the old RADIUS/UDP ports. Early drafts of this specification permitted this behavior. It is forbidden here, as it depended on behavior in DTLS which may change without notice.

Servers **MUST** authenticate clients. RADIUS is designed to be used by mutually trusted systems. Allowing anonymous clients would ensure privacy for RADIUS/DTLS traffic, but would negate all other security aspects of the protocol.

As RADIUS has no provisions for capability signalling, there is no way for a server to indicate to a client that it should transition to using DTLS. This action has to be taken by the administrators of the two systems, using a method other than RADIUS. This method will likely be out of band, or manual configuration.

Some servers maintain a list of allowed clients per destination port. Others maintain a global list of clients, which are permitted to send packets to any port. Where a client can send packets to multiple ports, the server **MUST** maintain a "DTLS Required" flag per client.

This flag indicates whether or not the client is required to use DTLS. When set, the flag indicates that the only traffic accepted from the client is over UDP/2083. When packets are received from a client on non-DTLS ports, for which DTLS is required, the server **MUST** silently discard these packets, as there is no RADIUS/UDP shared secret available.

This flag will often be set by an administrator. However, if a server receives DTLS traffic from a client, it SHOULD notify the administrator that DTLS is available for that client. It MAY mark the client as "DTLS Required".

It is RECOMMENDED that servers support the following perfect forward secrecy (PFS) cipher suites:

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Allowing RADIUS/UDP and RADIUS/DTLS from the same client exposes the traffic to downbidding attacks, and is NOT RECOMMENDED.

4. Client Behavior

When a client sends packets to the assigned RADIUS/DTLS port, all packets MUST be DTLS. RADIUS/UDP packets MUST NOT be sent to this port.

Clients MUST authenticate themselves to servers, via credentials which are unique to each client.

It is RECOMMENDED that clients support the following PFS cipher suites:

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

RADIUS/DTLS clients SHOULD NOT probe servers to see if they support DTLS transport. Instead, clients SHOULD use DTLS as a transport layer only when administratively configured. If a client is configured to use DTLS and the server appears to be unresponsive, the client MUST NOT fall back to using RADIUS/UDP. Instead, the client should treat the server as being down.

RADIUS clients often had multiple independent RADIUS implementations and/or processes that originate packets. This practice was simple to implement, but the result is that each independent subsystem must independently discover network issues or server failures. It is therefore RECOMMENDED that clients with multiple internal RADIUS sources use a local proxy as described in Section 6.1, below.

Clients may implement "pools" of servers for fail-over or load-balancing. These pools SHOULD NOT mix RADIUS/UDP and RADIUS/DTLS servers.

5. Session Management

Where [RFC6614] can rely on the TCP state machine to perform session tracking, this specification cannot. As a result, implementations of this specification may need to perform session management of the DTLS session in the application layer. This section describes logically how this tracking is done. Implementations may choose to use the method described here, or another, equivalent method.

We note that [RFC5080] Section 2.2.2 already mandates a duplicate detection cache. The session tracking described below can be seen as an extension of that cache, where entries contain DTLS sessions instead of RADIUS/UDP packets.

[RFC5080] section 2.2.2 describes how duplicate RADIUS/UDP requests result in the retransmission of a previously cached RADIUS/UDP response. Due to DTLS sequence window requirements, a server **MUST** NOT retransmit a previously sent DTLS packet. Instead, it should cache the RADIUS response packet, and re-process it through DTLS to create a new RADIUS/DTLS packet, every time it is necessary to retransmit a RADIUS response.

5.1. Server Session Management

A RADIUS/DTLS server **MUST** track ongoing DTLS sessions for each based the following 4-tuple:

- * source IP address
- * source port
- * destination IP address
- * destination port

Note that this 4-tuple is independent of IP address version (IPv4 or IPv6).

Each 4-tuple points to a unique session entry, which usually contain the following information:

DTLS Session

Any information required to maintain and manage the DTLS session.

Last Traffic

A variable containing a timestamp which indicates when this session last received valid traffic. If "Last Traffic" is not used, this variable may not exist.

DTLS Data

An implementation-specific variable which may contain information

about the active DTLS session. This variable may be empty or non-existent.

This data will typically contain information such as idle timeouts, session lifetimes, and other implementation-specific data.

5.1.1. Session Opening and Closing

Session tracking is subject to Denial of Service (DoS) attacks due to the ability of an attacker to forge UDP traffic. RADIUS/DTLS servers SHOULD use the stateless cookie tracking technique described in [RFC6347] Section 4.2.1. DTLS sessions SHOULD NOT be tracked until a ClientHello packet has been received with an appropriate Cookie value. Server implementation SHOULD have a way of tracking partially setup DTLS sessions. Servers MUST limit both the number and impact on resources of partial sessions.

Sessions (both 4-tuple and entry) MUST be deleted when a TLS Closure Alert ([RFC5246] Section 7.2.1) or a fatal TLS Error Alert ([RFC5246] Section 7.2.2) is received. When a session is deleted due to it failing security requirements, the DTLS session MUST be closed, and any TLS session resumption parameters for that session MUST be discarded, and all tracking information MUST be deleted.

Sessions MUST also be deleted when a RADIUS packet fails validation due to a packet being malformed, or when it has an invalid Message-Authenticator, or invalid Request Authenticator. There are other cases when the specifications require that a packet received via a DTLS session be "silently discarded". In those cases, implementations MAY delete the underlying session as described above. There are few reasons to communicate with a NAS which is not implementing RADIUS.

A session MUST be deleted when non-RADIUS traffic is received over it. This specification is for RADIUS, and there is no reason to allow non-RADIUS traffic over a RADIUS/DTLS session. A session MUST be deleted when RADIUS traffic fails to pass security checks. There is no reason to permit insecure networks. A session SHOULD NOT be deleted when a well-formed, but "unexpected" RADIUS packet is received over it. Future specifications may extend RADIUS/DTLS, and we do not want to forbid those specifications.

The goal of the above requirements is to ensure security, while maintaining flexibility. Any security related issue causes the connection to be closed. After the security restrictions have been applied, any unexpected traffic may be safely ignored, as it cannot cause a security issue. There is no need to close the session for unexpected but valid traffic, and the session can safely remain open.

Once a DTLS session is established, a RADIUS/DTLS server SHOULD use DTLS Heartbeats [RFC6520] to determine connectivity between the two servers. A server SHOULD also use watchdog packets from the client to determine that the session is still active.

As UDP does not guarantee delivery of messages, RADIUS/DTLS servers which do not implement an application-layer watchdog MUST also maintain a "Last Traffic" timestamp per DTLS session. The granularity of this timestamp is not critical, and could be limited to one second intervals. The timestamp SHOULD be updated on reception of a valid RADIUS/DTLS packet, or a DTLS Heartbeat, but no more than once per interval. The timestamp MUST NOT be updated in other situations.

When a session has not received a packet for a period of time, it is labelled "idle". The server SHOULD delete idle DTLS sessions after an "idle timeout". The server MAY cache the TLS session parameters, in order to provide for fast session resumption.

This session "idle timeout" SHOULD be exposed to the administrator as a configurable setting. It SHOULD NOT be set to less than 60 seconds, and SHOULD NOT be set to more than 600 seconds (10 minutes). The minimum value useful value for this timer is determined by the application-layer watchdog mechanism defined in the following section.

RADIUS/DTLS servers SHOULD also monitor the total number of open sessions. They SHOULD have a "maximum sessions" setting exposed to administrators as a configurable parameter. When this maximum is reached and a new session is started, the server MUST either drop an old session in order to open the new one, or instead not create a new session.

RADIUS/DTLS servers SHOULD implement session resumption, preferably stateless session resumption as given in [RFC5077]. This practice lowers the time and effort required to start a DTLS session with a client, and increases network responsiveness.

Since UDP is stateless, the potential exists for the client to initiate a new DTLS session using a particular 4-tuple, before the server has closed the old session. For security reasons, the server MUST keep the old session active until either it has received secure notification from the client that the session is closed, or when the server decides to close the session based on idle timeouts. Taking any other action would permit unauthenticated clients to perform a DoS attack, by re-using a 4-tuple, and thus causing the server to close an active (and authenticated) DTLS session.

As a result, servers MUST ignore any attempts to re-use an existing 4-tuple from an active session. This requirement can likely be reached by simply processing the packet through the existing session, as with any other packet received via that 4-tuple. Non-compliant, or unexpected packets will be ignored by the DTLS layer.

The above requirement is mitigated by the suggestion in Section 6.1, below, that the client use a local proxy for all RADIUS traffic. That proxy can then track the ports which it uses, and ensure that re-use of 4-tuples is avoided. The exact process by which this tracking is done is outside of the scope of this document.

5.2. Client Session Management

Clients SHOULD use PMTU discovery [RFC6520] to determine the PMTU between the client and server, prior to sending any RADIUS traffic. Once a DTLS session is established, a RADIUS/DTLS client SHOULD use DTLS Heartbeats [RFC6520] to determine connectivity between the two systems. RADIUS/DTLS clients SHOULD also use the application-layer watchdog algorithm defined in [RFC3539] to determine server responsiveness. The Status-Server packet defined in [RFC5997] SHOULD be used as the "watchdog packet" in any application-layer watchdog algorithm.

RADIUS/DTLS clients SHOULD pro-actively close sessions when they have been idle for a period of time. Clients SHOULD close a session when the DTLS Heartbeat algorithm indicates that the session is no longer active. Clients SHOULD close a session when no traffic other than watchdog packets and (possibly) watchdog responses have been sent for three watchdog timeouts. This behavior ensures that clients do not waste resources on the server by causing it to track idle sessions.

When client fails to implement both DTLS heartbeats and watchdog packets, it has no way of knowing that a DTLS session has been closed. There is therefore the possibility that the server closes the session without the client knowing. When that happens, the client may later transmit packets in a session, and those packets will be ignored by the server. The client is then forced to time out those packets and then the session, leading to delays and network instabilities.

For these reasons, it is RECOMMENDED that all DTLS sessions are configured to use DTLS heartbeats and/or watchdog packets.

DTLS sessions MUST also be deleted when a RADIUS packet fails validation due to a packet being malformed, or when it has an invalid Message-Authenticator, or invalid Response Authenticator. There are other cases when the specifications require that a packet received

via a DTLS session be "silently discarded". In those cases, implementations MAY delete the underlying DTLS session.

RADIUS/DTLS clients should not send both RADIUS/UDP and RADIUS/DTLS packets to different servers from the same source socket. This practice causes increased complexity in the client application, and increases the potential for security breaches due to implementation issues.

RADIUS/DTLS clients SHOULD implement session resumption, preferably stateless session resumption as given in [RFC5077]. This practice lowers the time and effort required to start a DTLS session with a server, and increases network responsiveness.

6. Implementation Guidelines

The text above describes the protocol. In this section, we give additional implementation guidelines. These guidelines are not part of the protocol, but may help implementors create simple, secure, and inter-operable implementations.

Where a TLS pre-shared key (PSK) method is used, implementations MUST support keys of at least 16 octets in length. Implementations SHOULD support key lengths of 32 octets, and SHOULD allow for longer keys. The key data MUST be capable of being any value (0 through 255, inclusive). Implementations MUST NOT limit themselves to using textual keys. It is RECOMMENDED that the administration interface allows for the keys to be entered as humanly readable strings in hex format.

When creating keys for use with PSK cipher suites, it is RECOMMENDED that keys be derived from a cryptographically secure pseudo-random number generator (CSPRNG) instead of administrators inventing keys on their own. If managing keys is too complicated, a certificate-based TLS method SHOULD be used instead.

6.1. Client Implementations

RADIUS/DTLS clients should use connected sockets where possible. Use of connected sockets means that the underlying kernel tracks the sessions, so that the client subsystem does not need to manage multiple sessions on one socket.

RADIUS/DTLS clients should use a single source (IP + port) when sending packets to a particular RADIUS/DTLS server. Doing so minimizes the number of DTLS session setups. It also ensures that information about the home server state is discovered only once.

In practice, this means that RADIUS/DTLS clients with multiple internal RADIUS sources should use a local proxy which arbitrates all RADIUS traffic between the client and all servers. The proxy should accept traffic only from the authorized subsystems on the client machine, and should proxy that traffic to known servers. Each authorized subsystem should include an attribute which uniquely identifies that subsystem to the proxy, so that the proxy can apply origin-specific proxy rules and security policies. We suggest using NAS-Identifier for this purpose.

The local proxy should be able to interact with multiple servers at the same time. There is no requirement that each server have its own unique proxy on the client, as that would be inefficient.

The suggestion to use a local proxy means that there is only one process which discovers network and/or connectivity issues with a server. If each client subsystem communicated directly with a server, issues with that server would have to be discovered independently by each subsystem. The side effect would be increased delays in re-routing traffic, error reporting, and network instabilities.

Each client subsystem can include a subsystem-specific NAS-Identifier in each request. The format of this attribute is implementation-specific. The proxy should verify that the request originated from the local system, ideally via a loopback address. The proxy MUST then re-write any subsystem-specific NAS-Identifier to a NAS-Identifier which identifies the client as a whole. Or, remove NAS-Identifier entirely and replace it with NAS-IP-Address or NAS-IPv6-Address.

In traditional RADIUS, the cost to set up a new "session" between a client and server was minimal. The client subsystem could simply open a port, send a packet, wait for the response, and then close the port. With RADIUS/DTLS, the connection setup is significantly more expensive. In addition, there may be a requirement to use DTLS in order to communicate with a server, as RADIUS/UDP may not be supported by that server. The knowledge of what protocol to use is best managed by a dedicated RADIUS subsystem, rather than by each individual subsystem on the client.

6.2. Server Implementations

RADIUS/DTLS servers should not use connected sockets to read DTLS packets from a client. This recommendation is because a connected UDP socket will accept packets only from one source IP address and port. This limitation would prevent the server from accepting packets from multiple clients on the same port.

7. Diameter Considerations

This specification defines a transport layer for RADIUS. It makes no other changes to the RADIUS protocol. As a result, there are no Diameter considerations.

8. IANA Considerations

No new RADIUS attributes or packet codes are defined. IANA is requested to update the "Service Name and Transport Protocol Port Number Registry". The entry corresponding to port service name "radsec", port number "2083", and transport protocol "UDP" should be updated as follows:

- o Assignee: change "Mike McCauley" to "IESG".
- o Contact: change "Mike McCauley" to "IETF Chair"
- o Reference: Add this document as a reference
- o Assignment Notes: add the text "The UDP port 2083 was already previously assigned by IANA for "RadSec", an early implementation of RADIUS/TLS, prior to issuance of this RFC."

9. Implementation Status

This section records the status of known implementations of RADIUS/DTLS at the time of posting of this Internet- Draft, and is based on a proposal described in [RFC6982].

The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

9.1. Radsecproxy

Organization: Radsecproxy

URL: <https://software.uninett.no/radsecproxy/>

Maturity: Widely-used software based on early drafts of this document.
The use of the DTLS functionality is not clear.

Coverage: The bulk of this specification is implemented, based on earlier versions of this document. Exact revisions which were implemented are unknown.

Licensing: Freely distributable with acknowledgement

Implementation experience: No comments from implementors.

9.2. jradius

Organization: Coova

URL: <http://www.coova.org/JRadius/RadSec>

Maturity: Production software based on early drafts of this document.
The use of the DTLS functionality is not clear.

Coverage: The bulk of this specification is implemented, based on earlier versions of this document. Exact revisions which were implemented are unknown.

Licensing: Freely distributable with requirement to redistribute source.

Implementation experience: No comments from implementors.

10. Security Considerations

The bulk of this specification is devoted to discussing security considerations related to RADIUS. However, we discuss a few additional issues here.

This specification relies on the existing DTLS, RADIUS/UDP, and RADIUS/TLS specifications. As a result, all security considerations for DTLS apply to the DTLS portion of RADIUS/DTLS. Similarly, the TLS and RADIUS security issues discussed in [RFC6614] also apply to this specification. Most of the security considerations for RADIUS apply to the RADIUS portion of the specification.

However, many security considerations raised in the RADIUS documents are related to RADIUS encryption and authorization. Those issues are largely mitigated when DTLS is used as a transport method. The issues that are not mitigated by this specification are related to the RADIUS packet format and handling, which is unchanged in this specification.

This specification also suggests that implementations use a session tracking table. This table is an extension of the duplicate detection cache mandated in [RFC5080] Section 2.2.2. The changes given here are that DTLS-specific information is tracked for each table entry. Section 5.1.1, above, describes steps to mitigate any

DoS issues which result from tracking additional information.

The fixed shared secret given above in Section 2.2.1 is acceptable only when DTLS is used with a non-null encryption method. When a DTLS session uses a null encryption method due to misconfiguration or implementation error, all of the RADIUS traffic will be readable by an observer. Implementations therefore MUST NOT use null encryption methods for RADIUS/DTLS.

For systems which perform protocol-based firewalling and/or filtering, it is RECOMMENDED that they be configured to permit only DTLS over the RADIUS/DTLS port.

10.1. Crypto-Agility

Section 4.2 of [RFC6421] makes a number of recommendations about security properties of new RADIUS proposals. All of those recommendations are satisfied by using DTLS as the transport layer.

Section 4.3 of [RFC6421] makes a number of recommendations about backwards compatibility with RADIUS. Section 3, above, addresses these concerns in detail.

Section 4.4 of [RFC6421] recommends that change control be ceded to the IETF, and that interoperability is possible. Both requirements are satisfied.

Section 4.5 of [RFC6421] requires that the new security methods apply to all packet types. This requirement is satisfied by allowing DTLS to be used for all RADIUS traffic. In addition, Section 3, above, addresses concerns about documenting the transition from legacy RADIUS to crypto-agile RADIUS.

Section 4.6 of [RFC6421] requires automated key management. This requirement is satisfied by using DTLS key management.

10.2. Legacy RADIUS Security

We reiterate here the poor security of the legacy RADIUS protocol. We suggest that RADIUS clients and servers implement either this specification, or [RFC6614]. New attacks on MD5 have appeared over the past few years, and there is a distinct possibility that MD5 may be completely broken in the near future. Such a break would mean that RADIUS/UDP was completely insecure.

The existence of fast and cheap attacks on MD5 could result in a loss of all network security which depends on RADIUS. Attackers could obtain user passwords, and possibly gain complete network access. We

cannot overstate the disastrous consequences of a successful attack on RADIUS.

We also caution implementors (especially client implementors) about using RADIUS/DTLS. It may be tempting to use the shared secret as the basis for a TLS pre-shared key (PSK) method, and to leave the user interface otherwise unchanged. This practice **MUST NOT** be used. The administrator **MUST** be given the option to use DTLS. Any shared secret used for RADIUS/UDP **MUST NOT** be used for DTLS. Re-using a shared secret between RADIUS/UDP and RADIUS/DTLS would negate all of the benefits found by using DTLS.

RADIUS/DTLS client implementors **MUST** expose a configuration that allows the administrator to choose the cipher suite. Where certificates are used, RADIUS/DTLS client implementors **MUST** expose a configuration which allows an administrator to configure all certificates necessary for certificate-based authentication. These certificates include client, server, and root certificates.

TLS-PSK methods are susceptible to dictionary attacks. Section 6, above, recommends deriving TLS-PSK keys from a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG), which makes dictionary attacks significantly more difficult. Servers **SHOULD** track failed client connections by TLS-PSK ID, and block TLS-PSK IDs which seem to be attempting brute-force searches of the keyspace.

The historic RADIUS practice of using shared secrets (here, PSKs) that are minor variations of words is **NOT RECOMMENDED**, as it would negate all of the security of DTLS.

10.3. Resource Exhaustion

The use of DTLS allows DoS attacks, and resource exhaustion attacks which were not possible in RADIUS/UDP. These attacks are the similar to those described in [RFC6614] Section 6, for TCP.

Session tracking as described in Section 5.1 can result in resource exhaustion. Servers **MUST** therefore limit the absolute number of sessions that they track. When the total number of sessions tracked is going to exceed the configured limit, servers **MAY** free up resources by closing the session which has been idle for the longest time. Doing so may free up idle resources which then allow the server to accept a new session.

Servers **MUST** limit the number of partially open DTLS sessions. These limits **SHOULD** be exposed to the administrator as configurable settings.

10.4. Client-Server Authentication with DTLS

We expect that the initial deployment of DTLS will be follow the RADIUS/UDP model of statically configured client-server relationships. The specification for dynamic discovery of RADIUS servers is under development, so we will not address that here.

Static configuration of client-server relationships for RADIUS/UDP means that a client has a fixed IP address for a server, and a shared secret used to authenticate traffic sent to that address. The server in turn has a fixed IP address for a client, and a shared secret used to authenticate traffic from that address. This model needs to be extended for RADIUS/DTLS.

Instead of a shared secret, TLS credentials **MUST** be used by each party to authenticate the other. The issue of identity is more problematic. As with RADIUS/UDP, IP addresses may be used as a key to determine the authentication credentials which a client will present to a server, or which credentials a server will accept from a client. This is the fixed IP address model of RADIUS/UDP, with the shared secret replaced by TLS credentials.

There are, however, additional considerations with RADIUS/DTLS. When a client is configured with a host name for a server, the server may present to the client a certificate containing a host name. The client **MUST** then verify that the host names match. Any mismatch is a security violation, and the connection **MUST** be closed.

A RADIUS/DTLS server **MAY** be configured with a "wildcard" IP address match for clients, instead of a unique fixed IP address for each client. In that case, clients **MUST** be individually configured with a unique certificate. When the server receives a connection from a client, it **MUST** determine client identity from the client certificate, and **MUST** authenticate (or not) the client based on that certificate. See [RFC6614] Section 2.4 for a discussion of how to match a certificate to a client identity.

However, servers **SHOULD** use IP address filtering to minimize the possibility of attacks. That is, they **SHOULD** permit clients only from a limited IP address range or ranges. They **SHOULD** silently discard all traffic from outside of those ranges.

Since the client-server relationship is static, the authentication credentials for that relationship must also be statically configured. That is, a client connecting to a DTLS server **SHOULD** be pre-configured with the servers credentials (e.g. PSK or certificate). If the server fails to present the correct credentials, the DTLS session **MUST** be closed. Each server **SHOULD** be preconfigured with

sufficient information to authenticate connecting clients.

The requirement for clients to be individually configured with a unique certificate can be met by using a private Certificate Authority (CA) for certificates used in RADIUS/DTLS environments. If a client were configured to use a public CA, then it could accept as valid any server which has a certificate signed by that CA. While the traffic would be secure from third-party observers, the server would, however, have unrestricted access to all of the RADIUS traffic, including all user credentials and passwords.

Therefore, clients SHOULD NOT be pre-configured with a list of known public CAs by the vendor or manufacturer. Instead, the clients SHOULD start off with an empty CA list. The addition of a CA SHOULD be done only when manually configured by an administrator.

This scenario is the opposite of web browsers, where they are pre-configured with many known CAs. The goal there is security from third-party observers, but also the ability to communicate with any unknown site which presents a signed certificate. In contrast, the goal of RADIUS/DTLS is both security from third-party observers, and the ability to communicate with only a small set of well-known servers.

This requirement does not prevent clients from using hostnames instead of IP addresses for locating a particular server. Instead, it means that the credentials for that server should be preconfigured on the client, and associated with that hostname. This requirement does suggest that in the absence of a specification for dynamic discovery, clients SHOULD use only those servers which have been manually configured by an administrator.

10.5. Network Address Translation

Network Address Translation (NAT) is fundamentally incompatible with RADIUS/UDP. RADIUS/UDP uses the source IP address to determine the shared secret for the client, and NAT hides many clients behind one source IP address. As a result, RADIUS/UDP clients can not be located behind a NAT gateway.

In addition, port re-use on a NAT gateway means that packets from different clients may appear to come from the same source port on the NAT. That is, a RADIUS server may receive a RADIUS/DTLS packet from one source IP/port combination, followed by the reception of a RADIUS/UDP packet from that same source IP/port combination. If this behavior is allowed, then the server would have an inconsistent view of the clients security profile, allowing an attacker to choose the most insecure method.

If more than one client is located behind a NAT gateway, then every client behind the NAT MUST use a secure transport such as TLS or DTLS. As discussed below, a method for uniquely identifying each client MUST be used.

10.6. Wildcard Clients

Some RADIUS server implementations allow for "wildcard" clients. That is, clients with an IPv4 netmask of other than 32, or an IPv6 netmask of other than 128. That practice is not recommended for RADIUS/UDP, as it means multiple clients will use the same shared secret.

The use of RADIUS/DTLS can allow for the safe usage of wildcards. When RADIUS/DTLS is used with wildcards, clients MUST be uniquely identified using TLS parameters, and any certificate or PSK used MUST be unique to each client.

10.7. Session Closing

Section 5.1.1, above, requires that DTLS sessions be closed when the transported RADIUS packets are malformed, or fail the authenticator checks. The reason is that the session is expected to be used for transport of RADIUS packets only.

Any non-RADIUS traffic on that session means the other party is misbehaving, and is a potential security risk. Similarly, any RADIUS traffic failing authentication vector or Message-Authenticator validation means that two parties do not have a common shared secret, and the session is therefore unauthenticated and insecure.

We wish to avoid the situation where a third party can send well-formed RADIUS packets which cause a DTLS session to close. Therefore, in other situations, the session SHOULD remain open in the face of non-conformant packets.

10.8. Client Subsystems

Many traditional clients treat RADIUS as subsystem-specific. That is, each subsystem on the client has its own RADIUS implementation and configuration. These independent implementations work for simple systems, but break down for RADIUS when multiple servers, fail-over, and load-balancing are required. They have even worse issues when DTLS is enabled.

As noted in Section 6.1, above, clients SHOULD use a local proxy which arbitrates all RADIUS traffic between the client and all servers. This proxy will encapsulate all knowledge about servers,

including security policies, fail-over, and load-balancing. All client subsystems SHOULD communicate with this local proxy, ideally over a loopback address. The requirements on using strong shared secrets still apply.

The benefit of this configuration is that there is one place in the client which arbitrates all RADIUS traffic. Subsystems which do not implement DTLS can remain unaware of DTLS. DTLS sessions opened by the proxy can remain open for long periods of time, even when client subsystems are restarted. The proxy can do RADIUS/UDP to some servers, and RADIUS/DTLS to others.

Delegation of responsibilities and separation of tasks are important security principles. By moving all RADIUS/DTLS knowledge to a DTLS-aware proxy, security analysis becomes simpler, and enforcement of correct security becomes easier.

11. References

11.1. Normative references

[RFC2865]

Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3539]

Aboba, B. et al., "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, June 2003.

[RFC5077]

Salowey, J, et al., "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008

[RFC5080]

Nelson, D. and DeKok, A, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.

[RFC5246]

Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5997]

DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", RFC 5997, August 2010.

[RFC6347]
Rescorla E., and Modadugu, N., "Datagram Transport Layer Security",
RFC 6347, April 2006.

[RFC6520]
Seggelmann, R., et al., "Transport Layer Security (TLS) and Datagram
Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520,
February 2012.

[RFC6613]
DeKok, A., "RADIUS over TCP", RFFC 6613, May 2012

[RFC6614]
Winter, S., et. al., "TLS encryption for RADIUS over TCP", RFFC
6614, May 2012

11.2. Informative references

[RFC1321]
Rivest, R. and S. Dusse, "The MD5 Message-Digest Algorithm", RFC
1321, April 1992.

[RFC2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement
Levels", RFC 2119, March, 1997.

[RFC2866]
Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

[RFC4107]
Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key
Management", BCP 107, RFC 4107, June 2005.

[RFC5176]
Chiba, M. et al., "Dynamic Authorization Extensions to Remote
Authentication Dial In User Service (RADIUS)", RFC 5176, January
2008.

[RFC6421]
Nelson, D. (Ed), "Crypto-Agility Requirements for Remote
Authentication Dial-In User Service (RADIUS)", RFC 6421, November
2011.

[RFC6982]
Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code:
The Implementation Status Section", RFC 6982, July 2013.

[MD5Attack]

Dobbertin, H., "The Status of MD5 After a Recent Attack",
CryptoBytes Vol.2 No.2, Summer 1996.

[MD5Break]

Wang, Xiaoyun and Yu, Hongbo, "How to Break MD5 and Other Hash
Functions", EUROCRYPT. ISBN 3-540-25910-4, 2005.

Acknowledgments

Parts of the text in Section 3 defining the Request and Response
Authenticators were taken with minor edits from [RFC2865] Section 3.

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project
<http://freeradius.org>

Email: aland@freeradius.org

RADIUS Extensions Working Group
Internet-Draft
Intended status: Experimental
Expires: November 1, 2015

S. Winter
RESTENA
M. McCauley
AirSpayce
April 30, 2015

NAI-based Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS
draft-ietf-radext-dynamic-discovery-15

Abstract

This document specifies a means to find authoritative RADIUS servers for a given realm. It is used in conjunction with either RADIUS/TLS and RADIUS/DTLS.

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 <http://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 November 1, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	5
1.2. Terminology	6
1.3. Document Status	6
2. Definitions	7
2.1. DNS Resource Record (RR) definition	7
2.1.1. S-NAPTR	7
2.1.2. SRV	11
2.1.3. Optional name mangling	12
2.2. Definition of the X.509 certificate property SubjectAltName:otherName:NAIRealm	13
3. DNS-based NAPTR/SRV Peer Discovery	15
3.1. Applicability	15
3.2. Configuration Variables	16
3.3. Terms	16
3.4. Realm to RADIUS server resolution algorithm	17
3.4.1. Input	17
3.4.2. Output	18
3.4.3. Algorithm	18
3.4.4. Validity of results	19
3.4.5. Delay considerations	20
3.4.6. Example	21
4. Operations and Manageability Considerations	23
5. Security Considerations	24
6. Privacy Considerations	25
7. IANA Considerations	26
8. References	28
8.1. Normative References	28
8.2. Informative References	29
Appendix A. Appendix A: ASN.1 Syntax of NAIRealm	30

1. Introduction

RADIUS in all its current transport variants (RADIUS/UDP, RADIUS/TCP, RADIUS/TLS, RADIUS/DTLS) requires manual configuration of all peers (clients, servers).

Where more than one administrative entity collaborates for RADIUS authentication of their respective customers (a "roaming consortium"), the Network Access Identifier (NAI) [I-D.ietf-radext-nai] is the suggested way of differentiating users between those entities; the part of a username to the right of the @ delimiter in an NAI is called the user's "realm". Where many realms and RADIUS forwarding servers are in use, the number of realms to be forwarded and the corresponding number of servers to configure may be significant. Where new realms with new servers are added or details

of existing servers change on a regular basis, maintaining a single monolithic configuration file for all these details may prove too cumbersome to be useful.

Furthermore, in cases where a roaming consortium consists of independently working branches (e.g. departments, national subsidiaries), each with their own forwarding servers, and who add or change their realm lists at their own discretion, there is additional complexity in synchronising the changed data across all branches.

Where realms can be partitioned (e.g. according to their top-level domain ending), forwarding of requests can be realised with a hierarchy of RADIUS servers, all serving their partition of the realm space. Figure 1 show an example of this hierarchical routing.

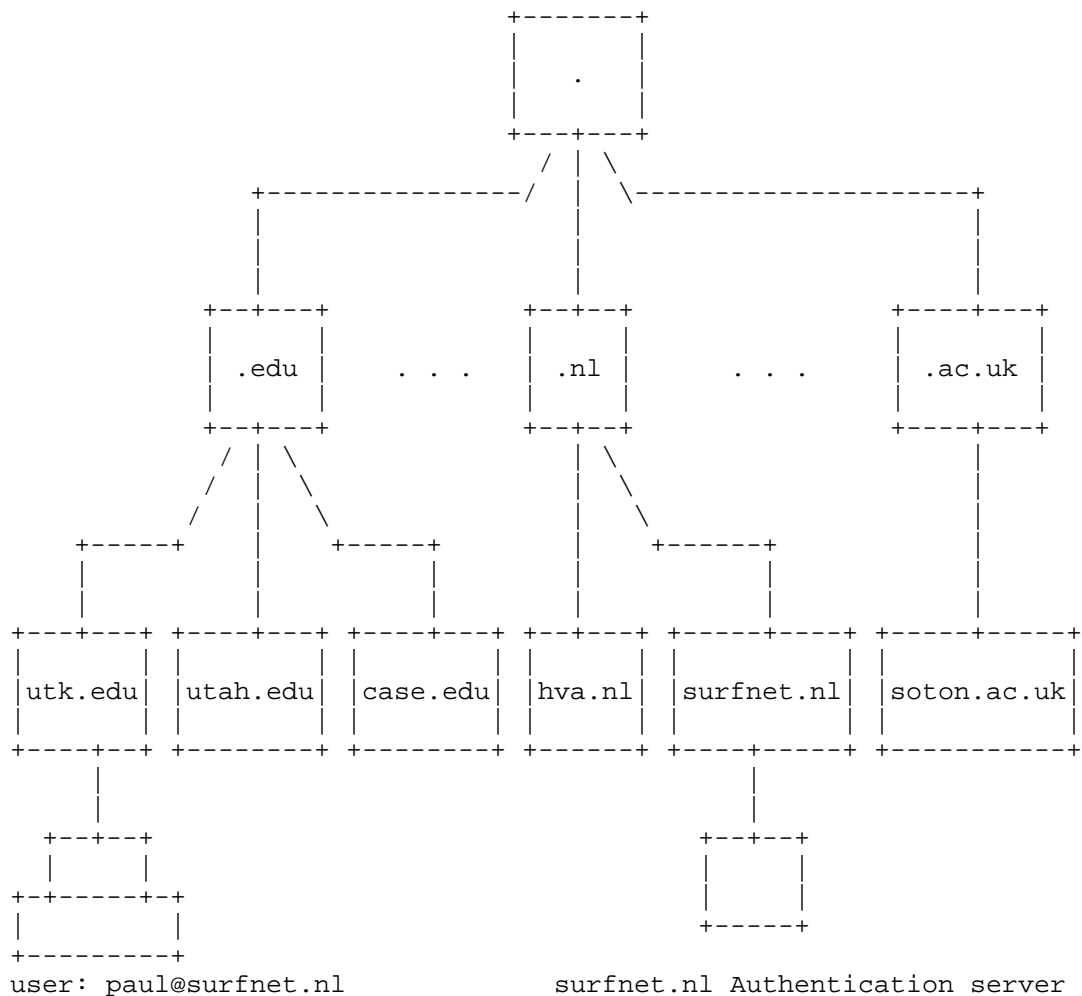


Figure 1: RADIUS hierarchy based on Top-Level Domain partitioning

However, such partitioning is not always possible. As an example, in one real-life deployment, the administrative boundaries and RADIUS forwarding servers are organised along country borders, but generic top-level domains such as .edu do not map to this choice of boundaries (see [I-D.wierenga-ietf-eduroam] for details). These situations can benefit significantly from a distributed mechanism for storing realm and server reachability information. This document describes one such mechanism: storage of realm-to-server mappings in DNS; realm-based request forwarding can then be realised without a static hierarchy such as in the following figure:

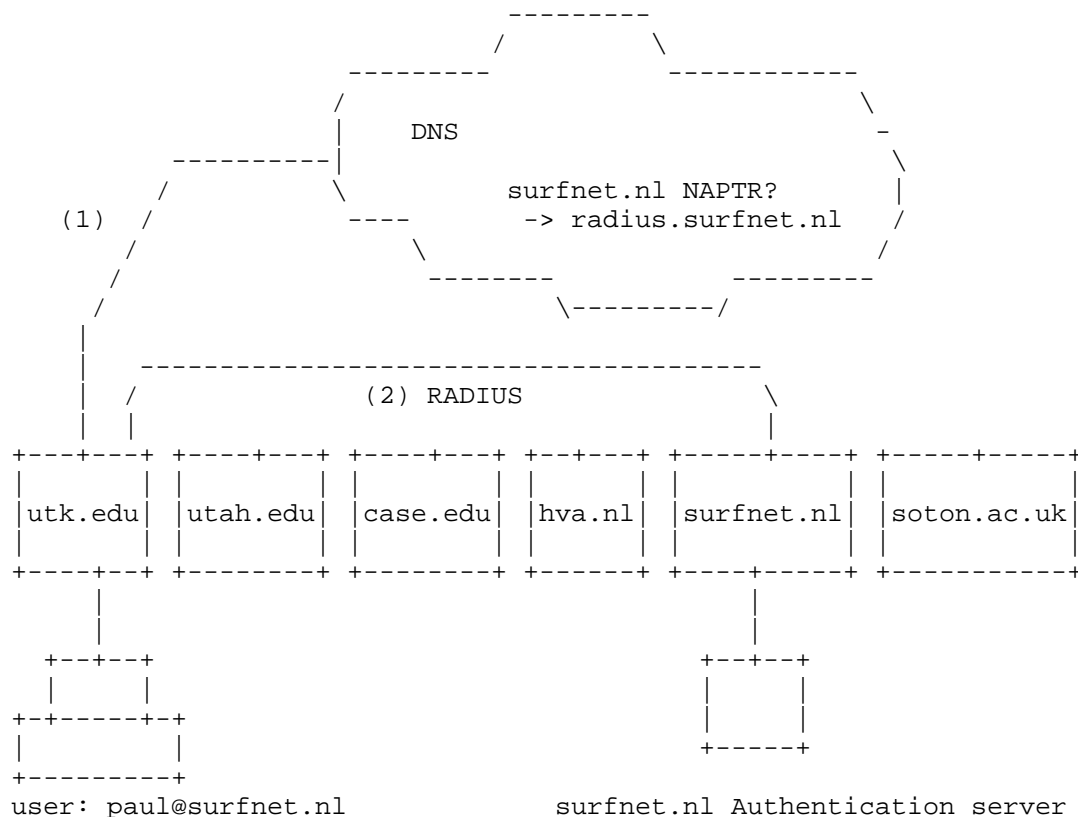


Figure 2: RADIUS hierarchy based on Top-Level Domain partitioning

This document also specifies various approaches for verifying that server information which was retrieved from DNS was from an authorised party; e.g. an organisation which is not at all part of a given roaming consortium may alter its own DNS records to yield a result for its own realm.

1.1. Requirements Language

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

1.2. Terminology

RADIUS/TLS Client: a RADIUS/TLS [RFC6614] instance which initiates a new connection.

RADIUS/TLS Server: a RADIUS/TLS [RFC6614] instance which listens on a RADIUS/TLS port and accepts new connections

RADIUS/TLS node: a RADIUS/TLS client or server

[I-D.ietf-radext-nai] defines the terms NAI, realm, consortium.

1.3. Document Status

This document is an Experimental RFC.

The communities expected to use this document are roaming consortia whose authentication services are based on the RADIUS protocol.

The duration of the experiment is undetermined; as soon as enough experience is collected on the choice points mentioned below, it is expected to be obsoleted by a standards-track version of the protocol which trims down the choice points.

If that removal of choice points obsoletes tags or service names as defined in this document and allocated by IANA, these items will be returned to IANA as per the provisions in [RFC6335].

The document provides a discovery mechanism for RADIUS which is very similar to the approach that is taken with the Diameter protocol [RFC6733]. As such, the basic approach (using Naming Authority Pointer (NAPTR) records in DNS domains which match NAI realms) is not of very experimental nature.

However, the document offers a few choice points and extensions which go beyond the provisions for Diameter. The list of major additions/deviations is

- o provisions for determining the authority of a server to act for users of a realm (declared out of scope for Diameter)
- o much more in-depth guidance on DNS regarding timeouts, failure conditions, alteration of Time-To-Live (TTL) information than the Diameter counterpart
- o a partially correct routing error detection during DNS lookups

2. Definitions

2.1. DNS Resource Record (RR) definition

DNS definitions of RADIUS/TLS servers can be either S-NAPTR records (see [RFC3958]) or Service Record (SRV) records. When both are defined, the resolution algorithm prefers S-NAPTR results (see Section 3.4 below).

2.1.1. S-NAPTR

2.1.1.1. Registration of Application Service and Protocol Tags

This specification defines three S-NAPTR service tags:

Service Tag	Use
aaa+auth	RADIUS Authentication, i.e. traffic as defined in [RFC2865]
aaa+acct	RADIUS Accounting, i.e. traffic as defined in [RFC2866]
aaa+dynauth	RADIUS Dynamic Authorisation, i.e. traffic as defined in [RFC5176]

Figure 3: List of Service Tags

This specification defines two S-NAPTR protocol tags:

Protocol Tag	Use
radius.tls.tcp	RADIUS transported over TLS as defined in [RFC6614]
radius.dtls.udp	RADIUS transported over DTLS as defined in [RFC7360]

Figure 4: List of Protocol Tags

Note well:

The S-NAPTR service and protocols are unrelated to the IANA Service Name and Transport Protocol Number registry.

The delimiter '.' in the protocol tags is only a separator for human reading convenience - not for structure or namespacing; it MUST NOT be parsed in any way by the querying application or resolver.

The use of the separator '.' is common also in other protocols' protocol tags. This is coincidence and does not imply a shared semantics with such protocols.

2.1.1.2. Definition of Conditions for Retry/Failure

RADIUS is a time-critical protocol; RADIUS clients which do not receive an answer after a configurable, but short, amount of time, will consider the request failed. Due to this, there is little leeway for extensive retries.

As a general rule, only error conditions which generate an immediate response from the other end are eligible for a retry of a discovered target. Any error condition involving timeouts, or the absence of a reply for more than one second during the connection setup phase is to be considered a failure; the next target in the set of discovered NAPTR targets is to be tried.

Note that [RFC3958] already defines that a failure to identify the server as being authoritative for the realm is always considered a failure; so even if a discovered target returns a wrong credential instantly, it is not eligible for retry.

Furthermore, the contacted RADIUS/TLS server verifies during connection setup whether or not it finds the connecting RADIUS/TLS client authorized or not. If the connecting RADIUS/TLS client is not found acceptable, the server will close the TLS connection immediately with an appropriate alert. Such TLS handshake failures are permanently fatal and not eligible for retry, unless the connecting client has more X.509 certificates to try; in this case, a retry with the remainder of its set of certificates SHOULD be attempted. Not trying all available client certificates potentially creates a DoS for the end-user whose authentication attempt triggered the discovery; one of the neglected certificates might have led to a successful RADIUS connection and subsequent end-user authentication.

If the TLS session setup to a discovered target does not succeed, that target (as identified by IP address and port number) SHOULD be ignored from the result set of any subsequent executions of the discovery algorithm at least until the target's Effective TTL (see

Section 3.3) has expired or until the entity which executes the algorithm changes its TLS context to either send a new client certificate or expect a different server certificate.

2.1.1.3. Server Identification and Handshake

After the algorithm in this document has been executed, a RADIUS/TLS session as per [RFC6614] is established. Since the dynamic discovery algorithm does not have provisions to establish confidential keying material between the RADIUS/TLS client (i.e. the server which executes the discovery algorithm) and the RADIUS/TLS server which was discovered, TLS-PSK ciphersuites cannot be used in the subsequent TLS handshake. Only TLS ciphersuites using X.509 certificates can be used with this algorithm.

There are numerous ways to define which certificates are acceptable for use in this context. This document defines one mandatory-to-implement mechanism which allows to verify whether the contacted host is authoritative for an NAI realm or not. It also gives one example of another mechanism which is currently in wide-spread deployment, and one possible approach based on DNSSEC which is yet unimplemented.

For the approaches which use trust roots (see the following two sections), a typical deployment will use a dedicated trust store for RADIUS/TLS certificate authorities, particularly a trust store which is independent from default "browser" trust stores. Often, this will be one or few CAs, and they only issue certificates for the specific purpose of establishing RADIUS server-to-server trust. It is important not to trust a large set of CAs which operate outside the control of the roaming consortium, for their issuance of certificates with the properties important for authorisation (such as NAIRealm and policyOID below) is difficult to verify. Therefore, clients SHOULD NOT be pre-configured with a list of known public CAs by the vendor or manufacturer. Instead, the clients SHOULD start off with an empty CA list. The addition of a CA SHOULD be done only when manually configured by an administrator.

2.1.1.3.1. Mandatory-to-implement mechanism: Trust Roots + NAIRealm

Verification of authority to provide AAA services over RADIUS/TLS is a two-step process.

Step 1 is the verification of certificate wellformedness and validity as per [RFC5280] and whether it was issued from a root certificate which is deemed trustworthy by the RADIUS/TLS client.

Step 2 is to compare the value of algorithm's variable "R" after the execution of step 3 of the discovery algorithm in Section 3.4.3 below

(i.e. after a consortium name mangling, but before conversion to a form usable by the name resolution library) to all values of the contacted RADIUS/TLS server's X.509 certificate property "subjectAlternativeName:otherName:NAIRealm" as defined in Section 2.2.

2.1.1.3.2. Other mechanism: Trust Roots + policyOID

Verification of authority to provide AAA services over RADIUS/TLS is a two-step process.

Step 1 is the verification of certificate wellformedness and validity as per [RFC5280] and whether it was issued from a root certificate which is deemed trustworthy by the RADIUS/TLS client.

Step 2 is to compare the values of the contacted RADIUS/TLS server's X.509 certificate's extensions of type "Policy OID" to a list of configured acceptable Policy OIDs for the roaming consortium. If one of the configured OIDs is found in the certificate's Policy OID extensions, then the server is considered authorized; if there is no match, the server is considered unauthorized.

This mechanism is inferior to the mandatory-to-implement mechanism in the previous section because all authorized servers are validated by the same OID value; the mechanism is not fine-grained enough to express authority for one specific realm inside the consortium. If the consortium contains members which are hostile against other members, this weakness can be exploited by one RADIUS/TLS server impersonating another if DNS responses can be spoofed by the hostile member.

The shortcomings in server identification can be partially mitigated by using the RADIUS infrastructure only with authentication payloads which provide mutual authentication and credential protection (i.e. EAP types passing the criteria of [RFC4017]): using mutual authentication prevents the hostile server from mimicking the real EAP server (it can't terminate the EAP authentication unnoticed because it does not have the server certificate from the real EAP server); protection of credentials prevents the impersonating server from learning usernames and passwords of the ongoing EAP conversation (other RADIUS attributes pertaining to the authentication, such as the EAP peer's Calling-Station-ID, can still be learned though).

2.1.1.3.3. Other mechanism: DNSSEC / DANE

Where DNSSEC is used, the results of the algorithm can be trusted; i.e. the entity which executes the algorithm can be certain that the realm that triggered the discovery is actually served by the server

that was discovered via DNS. However, this does not guarantee that the server is also authorized (i.e. a recognised member of the roaming consortium). The server still needs to present an X.509 certificate proving its authority to serve a particular realm.

The authorization can be sketched using DNSSEC+DANE as follows: DANE/TLSA records of all authorized servers are put into a DNSSEC zone which contains all known and authorised realms; the zone is rooted in a common, consortium-agreed branch of the DNS tree. The entity executing the algorithm uses the realm information from the authentication attempt, and then attempts to retrieve TLSA Resource Records (TLSA RR) for the DNS label "realm.commonroot". It then verifies that the presented server certificate during the RADIUS/TLS handshake matches the information in the TLSA record.

Example:

```
Realm = "example.com"

Common Branch = "idp.roaming-consortium.example."

label for TLSA query = "example.com.idp.roaming-
consortium.example."

result of discovery algorithm for realm "example.com" =
192.0.2.1:2083

( TLS certificate of 192.0.2.1:2083 matches TLSA RR ? "PASS" :
"FAIL" )
```

2.1.1.3.4. Client Authentication and Authorisation

Note that RADIUS/TLS connections always mutually authenticate the RADIUS server and the RADIUS client. This specification provides an algorithm for a RADIUS client to contact and verify authorization of a RADIUS server only. During connection setup, the RADIUS server also needs to verify whether it considers the connecting RADIUS client authorized; this is outside the scope of this specification.

2.1.2. SRV

This specification defines two SRV prefixes (i.e. two values for the "_service._proto" part of an SRV RR as per [RFC2782]):

SRV Label	Use
_radiustls._tcp	RADIUS transported over TLS as defined in [RFC6614]
_radiusdtls._udp	RADIUS transported over DTLS as defined in [RFC7360]

Figure 5: List of SRV Labels

Just like NAPTR records, the lookup and subsequent follow-up of SRV records may yield more than one server to contact in a prioritised list. [RFC2782] does not specify rules regarding "Definition of Conditions for Retry/Failure", nor "Server Identification and Handshake". This specification defines that the rules for these two topics as defined in Section 2.1.1.2 and Section 2.1.1.3 SHALL be used both for targets retrieved via an initial NAPTR RR as well as for targets retrieved via an initial SRV RR (i.e. in the absence of NAPTR RRs).

2.1.1.3. Optional name mangling

It is expected that in most cases, the SRV and/or NAPTR label used for the records is the DNS A-label representation of the literal realm name for which the server is the authoritative RADIUS server (i.e. the realm name after conversion according to section 5 of [RFC5891]).

However, arbitrary other labels or service tags may be used if, for example, a roaming consortium uses realm names which are not associated to DNS names or special-purpose consortia where a globally valid discovery is not a use case. Such other labels require a consortium-wide agreement about the transformation from realm name to lookup label, and/or which service tag to use.

Examples:

- a. A general-purpose RADIUS server for realm example.com might have DNS entries as follows:

```
example.com. IN NAPTR 50 50 "s" "aaa+auth:radius.tls.tcp" ""
_radiustls._tcp.foobar.example.com.

_radiustls._tcp.foobar.example.com. IN SRV 0 10 2083
radsec.example.com.
```

- b. The consortium "foo" provides roaming services for its members only. The realms used are of the form enterprise-name.example. The consortium operates a special purpose DNS server for the (private) TLD "example" which all RADIUS servers use to resolve realm names. "Company, Inc." is part of the consortium. On the consortium's DNS server, realm company.example might have the following DNS entries:

```
company.example. IN NAPTR 50 50 "a"
"aaa+auth:radius.dtls.udp" "" roamsvr.company.example.
```

- c. The eduroam consortium (see [I-D.wierenga-ietf-eduroam]) uses realms based on DNS, but provides its services to a closed community only. However, a AAA domain participating in eduroam may also want to expose AAA services to other, general-purpose, applications (on the same or other RADIUS servers). Due to that, the eduroam consortium uses the service tag "x-eduroam" for authentication purposes and eduroam RADIUS servers use this tag to look up other eduroam servers. An eduroam participant example.org which also provides general-purpose AAA on a different server uses the general "aaa+auth" tag:

```
example.org. IN NAPTR 50 50 "s" "x-eduroam:radius.tls.tcp" ""
_radiustls._tcp.eduroam.example.org.
```

```
example.org. IN NAPTR 50 50 "s" "aaa+auth:radius.tls.tcp" ""
_radiustls._tcp.aaa.example.org.
```

```
_radiustls._tcp.eduroam.example.org. IN SRV 0 10 2083 aaa-
eduroam.example.org.
```

```
_radiustls._tcp.aaa.example.org. IN SRV 0 10 2083 aaa-
default.example.org.
```

2.2. Definition of the X.509 certificate property

SubjectAltName:otherName:NAIRealm

This specification retrieves IP addresses and port numbers from the Domain Name System which are subsequently used to authenticate users via the RADIUS/TLS protocol. Regardless whether the results from DNS discovery are trustworthy or not (e.g. DNSSEC in use), it is always important to verify that the server which was contacted is authorized to service requests for the user which triggered the discovery process.

The input to the algorithm is an NAI realm as specified in Section 3.4.1. As a consequence, the X.509 certificate of the server which is ultimately contacted for user authentication needs to be

able to express that it is authorized to handle requests for that realm.

Current subjectAltName fields do not semantically allow to express an NAI realm; the field subjectAltName:dnsName is syntactically a good match but would inappropriately conflate DNS names and NAI realm names. Thus, this specification defines a new subjectAltName field to hold either a single NAI realm name or a wildcard name matching a set of NAI realms.

The subjectAltName:otherName:SRVName field certifies that a certificate holder is authorized to provide a service; this can be compared to the target of DNS label's SRV resource record. If the Domain Name System is insecure, it is required that the label of the SRV record itself is known-correct. In this specification, that label is not known-correct; it is potentially derived from a (potentially untrusted) NAPTR resource record of another label. If DNS is not secured with DNSSEC, the NAPTR resource record may have been altered by an attacker with access to the Domain Name System resolution, and thus the label to lookup the SRV record for may already be tainted. This makes subjectAltName:otherName:SRVName not a trusted comparison item.

Further to this, this specification's NAPTR entries may be of type "A" which do not involve resolution of any SRV records, which again makes subjectAltName:otherName:SRVName unsuited for this purpose.

This section defines the NAIRealm name as a form of otherName from the GeneralName structure in SubjectAltName defined in [RFC5280].

```
id-on-naiRealm OBJECT IDENTIFIER ::= { id-on XXX }
```

```
ub-naiRealm-length INTEGER ::= 255
```

```
NAIRealm ::= UTF8String (SIZE (1..ub-naiRealm-length))
```

The NAIRealm, if present, MUST contain an NAI realm as defined in [I-D.ietf-radext-nai]. It MAY substitute the leftmost dot-separated label of the NAI with the single character "*" to indicate a wildcard match for "all labels in this part". Further features of regular expressions, such as a number of characters followed by a * to indicate a common prefix inside the part, are not permitted.

The comparison of an NAIRealm to the NAI realm as derived from user input with this algorithm is a byte-by-byte comparison, except for the optional leftmost dot-separated part of the value whose content is a single "*" character; such labels match all strings in the same dot-separated part of the NAI realm. If at least one of the

sAN:otherName:NAIRealm values matches the NAI realm, the server is considered authorized; if none matches, the server is considered unauthorized.

Since multiple names and multiple name forms may occur in the subjectAltName extension, an arbitrary number of NAIRealms can be specified in a certificate.

Examples:

NAI realm (RADIUS)	NAIRealm (cert)	MATCH?
foo.example	foo.example	YES
foo.example	*.example	YES
bar.foo.example	*.example	NO
bar.foo.example	*ar.foo.example	NO (NAIRealm invalid)
bar.foo.example	bar.*.example	NO (NAIRealm invalid)
bar.foo.example	*.*.example	NO (NAIRealm invalid)
sub.bar.foo.example	*.*.example	NO (NAIRealm invalid)
sub.bar.foo.example	*.bar.foo.example	YES

Figure 6: Examples for NAI realm vs. certificate matching

Appendix A contains the ASN.1 definition of the above objects.

3. DNS-based NAPTR/SRV Peer Discovery

3.1. Applicability

Dynamic server discovery as defined in this document is only applicable for new AAA transactions and per service (i.e. distinct discovery is needed for Authentication, Accounting, and Dynamic Authorization) where a RADIUS entity which acts as a forwarding server for one or more realms receives a request with a realm for which it is not authoritative, and which no explicit next hop is configured. It is only applicable for

- a. new user sessions, i.e. for the initial Access-Request. Subsequent messages concerning this session, for example Access-Challenges and Access-Accepts use the previously-established communication channel between client and server.
- b. the first accounting ticket for a user session.
- c. the first RADIUS DynAuth packet for a user session.

3.2. Configuration Variables

The algorithm contains various variables for timeouts. These variables are named here and reasonable default values are provided. Implementations wishing to deviate from these defaults should make they understand the implications of changes.

DNS_TIMEOUT: maximum amount of time to wait for the complete set of all DNS queries to complete: Default = 3 seconds

MIN_EFF_TTL: minimum DNS TTL of discovered targets: Default = 60 seconds

BACKOFF_TIME: if no conclusive DNS response was retrieved after DNS_TIMEOUT, do not attempt dynamic discovery before BACKOFF_TIME has elapsed. Default = 600 seconds

3.3. Terms

Positive DNS response: a response which contains the RR that was queried for.

Negative DNS response: a response which does not contain the RR that was queried for, but contains an SOA record along with a TTL indicating cache duration for this negative result.

DNS Error: Where the algorithm states "name resolution returns with an error", this shall mean that either the DNS request timed out, or a DNS response which is neither a positive nor a negative response (e.g. SERVFAIL).

Effective TTL: The validity period for discovered RADIUS/TLS target hosts. Calculated as: Effective TTL (set of DNS TTL values) = max { MIN_EFF_TTL, min { DNS TTL values } }

SRV lookup: for the purpose of this specification, SRV lookup procedures are defined as per [RFC2782], but excluding that RFCs "A" fallback as defined in its section "Usage Rules", final "else" clause.

Greedy result evaluation: The NAPTR to SRV/A/AAAA resolution may lead to a tree of results, whose leafs are the IP addresses to contact. The branches of the tree are ordered according to their order/preference DNS properties. An implementation is executing greedy result evaluation if it uses a depth-first search in the tree along the highest order results, attempts to connect to the corresponding resulting IP addresses, and only backtracks to other branches if the higher ordered results did not end in successful connection attempts.

3.4. Realm to RADIUS server resolution algorithm

3.4.1. Input

For RADIUS Authentication and RADIUS Accounting server discovery, input I to the algorithm is the RADIUS User-Name attribute with content of the form "user@realm"; the literal @ sign being the separator between a local user identifier within a realm and its realm. The use of multiple literal @ signs in a User-Name is strongly discouraged; but if present, the last @ sign is to be considered the separator. All previous instances of the @ sign are to be considered part of the local user identifier.

For RADIUS DynAuth Server discovery, input I to the algorithm is the domain name of the operator of a RADIUS realm as was communicated during user authentication using the Operator-Name attribute ([RFC5580], section 4.1). Only Operator-Name values with the namespace "1" are supported by this algorithm - the input to the algorithm is the actual domain name, preceded with an "@" (but without the "1" namespace identifier byte of that attribute).

Note well: The attribute User-Name is defined to contain UTF-8 text. In practice, the content may or may not be UTF-8. Even if UTF-8, it may or may not map to a domain name in the realm part. Implementors MUST take possible conversion error paths into consideration when parsing incoming User-Name attributes. This document describes server discovery only for well-formed realms mapping to DNS domain names in UTF-8 encoding. The result of all other possible contents of User-Name is unspecified; this includes, but is not limited to:

- Usage of separators other than @.

- Encoding of User-Name in local encodings.

- UTF-8 realms which fail the conversion rules as per [RFC5891].

- UTF-8 realms which end with a . ("dot") character.

For the last bullet point, "trailing dot", special precautions should be taken to avoid problems when resolving servers with the algorithm below: they may resolve to a RADIUS server even if the peer RADIUS server only is configured to handle the realm without the trailing dot. If that RADIUS server again uses NAI discovery to determine the authoritative server, the server will forward the request to localhost, resulting in a tight endless loop.

3.4.2. Output

Output O of the algorithm is a two-tuple consisting of: O-1) a set of tuples {hostname; port; protocol; order/preference; Effective TTL} - the set can be empty; and O-2) an integer: if the set in the first part of the tuple is empty, the integer contains the Effective TTL for backoff timeout, if the set is not empty, the integer is set to 0 (and not used).

3.4.3. Algorithm

The algorithm to determine the RADIUS server to contact is as follows:

1. Determine P = (position of last "@" character) in I.
2. generate R = (substring from P+1 to end of I)
3. modify R according to agreed consortium procedures if applicable
4. convert R to a representation usable by the name resolution library if needed
5. Initialize TIMER = 0; start TIMER. If TIMER reaches DNS_TIMEOUT, continue at step 20.
6. Using the host's name resolution library, perform a NAPTR query for R (see "Delay considerations" below). If the result is a negative DNS response, O-2 = Effective TTL (TTL value of the SOA record) and continue at step 13. If name resolution returns with error, O-1 = { empty set }, O-2 = BACKOFF_TIME and terminate.
7. Extract NAPTR records with service tag "aaa+auth", "aaa+acct", "aaa+dynauth" as appropriate. Keep note of the protocol tag and remaining TTL of each of the discovered NAPTR records.
8. If no records found, continue at step 13.
9. For the extracted NAPTRs, perform successive resolution as defined in [RFC3958], section 2.2. An implementation MAY use greedy result evaluation according to the NAPTR order/preference fields (i.e. can execute the subsequent steps of this algorithm for the highest-order entry in the set of results, and only lookup the remainder of the set if necessary).
10. If the set of hostnames is empty, O-1 = { empty set }, O-2 = BACKOFF_TIME and terminate.

11. $O' = (\text{set of } \{\text{hostname; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this hostname) } \} \text{ for all terminal lookup results}).$
12. Proceed with step 18.
13. Generate $R' = (\text{prefix R with } _radiustls._tcp. \text{ and/or } _radiustls._udp.)$
14. Using the host's name resolution library, perform SRV lookup with R' as label (see "Delay considerations" below).
15. If name resolution returns with error, $O-1 = \{ \text{empty set} \}$, $O-2 = \text{BACKOFF_TIME}$ and terminate.
16. If the result is a negative DNS response, $O-1 = \{ \text{empty set} \}$, $O-2 = \min \{ O-2, \text{Effective TTL (TTL value of the SOA record) } \}$ and terminate.
17. $O' = (\text{set of } \{\text{hostname; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this result) } \} \text{ for all hostnames}).$
18. Generate $O-1$ by resolving hostnames in O' into corresponding A and/or AAAA addresses: $O-1 = (\text{set of } \{\text{IP address; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this result) } \} \text{ for all hostnames }), O-2 = 0.$
19. For each element in $O-1$, test if the original request which triggered dynamic discovery was received on $\{\text{IP address; port}\}$. If yes, $O-1 = \{ \text{empty set} \}$, $O-2 = \text{BACKOFF_TIME}$, log error, Terminate (see next section for a rationale). If no, O is the result of dynamic discovery. Terminate.
20. $O-1 = \{ \text{empty set} \}$, $O-2 = \text{BACKOFF_TIME}$, log error, Terminate.

3.4.4. Validity of results

The dynamic discovery algorithm is used by servers which do not have sufficient configuration information to process an incoming request on their own. If the discovery algorithm result contains the server's own listening address (IP address and port), then there is a potential for an endless forwarding loop. If the listening address is the DNS result with the highest priority, the server will enter a tight loop (the server would forward the request to itself, triggering dynamic discovery again in a perpetual loop). If the address has a lower priority in the set of results, there is a potential loop with intermediate hops in between (the server could

forward to another host with a higher priority, which might use DNS itself and forward the packet back to the first server). The underlying reason that enables these loops is that the server executing the discovery algorithm is seriously misconfigured in that it does not recognise the request as one that is to be processed by itself. RADIUS has no built-in loop detection, so any such loops would remain undetected. So, if step 18 of the algorithm discovers such a possible-loop situation, the algorithm should be aborted and an error logged. Note that this safeguard does not provide perfect protection against routing loops. One reason which might introduce a loop include the possibility that a subsequent hop has a statically configured next-hop which leads to an earlier host in the loop. Another reason for occurring loops is if the algorithm was executed with greedy result evaluation, and the own address was in a lower-priority branch of the result set which was not retrieved from DNS at all, and thus can't be detected.

After executing the above algorithm, the RADIUS server establishes a connection to a home server from the result set. This connection can potentially remain open for an indefinite amount of time. This conflicts with the possibility of changing device and network configurations on the receiving end. Typically, TTL values for records in the name resolution system are used to indicate how long it is safe to rely on the results of the name resolution. If these TTLs are very low, thrashing of connections becomes possible; the Effective TTL mitigates that risk. When a connection is open and the smallest of the Effective TTL value which was learned during discovering the server has not expired, subsequent new user sessions for the realm which corresponds to that open connection SHOULD re-use the existing connection and SHOULD NOT re-execute the dynamic discovery algorithm nor open a new connection. To allow for a change of configuration, a RADIUS server SHOULD re-execute the dynamic discovery algorithm after the Effective TTL that is associated with this connection has expired. The server SHOULD keep the session open during this re-assessment to avoid closure and immediate re-opening of the connection should the result not have changed.

Should the algorithm above terminate with O-1 = empty set, the RADIUS server SHOULD NOT attempt another execution of this algorithm for the same target realm before the timeout O-2 has passed.

3.4.5. Delay considerations

The host's name resolution library may need to contact outside entities to perform the name resolution (e.g. authoritative name servers for a domain), and since the NAI discovery algorithm is based on uncontrollable user input, the destination of the lookups is out of control of the server that performs NAI discovery. If such

outside entities are misconfigured or unreachable, the algorithm above may need an unacceptably long time to terminate. Many RADIUS implementations time out after five seconds of delay between Request and Response. It is not useful to wait until the host name resolution library signals a timeout of its name resolution algorithms. The algorithm therefore controls execution time with TIMER. Execution of the NAI discovery algorithm SHOULD be non-blocking (i.e. allow other requests to be processed in parallel to the execution of the algorithm).

3.4.6. Example

Assume

a user from the Technical University of Munich, Germany, has a RADIUS User-Name of "foobar@tu-m[U+00FC]nchen.example".

The name resolution library on the RADIUS forwarding server does not have the realm tu-m[U+00FC]nchen.example in its forwarding configuration, but uses DNS for name resolution and has configured the use of Dynamic Discovery to discover RADIUS servers.

It is IPv6-enabled and prefers AAAA records over A records.

It is listening for incoming RADIUS/TLS requests on 192.0.2.1, TCP /2083.

May the configuration variables be

DNS_TIMEOUT = 3 seconds

MIN_EFF_TTL = 60 seconds

BACKOFF_TIME = 3600 seconds

If DNS contains the following records:

xn--tu-mnchen-t9a.example. IN NAPTR 50 50 "s"
"aaa+auth:radius.tls.tcp" "" _myradius._tcp.xn--tu-mnchen-t9a.example.

xn--tu-mnchen-t9a.example. IN NAPTR 50 50 "s"
"fooservice:bar.dccp" "" _abc123._def.xn--tu-mnchen-t9a.example.

_myradius._tcp.xn--tu-mnchen-t9a.example. IN SRV 0 10 2083
radsecserver.xn--tu-mnchen-t9a.example.

```
_myradius._tcp.xn--tu-mnchen-t9a.example.  IN SRV 0 20 2083  
backupserver.xn--tu-mnchen-t9a.example.
```

```
radsecserver.xn--tu-mnchen-t9a.example.  IN AAAA  
2001:0DB8::202:44ff:fe0a:f704
```

```
radsecserver.xn--tu-mnchen-t9a.example.  IN A 192.0.2.3
```

```
backupserver.xn--tu-mnchen-t9a.example.  IN A 192.0.2.7
```

Then the algorithm executes as follows, with I =
"foobar@tu-m[U+00FC]nchen.example", and no consortium name mangling
in use:

1. P = 7
2. R = "tu-m[U+00FC]nchen.example"
3. NOOP
4. name resolution library converts R to xn--tu-mnchen-t9a.example
5. TIMER starts.
6. Result:

```
(TTL = 47) 50 50 "s" "aaa+auth:radius.tls.tcp" ""  
_myradius._tcp.xn--tu-mnchen-t9a.example.  
  
(TTL = 522) 50 50 "s" "fooservice:bar.dccp" ""  
_abc123._def.xn--tu-mnchen-t9a.example.
```
7. Result:

```
(TTL = 47) 50 50 "s" "aaa+auth:radius.tls.tcp" ""  
_myradius._tcp.xn--tu-mnchen-t9a.example.
```
8. NOOP
9. Successive resolution performs SRV query for label
_myradius._tcp.xn--tu-mnchen-t9a.example, which results in

```
(TTL 499) 0 10 2083 radsec.xn--tu-mnchen-t9a.example.  
  
(TTL 2200) 0 20 2083 backup.xn--tu-mnchen-t9a.example.
```
10. NOOP

11. $O' = \{$
 (radsec.xn--tu-mnchen-t9a.example.; 2083; RADIUS/TLS; 10;
 60),
 (backup.xn--tu-mnchen-t9a.example.; 2083; RADIUS/TLS; 20; 60)
 } // minimum TTL is 47, up'ed to MIN_EFF_TTL
12. Continuing at 18.
13. (not executed)
14. (not executed)
15. (not executed)
16. (not executed)
17. (not executed)
18. $O-1 = \{$
 (2001:0DB8::202:44ff:fe0a:f704; 2083; RADIUS/TLS; 10; 60),
 (192.0.2.7; 2083; RADIUS/TLS; 20; 60)
 }; $O-2 = 0$
19. No match with own listening address; terminate with tuple (O-1,
 O-2) from previous step.

The implementation will then attempt to connect to two servers, with preference to [2001:0DB8::202:44ff:fe0a:f704]:2083 using the RADIUS/TLS protocol.

4. Operations and Manageability Considerations

The discovery algorithm as defined in this document contains several options; the major ones being use of NAPTR vs. SRV; how to determine the authorization status of a contacted server for a given realm; which trust anchors to consider trustworthy for the RADIUS conversation setup.

Random parties which do not agree on the same set of options may not be able to interoperate. However, such a global interoperability is not intended by this document.

Discovery as per this document becomes important inside a roaming consortium, which has set up roaming agreements with the other partners. Such roaming agreements require much more than a technical means of server discovery; there are administrative and contractual considerations at play (service contracts, backoffice compensations, procedures, ...).

A roaming consortium's roaming agreement must include a profile of which choice points of this document to use. So long as the roaming consortium can settle on one deployment profile, they will be able to interoperate based on that choice; this per-consortium interoperability is the intended scope of this document.

5. Security Considerations

When using DNS without DNSSEC security extensions and validation for all of the replies to NAPTR, SRV and A/AAAA requests as described in section Section 3, the result of the discovery process can not be trusted. Even if it can be trusted (i.e. DNSSEC is in use), actual authorization of the discovered server to provide service for the given realm needs to be verified. A mechanism from section Section 2.1.1.3 or equivalent MUST be used to verify authorization.

The algorithm has a configurable completion timeout `DNS_TIMEOUT` defaulting to three seconds for RADIUS' operational reasons. The lookup of DNS resource records based on unverified user input is an attack vector for DoS attacks: an attacker might intentionally craft bogus DNS zones which take a very long time to reply (e.g. due to a particularly byzantine tree structure, or artificial delays in responses).

To mitigate this DoS vector, implementations SHOULD consider rate-limiting either their amount of new executions of the dynamic discovery algorithm as a whole, or the amount of intermediate responses to track, or at least the number of pending DNS queries. Implementations MAY choose lower values than the default for `DNS_TIMEOUT` to limit the impact of DoS attacks via that vector. They MAY also continue their attempt to resolve DNS records even after `DNS_TIMEOUT` has passed; a subsequent request for the same realm might benefit from retrieving the results anyway. The amount of time to spent waiting for a result will influence the impact of a possible DoS attack; the waiting time value is implementation dependent and outside the scope of this specification.

With Dynamic Discovery being enabled for a RADIUS Server, and depending on the deployment scenario, the server may need to open up its target IP address and port for the entire internet, because arbitrary clients may discover it as a target for their

authentication requests. If such clients are not part of the roaming consortium, the RADIUS/TLS connection setup phase will fail (which is intended) but the computational cost for the connection attempt is significant. With the port for a TLS-based service open, the RADIUS server shares all the typical attack vectors for services based on TLS (such as HTTPS, SMTPS, ...). Deployments of RADIUS/TLS with Dynamic Discovery should consider these attack vectors and take appropriate counter-measures (e.g. blacklisting known-bad IPs on a firewall, rate-limiting new connection attempts, etc.).

6. Privacy Considerations

The classic RADIUS operational model (known, pre-configured peers, shared secret security, mostly plaintext communication) and this new RADIUS dynamic discovery model (peer discovery with DNS, PKI security and packet confidentiality) differ significantly in their impact on the privacy of end users trying to authenticate to a RADIUS server.

With classic RADIUS, traffic in large environments gets aggregated by statically configured clearinghouses. The packets sent to those clearinghouses and their responses are mostly unprotected. As a consequence,

- o All intermediate IP hops can inspect most of the packet payload in clear text, including the User-Name and Calling-Station-Id attributes, and can observe which client sent the packet to which clearinghouse. This allows the creation of mobility profiles for any passive observer on the IP path.
- o The existence of a central clearinghouse creates an opportunity for the clearinghouse to trivially create the same mobility profiles. The clearinghouse may or may not be trusted not to do this, e.g. by sufficiently threatening contractual obligations.
- o In addition to that, with the clearinghouse being a RADIUS intermediate in possession of a valid shared secret, the clearinghouse can observe and record even the security-critical RADIUS attributes such as User-Password. This risk may be mitigated by choosing authentication payloads which are cryptographically secured and do not use the attribute User-Password - such as certain EAP types.
- o There is no additional information disclosure to parties outside the IP path between the RADIUS client and server (in particular, no DNS servers learn about realms of current ongoing authentications).

With RADIUS and dynamic discovery,

- o This protocol allows for RADIUS clients to identify and directly connect to the RADIUS home server. This can eliminate the use of clearinghouses to do forwarding of requests, and it also eliminates the ability of the clearinghouse to then aggregate the user information that flows through it. However, there exist reasons why clearinghouses might still be used. One reason to keep a clearinghouse is to act as a gateway for multiple backends in a company; another reason may be a requirement to sanitise RADIUS datagrams (filter attributes, tag requests with new attributes, ...).
- o Even where intermediate proxies continue to be used for reasons unrelated to dynamic discovery, the number of such intermediates may be reduced by removing those proxies which are only deployed for pure request routing reasons. This reduces the number of entities which can inspect the RADIUS traffic.
- o RADIUS clients which make use of dynamic discovery will need to query the Domain Name System, and use a user's realm name as the query label. A passive observer on the IP path between the RADIUS client and the DNS server(s) being queried can learn that a user of that specific realm was trying to authenticate at that RADIUS client at a certain point in time. This may or may not be sufficient for the passive observer to create a mobility profile. During the recursive DNS resolution, a fair number of DNS servers and the IP hops in between those get to learn that information. Not every single authentication triggers DNS lookups, so there is no one-to-one relation of leaked realm information and the number of authentications for that realm.
- o Since dynamic discovery operates on a RADIUS hop-by-hop basis, there is no guarantee that the RADIUS payload is not transmitted between RADIUS systems which do not make use of this algorithm, and possibly using other transports such as RADIUS/UDP. On such hops, the enhanced privacy is jeopardized.

In summary, with classic RADIUS, few intermediate entities learn very detailed data about every ongoing authentications, while with dynamic discovery, many entities learn only very little about recently authenticated realms.

7. IANA Considerations

This document requests IANA registration of the following entries in existing registries:

- o S-NAPTR Application Service Tags registry

- * aaa+auth
- * aaa+acct
- * aaa+dynauth
- o S-NAPTR Application Protocol Tags registry
 - * radius.tls.tcp
 - * radius.dtls.udp

This document reserves the use of the "radiustls" and "radiusdtls" service names. Registration information as per [RFC6335] section 8.1.1 is as follows:

Service Name: radiustls; radiusdtls

Transport Protocols: TCP (for radiustls), UDP (for radiusdtls)

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Authentication, Accounting and Dynamic authorization via the RADIUS protocol. These service names are used to construct the SRV service labels "_radiustls" and "_radiusdtls" for discovery of RADIUS/TLS and RADIUS/DTLS servers, respectively.

Reference: RFC Editor Note: please insert the RFC number of this document. The protocol does not use broadcast, multicast or anycast communication.

This specification makes use of the SRV Protocol identifiers "_tcp" and "_udp" which are mentioned as early as [RFC2782] but do not appear to be assigned in an actual registry. Since they are in widespread use in other protocols, this specification refrains from requesting a new registry "RADIUS/TLS SRV Protocol Registry" and continues to make use of these tags implicitly.

This document requires that a number of Object Identifiers be assigned. They are now under the control of IANA following [RFC7299]

IANA is requested to assign the following identifiers:

TBD99 is to be assigned from the "SMI Security for PKIX Module Identifier Registry". The suggested description is id-mod-nai-realm-08.

TBD98 is to be assigned from the "SMI Security for PKIX Other Name Forms Registry." The suggested description is id-on-naiRealm.

RFC Editor Note: please replace the occurrences of TBD98 and TBD99 in Appendix A of the document with the actually assigned numbers.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.
- [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, January 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5580] Tschofenig, H., Adrangi, F., Jones, M., Lior, A., and B. Aboba, "Carrying Location Objects in RADIUS and Diameter", RFC 5580, August 2009.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.

[RFC7360] DeKok, A., "Datagram Transport Layer Security (DTLS) as a Transport Layer for RADIUS", RFC 7360, September 2014.

[I-D.ietf-radext-nai]

DeKok, A., "The Network Access Identifier", draft-ietf-radext-nai-15 (work in progress), December 2014.

8.2. Informative References

[RFC4017] Stanley, D., Walker, J., and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs", RFC 4017, March 2005.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.

[RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", RFC 7299, July 2014.

[I-D.wierenga-ietf-eduroam]

Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam architecture for network roaming", draft-wierenga-ietf-eduroam-05 (work in progress), March 2015.

Appendix A. Appendix A: ASN.1 Syntax of NAIRealm

```
PKIXNaiRealm08 {iso(1) identified-organization(3) dod(6)
  internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-nai-realm-08 (TBD99) }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS

    id-pkix
    FROM PKIX1Explicit-2009
        {iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-pkix1-explicit-02(51)}
        -- from RFC 5280, RFC 5912

    OTHER-NAME
    FROM PKIX1Implicit-2009
        {iso(1) identified-organization(3) dod(6) internet(1) security(5)
        mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-implicit-02(59)}
        -- from RFC 5280, RFC 5912
;

-- Service Name Object Identifier

id-on    OBJECT IDENTIFIER ::= { id-pkix 8 }

id-on-naiRealm OBJECT IDENTIFIER ::= { id-on TBD98 }

-- Service Name

naiRealm OTHER-NAME ::= { NAIRealm IDENTIFIED BY { id-on-naiRealm }}

ub-naiRealm-length INTEGER ::= 255

NAIRealm ::= UTF8String (SIZE (1..ub-naiRealm-length))

END
```

Authors' Addresses

Stefan Winter
Fondation RESTENA
6, rue Richard Coudenhove-Kalergi
Luxembourg 1359
LUXEMBOURG

Phone: +352 424409 1
Fax: +352 422473
EMail: stefan.winter@restena.lu
URI: <http://www.restena.lu>.

Mike McCauley
AirSpayce Pty Ltd
9 Bulbul Place
Currumbin Waters QLD 4223
AUSTRALIA

Phone: +61 7 5598 7474
EMail: mikem@airspayce.com
URI: <http://www.airspayce.com>

RADEXT Working Group
INTERNET-DRAFT
Category: Proposed Standard
Expires: October 1, 2014
Updates: 3580, 4072

Bernard Aboba
Microsoft Corporation
Jouni Malinen
Devicescape Software
Paul Congdon
Tallac Networks
Joseph Salowey
Cisco Systems
Mark Jones
Azuca Systems
28 March 2014

RADIUS Attributes for IEEE 802 Networks
draft-ietf-radext-ieee802ext-12.txt

Abstract

RFC 3580 provides guidelines for the use of the Remote Authentication Dialin User Service (RADIUS) within IEEE 802 local area networks (LANs). This document defines additional attributes for use within IEEE 802 networks, as well as clarifying the usage of the EAP-Key-Name attribute and the Called-Station-Id attribute. This document updates RFC 3580 as well as RFC 4072.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on October 1, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1.	Introduction	4
1.1	Terminology	4
1.2	Requirements Language	5
2.	RADIUS attributes	5
2.1	Allowed-Called-Station-Id	5
2.2	EAP-Key-Name	6
2.3	EAP-Peer-Id	7
2.4	EAP-Server-Id	8
2.5	Mobility-Domain-Id	9
2.6	Preauth-Timeout	10
2.7	Network-Id-Name	11
2.8	EAPoL-Announcement	12
2.9	WLAN-HESSID	14
2.10	WLAN-Venue-Info	14
2.11	WLAN-Venue-Language	15
2.12	WLAN-Venue-Name	16
2.13	WLAN-Reason-Code	17
2.14	WLAN-Pairwise-Cipher	18
2.15	WLAN-Group-Cipher	19
2.16	WLAN-AKM-Suite	20
2.17	WLAN-Group-Mgmt-Cipher	21
2.18	WLAN-RF-Band	22
3.	Table of attributes	23
4.	IANA Considerations	24
5.	Security Considerations	24
6.	References	25
6.1	Normative References	25
6.2	Informative References	26
	ACKNOWLEDGMENTS	26
	AUTHORS' ADDRESSES	27

1. Introduction

In situations where it is desirable to centrally manage authentication, authorization and accounting (AAA) for IEEE 802 [IEEE-802] networks, deployment of a backend authentication and accounting server is desirable. In such situations, it is expected that IEEE 802 authenticators will function as AAA clients.

"IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines" [RFC3580] provides guidelines for the use of the Remote Authentication Dialin User Service (RADIUS) within networks utilizing IEEE 802 local area networks. This document defines additional attributes suitable for usage by IEEE 802 authenticators acting as AAA clients.

1.1. Terminology

This document uses the following terms:

Access Point (AP)	A Station that provides access to the distribution services via the wireless medium for associated Stations.
Association	The service used to establish Access Point/Station mapping and enable Station invocation of the distribution system services.
authenticator	An authenticator is an entity that require authentication from the supplicant. The authenticator may be connected to the supplicant at the other end of a point-to-point LAN segment or wireless link.
authentication server	An authentication server is an entity that provides an authentication service to an authenticator. This service verifies from the credentials provided by the supplicant, the claim of identity made by the supplicant.
Station (STA)	Any device that contains an IEEE 802.11 conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium (WM).
Supplicant	A supplicant is an entity that is being authenticated by an authenticator. The supplicant may be connected to the authenticator at one end of a point-to-point LAN segment or 802.11 wireless link.

1.2. Requirements Language

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. RADIUS attributes

2.1. Allowed-Called-Station-Id

Description

The Allowed-Called-Station-Id Attribute allows the RADIUS server to specify the authenticator MAC addresses and/or networks to which the user is allowed to connect. One or more Allowed-Called-Station-Id attributes MAY be included in an Access-Accept, CoA-Request or Accounting-Request packet.

The Allowed-Called-Station-Id Attribute can be useful in situations where pre-authentication is supported (e.g. IEEE 802.11 pre-authentication). In these scenarios, a Called-Station-Id Attribute typically will not be included within the Access-Request so that the RADIUS server will not know the network that the user is attempting to access. The Allowed-Called-Station-Id enables the RADIUS server to restrict the networks and attachment points to which the user can subsequently connect.

A summary of the Allowed-Called-Station-Id Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
+++++																																							
Type										Length										String...																			
+++++																																							

Code

TBD1

Length

 ≥ 3

String

The String field is one or more octets, specifying a Called-Station-Id that the user MAY connect to; if the Called-Station-Id that the user connects to does not match one of the Allowed-Called-Station-Id Attributes, the Network Authentication Server (NAS) MUST NOT permit the user to access the network.

In the case of IEEE 802, the Allowed-Called-Station-Id Attribute is used to store the Medium Access Control (MAC) address in ASCII format (upper case only), with octet values separated by a "-". Example: "00-10-A4-23-19-C0". Where restrictions on both the network and authenticator MAC address usage are intended, the network name MUST be appended to the authenticator MAC address, separated from the MAC address with a ":". Example: "00-10-A4-23-19-C0:AP1". Where no MAC address restriction is intended, the MAC address field MUST be omitted, but ":" and the network name field MUST be included. Example: ":AP1".

Within IEEE 802.11 [IEEE-802.11], the SSID constitutes the network name; within IEEE 802.1X [IEEE-802.1X] wired networks, the Network-Id Name (NID-Name) constitutes the network name. Since a NID-Name can be up to 253 octets in length, when used with [IEEE-802.1X] wired networks, there may not be sufficient room within the Allowed-Called-Station-Id Attribute to include both a MAC address and a Network Name. However, as the Allowed-Called-Station-Id Attribute is expected to be used largely in wireless access scenarios, this restriction is not considered serious.

2.2. EAP-Key-Name

Description

The EAP-Key-Name Attribute, defined in "Diameter Extensible Authentication Protocol (EAP) Application" [RFC4072], contains the EAP Session-Id, as described in "Extensible Authentication Protocol (EAP) Key Management Framework" [RFC5247]. Exactly how this Attribute is used depends on the link layer in question.

It should be noted that not all link layers use this name. An EAP-Key-Name Attribute MAY be included within Access-Request, Access-Accept and CoA-Request packets. A summary of the EAP-Key-Name Attribute format is shown below. The fields are transmitted from left to right.

0	1	2	3
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+			
Type		Length String...	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+			

Code

102 [RFC4072]

Length

>=3

String

The String field is one or more octets, containing the EAP Session-Id, as defined in "Extensible Authentication Protocol (EAP) Key Management Framework" [RFC5247]. Since the NAS operates as a pass-through in EAP, it cannot know the EAP Session-Id before receiving it from the RADIUS server. As a result, an EAP-Key-Name Attribute sent in an Access-Request MUST only contain a single NUL character. A RADIUS server receiving an Access-Request with an EAP-Key-Name Attribute containing anything other than a single NUL character MUST silently discard the Attribute. In addition, the RADIUS server SHOULD include this Attribute in an Access-Accept or CoA-Request only if an EAP-Key-Name Attribute was present in the Access-Request. Since a NAS will typically only include a EAP-Key-Name Attribute in an Access-Request in situations where the Attribute is required to provision service, if an EAP-Key-Name Attribute is included in an Access-Request but is not present in the Access-Accept, the NAS SHOULD treat the Access-Accept as though it were an Access-Reject. If an EAP-Key-Name Attribute was not present in the Access-Request but is included in the Access-Accept, then the NAS SHOULD silently discard the EAP-Key-Name Attribute. As noted in [IEEE-802.1X] Section 6.2.2, the Connectivity Association Key Name (CKN) is derived from the EAP Session-Id, and as described in Section 9.3.3, the CKN is subsequently used in the derivation of the Key Encrypting Key (KEK) and the Integrity Check Value Key (ICK), utilized to protect the secret keys (SAKs) utilized by Media Access Control Security (MACsec). As a result, for the NAS to acquire information needed in the MACsec Key Agreement (MKA) exchange, it needs to include the EAP-Key-Name attribute in the Access-Request and receive it from the RADIUS server in the Access-Accept.

2.3. EAP-Peer-Id

Description

The EAP-Peer-Id Attribute contains a Peer-Id generated by the EAP method. Exactly how this name is used depends on the link layer in question. See [RFC5247] for more discussion. The EAP-Peer-Id Attribute MAY be included in Access-Request, Access-Accept and

Accounting-Request packets. More than one EAP-Peer-Id Attribute MUST NOT be included in an Access-Request; one or more EAP-Peer-Id attributes MAY be included in an Access-Accept.

It should be noted that not all link layers use this name, and existing EAP method implementations do not generate it. Since the NAS operates as a pass-through in EAP [RFC3748], it cannot know the EAP-Peer-Id before receiving it from the RADIUS server. As a result, an EAP-Peer-Id Attribute sent in an Access-Request MUST only contain a single NUL character. A home RADIUS server receiving an Access-Request an EAP-Peer-Id Attribute containing anything other than a single NUL character MUST silently discard the Attribute. In addition, the home RADIUS server SHOULD include one or more EAP-Peer-Id attributes in an Access-Accept only if an EAP-Peer-Id Attribute was present in the Access-Request. If a NAS receives EAP-Peer-Id Attribute(s) in an Access-Accept without having included one in an Access-Request, the NAS SHOULD silently discard the Attribute(s). A summary of the EAP-Peer-Id Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      | Length      | String... |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Code

TBD2

Length

>=3

String

The String field is one or more octets containing a EAP Peer-Id exported by the EAP method. For details, see [RFC5247] Appendix A. A robust implementation SHOULD support the field as undistinguished octets. Only a single EAP Peer-Id may be included per Attribute.

2.4. EAP-Server-Id

Description

The EAP-Server-Id Attribute contains a Server-Id generated by the

EAP method. Exactly how this name is used depends on the link layer in question. See [RFC5247] for more discussion. The EAP-Server-Id Attribute is only allowed in Access-Request, Access-Accept, and Accounting-Request packets. More than one EAP-Server-Id Attribute MUST NOT be included in an Access-Request; one or more EAP-Server-Id attributes MAY be included in an Access-Accept.

It should be noted that not all link layers use this name, and existing EAP method implementations do not generate it. Since the NAS operates as a pass-through in EAP [RFC3748], it cannot know the EAP-Server-Id before receiving it from the RADIUS server. As a result, an EAP-Server-Id Attribute sent in an Access-Request MUST contain only a single NUL character. A home RADIUS server receiving in an Access-Request an EAP-Server-Id Attribute containing anything other than a single NUL character MUST silently discard the Attribute. In addition, the home RADIUS server SHOULD include this Attribute an Access-Accept only if an EAP-Server-Id Attribute was present in the Access-Request. A summary of the EAP-Server-Id Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
+++++																																							
Type										Length										String...																			
+++++																																							

Code

TBD3

Length

$$\geq 3$$

String

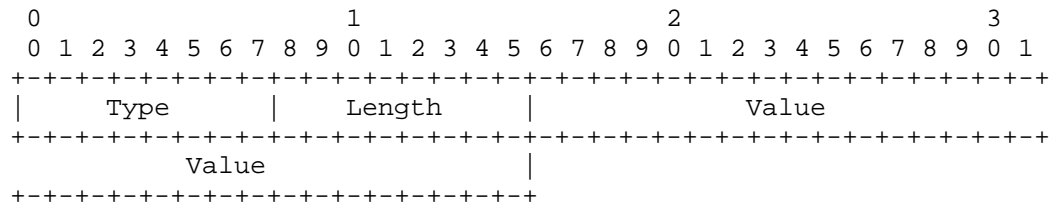
The String field is one or more octets, containing a EAP Server-Id exported by the EAP method. For details, see [RFC5247] Appendix A. A robust implementation SHOULD support the field as undistinguished octets.

2.5. Mobility-Domain-Id

Description

A single Mobility-Domain-Id Attribute MAY be included in an Access-Request or Accounting-Request, in order to enable the NAS

to provide the RADIUS server with the Mobility Domain Identifier (MDID), defined in Section 8.4.2.49 of [IEEE-802.11]. A summary of the Mobility-Domain-Id Attribute format is shown below. The fields are transmitted from left to right.



Code

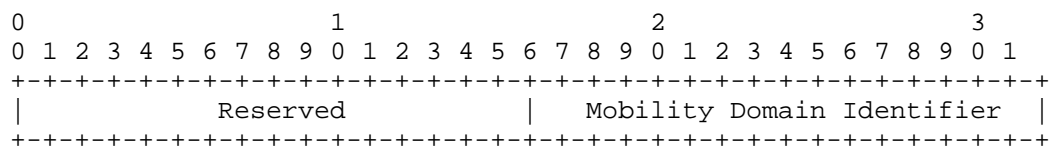
TBD4

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer. The two most significant octets MUST be set to zero by the sender, and are ignored by the receiver; the two least significant octets contain the Mobility Domain Identifier (MDID) defined in Section 8.4.2.49 of [IEEE-802.11].



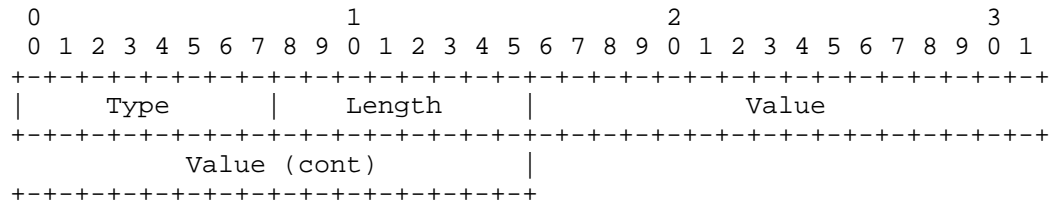
2.6. Preauth-Timeout

Description

This Attribute sets the maximum number of seconds which pre-authentication state is required to be kept by the NAS, without being utilized within a user session. For example, when [IEEE-802.11] pre-authentication is used, if a user has not attempted to utilize the Pairwise Master Key (PMK) derived as a result of pre-authentication within the time specified by the Preauth-Timeout Attribute, the PMK MAY be discarded by the Access Point. However, once the session is underway, the Preauth-Timeout Attribute has no bearing on the maximum session time for the user,

or the maximum time during which key state may be kept prior to re-authentication. This is determined by the Session-Timeout Attribute, if present.

A single Preauth-Timeout Attribute MAY be included within an Access-Accept or CoA-Request packet. A summary of the Preauth-Timeout Attribute format is shown below. The fields are transmitted from left to right.



Code

TBD5

Length

6

Value

The field is 4 octets, containing a 32-bit unsigned integer encoding the maximum time in seconds that pre-authentication state should be retained by the NAS.

2.7. Network-Id-Name

Description

The Network-Id-Name Attribute is utilized by implementations of IEEE-802.1X [IEEE-802.1X] to specify the name of a Network-Id (NID-Name).

Unlike the IEEE 802.11 SSID (which is a maximum of 32 octets in length), the NID-Name may be up to 253 octets in length. Consequently, if the MAC address is included within the Called-Station-Id Attribute, it is possible that there will not be enough remaining space to encode the NID-Name as well. Therefore when used with IEEE 802.1X [IEEE-802.1X], the Called-Station-Id Attribute SHOULD contain only the MAC address, with the Network-Id-Name Attribute used to transmit the NID-Name. The Network-Id-Name Attribute MUST NOT be used to encode the IEEE 802.11 SSID; as

noted in [RFC3580], the Called-Station-Id Attribute is used for this purpose.

Zero or one Network-Id-Name Attribute is permitted within an Access-Request, Access-Challenge, Access-Accept or Accounting-Request packet. When included within an Access-Request packet, the Network-Id-Name Attribute represents a hint of the NID-Name to which the Supplicant should be granted access. When included within an Access-Accept packet, the Network-Id-Name Attribute represents the NID-Name to which the Supplicant is to be granted access. When included within an Accounting-Request packet, the Network-Id-Name Attribute represents the NID-Name to which the Supplicant has been granted access.

A summary of the Network-Id-Name Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      | Length      | String... |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Code

TBD6

Length

>=3

String

The String field is one or more octets, containing a NID-Name. For details, see [IEEE-802.1X]. A robust implementation SHOULD support the field as undistinguished octets.

2.8. EAPoL-Announcement

Description

The Extensible Authentication Protocol over Local Area Network (EAPoL)-Announcement Attribute contains EAPoL-Announcement Type Length Value Tuples (TLVs) defined within Table 11-8 of IEEE-802.1X [IEEE-802.1X].

Zero or more EAPoL-Announcement attributes are permitted within an Access-Request, Access-Accept, Access-Challenge, Access-Reject,

Accounting-Request, CoA-Request or Disconnect-Request packet.

When included within an Access-Request packet, EAPoL-Announcement attributes contain EAPoL-Announcement TLVs that the user sent in an EAPoL-Announcement. When included within an Access-Accept, Access-Challenge, Access-Reject, CoA-Request or Disconnect-Request packet, EAPoL-Announcement attributes contain EAPoL-Announcement TLVs that the NAS is to send to the user in a unicast EAPoL-Announcement. When sent within an Accounting-Request packet, EAPoL-Announcement attributes contain EAPoL-Announcement TLVs that the NAS has most recently sent to the user in a unicast EAPoL-Announcement.

A summary of the EAPoL-Announcement Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |      Length      |      String...      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Code

TBD7

Length

>=3

String

The String field is one or more octets, containing EAPoL-Announcement TLVs in the format defined in Figure 11-8 of Section 11.12 of [IEEE-802.1X]. Any EAPoL-Announcement TLV Type MAY be included within an EAPoL-Announcement Attribute, including Organizationally Specific TLVs. If multiple EAPoL-Announcement attributes are present in a packet, their String fields MUST be concatenated before being parsed for EAPoL-Announcement TLVs; this allows EAPoL-Announcement TLVs longer than 253 octets to be transported by RADIUS. Similarly, EAPoL-Announcement TLVs larger than 253 octets MUST be fragmented between multiple EAPoL-Announcement attributes.

2.9. WLAN-HESSID

Description

The WLAN-HESSID attribute contains a MAC address that identifies the Homogenous Extended Service Set. The HESSID is a globally unique identifier that in conjunction with the SSID, encoded within the Called-Station-Id Attribute as described in [RFC3580], may be used to provide network identification for a subscription service provider network (SSPN), as described in Section 8.4.2.94 of [IEEE-802.11]. Zero or one WLAN-HESSID Attribute is permitted within an Access-Request or Accounting-Request packet.

A summary of the WLAN-HESSID Attribute format is shown below. The fields are transmitted from left to right.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |      Length      |      String...      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Code

TBD8

Length

19

String

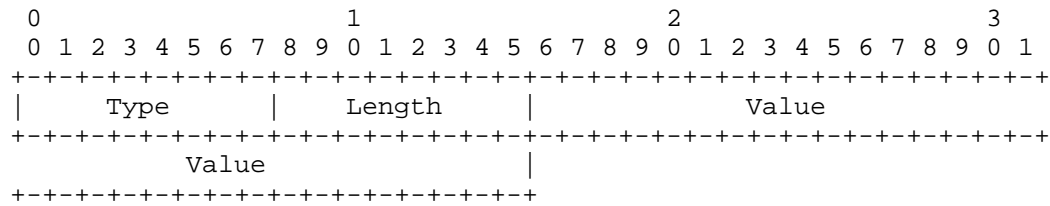
The String field is encoded in upper-case ASCII characters with the octet values separated by dash characters, as described in RFC 3580 [RFC3580]. Example: "00-10-A4-23-19-C0".

2.10. WLAN-Venue-Info

Description

The WLAN-Venue-Info attribute identifies the category of venue hosting the WLAN, as defined in Section 8.4.1.34 of [IEEE-802.11]. Zero or more WLAN-Venue-Info attributes may be included in an Access-Request or Accounting-Request.

A summary of the WLAN-Venue-Info Attribute format is shown below. The fields are transmitted from left to right.



Code

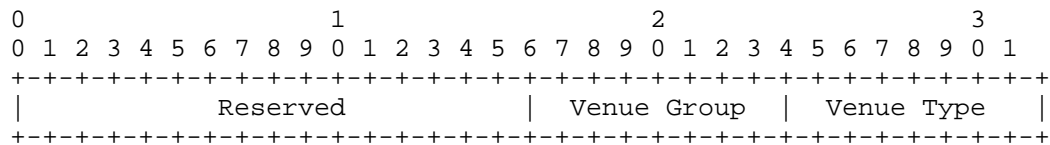
TBD9

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer. The two most significant octets **MUST** be set to zero by the sender, and are ignored by the receiver; the two least significant octets contain the Venue Group and Venue Type fields.



Venue Group

The Venue Group field is a single octet and describes the broad category of the venue, e.g. "Assembly". See Section 8.4.1.34 [IEEE-802.11] for Venue Group codes and descriptions.

Venue Type

The Venue Type field is a single octet and describes the venue in a finer granularity within the Venue Group, e.g. "Library". See Section 8.4.1.34 of [IEEE-802.11] for Venue Type codes and descriptions.

2.11. WLAN-Venue-Language

Description

The WLAN-Venue-Language attribute is an ISO-14962-1997 [ISO-14962-1997] encoded string that defines the language used in

the WLAN-Venue-Name attribute. Zero or more WLAN-Venue-Language attributes may be included in an Access-Request or Accounting-Request and each one indicates the language of the WLAN-Venue-Name attribute that follows it.

A summary of the WLAN-Venue-Language Attribute format is shown below. The fields are transmitted from left to right.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      |      String...      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| String (cont) |
+---+---+---+---+---+

```

Code

TBD10

Length

4-5

String

The String field is a two or three character language code selected from ISO-639 [ISO-639]. A two character language code has a zero ("null" in ISO-14962-1997) appended to make it 3 octets in length.

2.12. WLAN-Venue-Name

Description

The WLAN-Venue-Name attribute provides additional metadata on the BSS. For example, this information may be used to assist a user in selecting the appropriate BSS with which to associate. Zero or more WLAN-Venue-Name attributes may be included in an Access-Request or Accounting-Request in the same or different languages.

A summary of the WLAN-Venue-Name Attribute format is shown below. The fields are transmitted from left to right.

```

      0                               1                               2                               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      |      String...      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Code

TBD11

Length

>=3

String

The String field is a UTF-8 formatted field containing the venue's name. The maximum length of this field is 252 octets.

2.13. WLAN-Reason-Code

Description

The WLAN-Reason-Code Attribute contains information on the reason why a station has been refused network access and has been disassociated or de-authenticated. This can occur due to policy or for reasons related to the user's subscription.

A WLAN-Reason-Code Attribute MAY be included within an Access-Reject or Disconnect-Request packet, as well as within an Accounting-Request packet. Upon receipt of an Access-Reject or Disconnect-Request packet containing a WLAN-Reason-Code Attribute, the WLAN-Reason-Code value is copied by the Access Point into the Reason Code field of a Disassociation or Deauthentication frame (see clause 8.3.3.4 and 8.3.3.12 respectively in [IEEE- 802.11]), which is subsequently transmitted to the station.

A summary of the WLAN-Reason-Code Attribute format is shown below. The fields are transmitted from left to right.

```

      0                               1                               2                               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      |      Value      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Value      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```


Code

TBD12

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer. The two most significant octets MUST be set to zero by the sender, and are ignored by the receiver; the two least significant octets contain the Reason Code values defined in Table 8-36 of Section 8.4.1.7 of [IEEE-802.11].

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Reserved										Reason Code																													

2.14. WLAN-Pairwise-Cipher

Description

The WLAN-Pairwise-Cipher Attribute contains information on the pairwise cipher suite used to establish the robust security network association (RSNA) between the AP and mobile device. A WLAN-Pairwise-Cipher Attribute MAY be included within Access-Request and Accounting-Request packets.

A summary of the WLAN-Pairwise-Cipher Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										Length										Value																			
										Value																													

Code

TBD13

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer, in Suite selector format as specified in Figure 8-187 within Section 8.4.2.27.2 of [IEEE-802.11], with values of OUI and Suite type drawn from Table 8-99.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               OUI                               | Suite Type |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

2.15. WLAN-Group-Cipher

Description

The WLAN-Group-Cipher Attribute contains information on the group cipher suite used to establish the robust security network association (RSNA) between the AP and mobile device. A WLAN-Group-Cipher Attribute MAY be included within Access-Request and Accounting-Request packets.

A summary of the WLAN-Group-Cipher Attribute format is shown below. The fields are transmitted from left to right.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      | Length |                               Value                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               Value                               |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Code

TBD14

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer, in Suite selector format as specified in Figure 8-187

within Section 8.4.2.27.2 of [IEEE-802.11], with values of OUI and Suite type drawn from Table 8-99.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               OUI                               | Suite Type |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

2.16. WLAN-AKM-Suite

Description

The WLAN-AKM-Suite Attribute contains information on the authentication and key management suite used to establish the robust security network association (RSNA) between the AP and mobile device. A WLAN-AKM-Suite Attribute MAY be included within Access-Request and Accounting-Request packets.

A summary of the WLAN-AKM-Suite Attribute format is shown below. The fields are transmitted from left to right.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      | Length |                               Value                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               Value                               |
+-----+-----+-----+-----+-----+-----+

```

Code

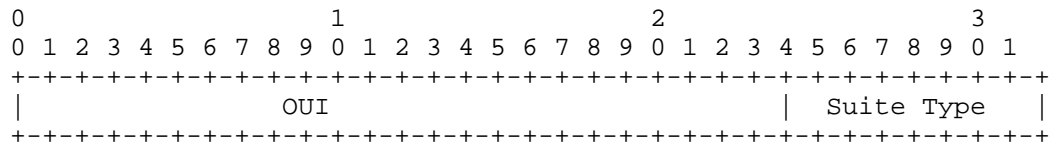
TBD15

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer, in Suite selector format as specified in Figure 8-187 within Section 8.4.2.27.2 of [IEEE-802.11], with values of OUI and Suite type drawn from Table 8-101:



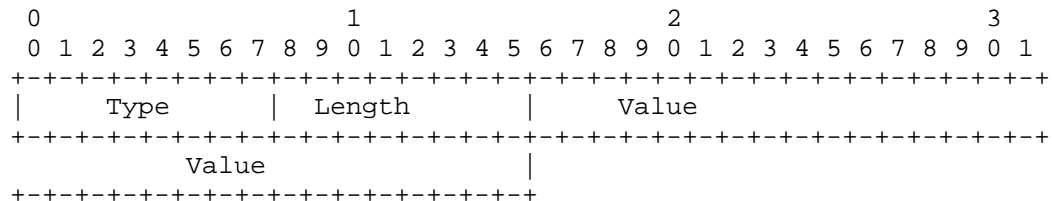
2.17. WLAN-Group-Mgmt-Cipher

Description

The WLAN-Group-Mgmt-Cipher Attribute contains information on group management cipher used to establish the robust security network association (RSNA) between the AP and mobile device.

Zero or one WLAN-Group-Mgmt-Cipher Attribute MAY be included within Access-Request and Accounting-Request packets. Presence of the attribute indicates that the station negotiated to use management frame protection during association.

A summary of the WLAN-Group-Mgmt-Cipher Attribute format is shown below. The fields are transmitted from left to right.



Code

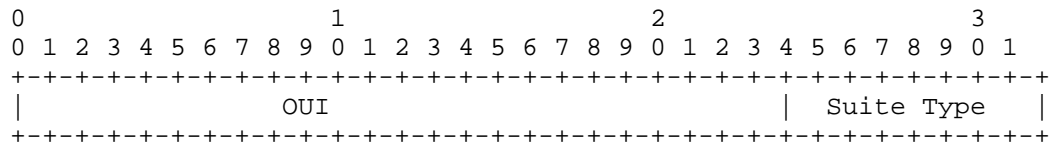
TBD16

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer, in Suite selector format as specified in Figure 8-187 within Section 8.4.2.27.2 of [IEEE-802.11], with values of OUI and Suite type drawn from Table 8-99:

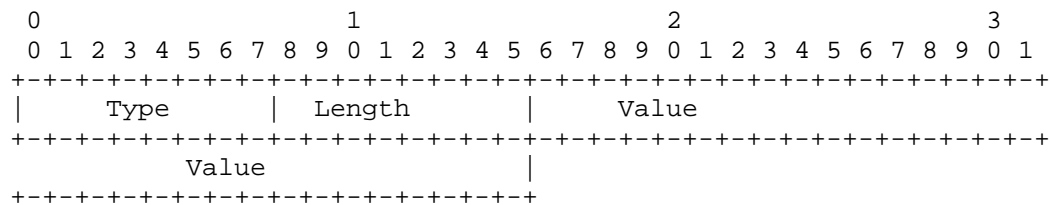


2.18. WLAN-RF-Band

Description

The WLAN-RF-Band Attribute contains information on the RF band used by the Access Point for transmission and reception of information to and from the mobile device. Zero or one WLAN-RF-Band Attribute MAY be included within an Access-Request or Accounting-Request packet.

A summary of the WLAN-RF-Band Attribute format is shown below. The fields are transmitted from left to right.



Code

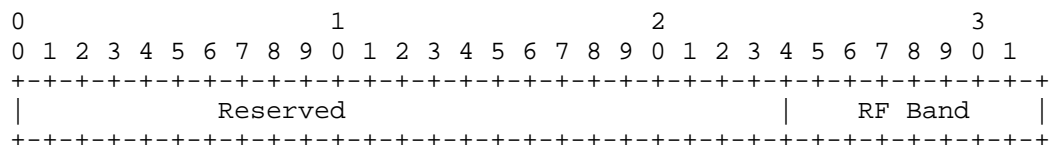
TBD17

Length

6

Value

The Value field is four octets, containing a 32-bit unsigned integer. The three most significant octets MUST be set to zero by the sender, and are ignored by the receiver; the least significant octet contains the RF Band field, whose values are defined in Table 8-53a of [IEEE-802.11ad].



3. Table of attributes

The following table provides a guide to which attributes may be found in which kinds of packets, and in what quantity.

Access-Request	Access-Accept	Access-Reject	Access-Challenge	#	Attribute
0	0+	0	0	TBD1	Allowed-Called-Station-Id
0-1	0-1	0	0	102	EAP-Key-Name
0-1	0+	0	0	TBD2	EAP-Peer-Id
0-1	0+	0	0	TBD3	EAP-Server-Id
0-1	0	0	0	TBD4	Mobility-Domain-Id
0-1	0-1	0	0	TBD5	Preauth-Timeout
0-1	0	0	0	TBD6	Network-Id-Name
0+	0+	0+	0+	TBD7	EAPoL-Announcement
0-1	0	0	0	TBD8	WLAN-HESSID
0-1	0	0	0	TBD9	WLAN-Venue-Info
0+	0	0	0	TBD10	WLAN-Venue-Language
0+	0	0	0	TBD11	WLAN-Venue-Name
0	0	0-1	0	TBD12	WLAN-Reason-Code
0-1	0	0	0	TBD13	WLAN-Pairwise-Cipher
0-1	0	0	0	TBD14	WLAN-Group-Cipher
0-1	0	0	0	TBD15	WLAN-AKM-Suite
0-1	0	0	0	TBD16	WLAN-Group-Mgmt-Cipher
0-1	0	0	0	TBD17	WLAN-RF-Band

CoA-Req	Dis-Req	Acct-Req	#	Attribute
0+	0	0+	TBD1	Allowed-Called-Station-Id
0-1	0	0	102	EAP-Key-Name
0	0	0+	TBD2	EAP-Peer-Id
0	0	0+	TBD3	EAP-Server-Id
0	0	0-1	TBD4	Mobility-Domain-Id
0-1	0	0	TBD5	Preauth-Timeout
0	0	0-1	TBD6	Network-Id-Name
0+	0+	0+	TBD7	EAPoL-Announcement
0	0	0-1	TBD8	WLAN-HESSID
0	0	0-1	TBD9	WLAN-Venue-Info
0	0	0+	TBD10	WLAN-Venue-Language
0	0	0+	TBD11	WLAN-Venue-Name
0	0-1	0-1	TBD12	WLAN-Reason-Code
0	0	0-1	TBD13	WLAN-Pairwise-Cipher
0	0	0-1	TBD14	WLAN-Group-Cipher
0	0	0-1	TBD15	WLAN-AKM-Suite
0	0	0-1	TBD16	WLAN-Group-Mgmt-Cipher
0	0	0-1	TBD17	WLAN-RF-Band

The following table defines the meaning of the above table entries.

- 0 This Attribute MUST NOT be present in packet.
- 0+ Zero or more instances of this Attribute MAY be present in the packet.
- 0-1 Zero or one instance of this Attribute MAY be present in the packet.

4. IANA Considerations

This document uses the RADIUS [RFC2865] namespace, see <http://www.iana.org/assignments/radius-types>. This specification requires assignment of a RADIUS attribute types for the following attributes:

Attribute	Type
=====	=====
Allowed-Called-Station-Id	TBD1
EAP-Peer-Id	TBD2
EAP-Server-Id	TBD3
Mobility-Domain-Id	TBD4
Preauth-Timeout	TBD5
Network-Id-Name	TBD6
EAPoL-Announcement	TBD7
WLAN-HESSID	TBD8
WLAN-Venue-Info	TBD9
WLAN-Venue-Language	TBD10
WLAN-Venue-Name	TBD11
WLAN-Reason-Code	TBD12
WLAN-Pairwise-Cipher	TBD13
WLAN-Group-Cipher	TBD14
WLAN-AKM-Suite	TBD15
WLAN-Group-Mgmt-Cipher	TBD16
WLAN-RF-Band	TBD17

Since this specification relies entirely on values assigned by IEEE 802, no registries are established for maintenance by the IANA.

5. Security Considerations

Since this document describes the use of RADIUS for purposes of authentication, authorization, and accounting in IEEE 802 networks, it is vulnerable to all of the threats that are present in other RADIUS applications. For a discussion of these threats, see [RFC2607], [RFC2865], [RFC3162], [RFC3579], [RFC3580] and [RFC5176]. In particular, when RADIUS traffic is sent in the clear, the attributes defined in this document can be obtained by an attacker snooping the exchange between the RADIUS client and server. As a result, RADIUS confidentiality is desirable; for a review of RADIUS security and crypto-agility requirements, see [RFC6421].

While it is possible for a RADIUS server to make decisions on whether to Accept or Reject an Access-Request based on the values of the WLAN-Pairwise-Cipher, WLAN-Group-Cipher, WLAN-AKM-Suite, WLAN-Group-Mgmt-Cipher and WLAN-RF-Band Attributes the value of doing this is limited. In general, an Access-Reject should not be necessary, except where Access Points and Stations are misconfigured so as to enable connections to be made with unacceptable values. Rather than rejecting access on an ongoing basis, users would be better served by fixing the misconfiguration.

Where access does need to be rejected, the user should be provided with an indication of why the problem has occurred, or else they are likely to become frustrated. For example, if the values of the WLAN-Pairwise-Cipher, WLAN-Group-Cipher, WLAN-AKM-Suite or WLAN-Group-Mgmt-Cipher Attributes included in the Access-Request are not acceptable to the RADIUS server, then a WLAN-Reason-Code Attribute with a value of 29 (Requested service rejected because of service provider cipher suite or AKM requirement) SHOULD be returned in the Access-Reject. Similarly, if the value of the WLAN-RF-Band Attribute included in the Access-Request is not acceptable to the RADIUS server, then a WLAN-Reason-Code Attribute with a value of 11 (Disassociated because the information in the Supported Channels element is unacceptable) SHOULD be returned in the Access-Reject.

6. References

6.1. Normative references

[IEEE-802] IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture, ANSI/IEEE Std 802, 1990.

[IEEE-802.11]
Information technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-2012, 2012.

[IEEE-802.11ad]
Information technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band, IEEE Std. 802.11ad-2012, 2012.

[IEEE-802.1X]
IEEE Standard for Local and Metropolitan Area Networks -

Port-Based Network Access Control, IEEE 802.1X-2010, February 2010.

- [ISO-639] ISO, "Codes for the Representation of Names of Languages".
- [ISO-14962-1997]
ISO, "Space data and information transfer systems - ASCII encoded English", 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997.
- [RFC2865] Rigney, C., Rubens, A., Simpson, W. and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC4072] Eronen, P., Hiller, T. and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.
- [RFC5247] Aboba, B., Simon, D. and P. Eronen, "EAP Key Management Framework", RFC 5247, August 2008.

6.2. Informative references

- [RFC2607] Aboba, B. and J. Vollbrecht, "Proxy Chaining and Policy Implementation in Roaming", RFC 2607, June 1999.
- [RFC3162] Aboba, B., Zorn, G. and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS Support for Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.
- [RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G. and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, September 2003.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [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, January 2008.
- [RFC6421] Nelson, D., "Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS)", RFC 6421,

November 2011.

Acknowledgments

The authors would like to acknowledge Maximilian Riegel, Dorothy Stanley, Yoshihiro Ohba, and the contributors to the IEEE 802.1 and IEEE 802.11 reviews of this document, for useful discussions.

Authors' Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: bernard_aboba@hotmail.com

Jouni Malinen
EMail: j@w1.fi

Paul Congdon
Tallac Networks
6528 Lonetree Blvd.
Rocklin, CA 95765

Phone: +19167576350
EMail: paul.congdon@tallac.com

Joseph Salowey
Cisco Systems
EMail: jsalowey@cisco.com

Mark Jones
Azuca Systems
EMail: mark@azu.ca

RADEXT Working Group
INTERNET-DRAFT
Obsoletes: 4282
Category: Standards Track
<draft-ietf-radext-nai-15.txt>
17 December 2014

DeKok, Alan
FreeRADIUS

The Network Access Identifier
draft-ietf-radext-nai-15

Abstract

In order to provide inter-domain authentication services, it is necessary to have a standardized method that domains can use to identify each other's users. This document defines the syntax for the Network Access Identifier (NAI), the user identifier submitted by the client prior to accessing resources. This document is a revised version of RFC 4282, which addresses issues with international character sets, as well as a number of other corrections to the previous document.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on June 17, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

Appendix A - Changes from RFC4282	3
1. Introduction	4
1.1. Terminology	6
1.2. Requirements Language	7
1.3. Purpose	8
1.4. Motivation	9
2. NAI Definition	10
2.1. UTF-8 Syntax and Normalization	10
2.2. Formal Syntax	11
2.3. NAI Length Considerations	11
2.4. Support for Username Privacy	12
2.5. International Character Sets	13
2.6. The Normalization Process	14
2.6.1. Issues with the Normalization Process	15
2.7. Use in Other Protocols	16
2.8. Using the NAI format for other identifiers	17
3. Routing inside of AAA Systems	18
3.1. Compatibility with Email Usernames	19
3.2. Compatibility with DNS	19
3.3. Realm Construction	20
3.3.1. Historical Practices	21
3.4. Examples	22
4. Security Considerations	23
4.1. Correlation of Identities over Time and Protocols ...	23
4.2. Multiple Identifiers	23
5. Administration of Names	24
6. IANA Considerations	25
7. References	25
7.1. Normative References	25
7.2. Informative References	26
Appendix A - Changes from RFC4282	29

1. Introduction

Considerable interest exists for a set of features that fit within the general category of inter-domain authentication, or "roaming capability" for network access, including dialup Internet users, Virtual Private Network (VPN) usage, wireless LAN authentication, and other applications.

By "inter-domain authentication", this document refers to situations where a user has authentication credentials at one "home" domain, but is able to present them at a second "visited" domain to access certain services at the visited domain. The two domains generally have a pre-existing relationship, so that the credentials can be passed from the visited domain to the home domain for verification. The home domain typically responds with a permit / deny response, which may also include authorization parameters which the visited domain is expected to enforce on the user.

That is, the "roaming" scenario involves a user visiting, or "roaming" to a non-home domain, and requesting the use of services at that visited domain.

Interested parties have included the following:

- * Regional Internet Service Providers (ISPs) operating within a particular state or province, looking to combine their efforts with those of other regional providers to offer dialup service over a wider area.
- * Telecommunications companies who wish to combine their operations with those of one or more companies in another areas or nations, in order to offer more comprehensive network access service in areas where there is no native service. e.g. In another country.
- * Wireless LAN hotspots providing service to one or more ISPs.
- * Businesses desiring to offer their employees a comprehensive package of dialup services on a global basis. Those services may include Internet access as well as secure access to corporate intranets via a VPN, enabled by tunneling protocols such as the Point-to-Point Tunneling Protocol (PPTP) [RFC2637], the Layer 2 Forwarding (L2F) protocol [RFC2341], the Layer 2 Tunneling Protocol (L2TP) [RFC2661], and the IPsec tunnel mode [RFC4301].
- * Other protocols which are interested in leveraging the users credentials in order to take advantage of an existing authentication framework.

In order to enhance the interoperability of these services, it is necessary to have a standardized method for identifying users. This document defines syntax for the Network Access Identifier (NAI). Examples of implementations that use the NAI, and descriptions of its semantics, can be found in [RFC2194].

When the NAI was defined for network access, it had the side effect of defining an identifier which could be used in non-AAA systems. Some non-AAA systems defined identifiers which were compatible with the NAI, and deployments used the NAI. This process simplified the management of credentials, by re-using the same credential in multiple situations. Protocols that re-use the same credential or the same identifier format can benefit from this management simplicity. The alternative is to have protocol-specific credentials or identifier formats, which increases cost to both the user and the administrator.

There are privacy implications to using one identifier across multiple protocols. See Section 2.7 and Section 4 for further discussion of this topic.

The goal of this document is to define the format of an identifier which can be used in many protocols. A protocol may transport an encoded version of the NAI (e.g. '.' as %2E). However, the definition of the NAI is protocol independent. The goal of this document is to encourage the wide-spread adoption of the NAI format. This adoption will decrease work required to leverage identification and authentication in other protocols. It will also decrease the complexity of non-AAA systems for end users and administrators.

This document only suggests that the NAI format be used, but does not require such use. Many protocols already define their own identifier formats. Some of these are incompatible with the NAI, while others allow the NAI in addition to non-NAI identifiers. The definition of the NAI in this document has no requirements on protocol specifications, implementations, or deployments.

However, this document suggests that using one standard identifier format is preferable to using multiple incompatible identifier formats. Where identifiers need to be used in new protocols and/or specifications, it is RECOMMENDED that the format of the NAI be used. That is, the interpretation of the identifier is context-specific, while the format of the identifier remains the same. These issues are discussed in more detail in Section 2.8, below.

The recommendation for a standard identifier format is not a recommendation that each user have one universal identifier. In contrast, this document allows for the use of multiple identifiers,

and recommends the use of anonymous identifiers where those identifiers are publicly visible.

This document is a revised version of [RFC4282], which originally defined internationalized NAIs. Differences and enhancements compared to that document are listed in Appendix A.

1.1. Terminology

This document frequently uses the following terms:

"Local" or "localized" text

Text which is either in non-UTF-8, or in non-normalized form. The character set, encoding, and locale are (in general) unknown to Authentication, Authorization, and Accounting (AAA) network protocols. The client which "knows" the locale may have a different concept of this text than other AAA entities, which do not know the same locale.

Network Access Identifier

The Network Access Identifier (NAI) is a common format for user identifiers submitted by a client during authentication. The purpose of the NAI is to allow a user to be associated with an account name, as well as to assist in the routing of the authentication request across multiple domains. Please note that the NAI may not necessarily be the same as the user's email address or the user identifier submitted in an application layer authentication.

Network Access Server

The Network Access Server (NAS) is the device that clients connect to in order to get access to the network. In PPTP terminology, this is referred to as the PPTP Access Concentrator (PAC), and in L2TP terminology, it is referred to as the L2TP Access Concentrator (LAC). In IEEE 802.11, it is referred to as an Access Point.

Roaming Capability

Roaming capability can be loosely defined as the ability to use any one of multiple Internet Service Providers (ISPs), while maintaining a formal, customer-vendor relationship with only one. Examples of cases where roaming capability might be required include ISP "confederations" and ISP-provided corporate network access support.

Normalization or Canonicalization

These terms are defined in [RFC6365] Section 4. Those definitions are incorporated here by reference.

Locale

This term is defined in [RFC6365] Section 8. Those definitions are incorporated here by reference.

Tunneling Service

A tunneling service is any network service enabled by tunneling protocols such as PPTP, L2F, L2TP, and IPsec tunnel mode. One example of a tunneling service is secure access to corporate intranets via a Virtual Private Network (VPN).

1.2. Requirements Language

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 [RFC2119].

1.3. Purpose

As described in [RFC2194], there are a number of providers offering network access services, and essentially all Internet Service Providers are involved in roaming consortia.

In order to be able to offer roaming capability, one of the requirements is to be able to identify the user's home authentication server. For use in roaming, this function is accomplished via the Network Access Identifier (NAI) submitted by the user to the NAS in the initial network authentication. It is also expected that NASes will use the NAI as part of the process of opening a new tunnel, in order to determine the tunnel endpoint.

This document suggests that other protocols can take advantage of the NAI format. Many protocols include authentication capabilities, including defining their own identifier formats. These identifiers can then end up being transported in AAA protocols, so that the originating protocols can leverage AAA for user authentication. There is therefore a need for a definition of a user identifier which can be used in multiple protocols.

While the NAI is defined herein, it should be noted that existing protocols and deployments do not always use it. AAA systems **MUST** therefore be able to handle user identifiers which are not in the NAI format. The process by which that is done is outside of the scope of this document.

Non-AAA systems can accept user identifiers in forms other than the NAI. This specification does not forbid that practice. It only codifies the format and interpretation of the NAI. This document cannot change existing protocols or practices. It can, however, suggest that using a consistent form for a user identifier is of a benefit to the community.

This document does not make any protocol-specific definitions for an identifier format, and it does not make changes to any existing protocol. Instead, it defines a protocol-independent form for the NAI. It is hoped that the NAI is a user identifier which can be used in multiple protocols.

Using a common identifier format simplifies protocols requiring authentication, as they no longer need to specify protocol-specific format for user identifiers. It increases security, as multiple identifier formats allow attackers to make contradictory claims without being detected (see Section 4.2 for further discussion of this topic). It simplifies deployments, as a user can have one identifier in multiple contexts, which allows them to be uniquely

identified, so long as that identifier is itself protected against unauthorized access.

In short, having a standard is better than having no standard at all.

1.4. Motivation

The changes from [RFC4282] are listed in detail in Appendix A. However, some additional discussion is appropriate to motivate those changes.

The motivation to revise [RFC4282] began with internationalization concerns raised in the context of [EDUROAM]. Section 2.1 of [RFC4282] defines ABNF for realms which limits the realm grammar to English letters, digits, and the hyphen "-" character. The intent appears to have been to encode, compare, and transport realms with the Punycode [RFC3492] encoding form as described in [RFC5891]. There are a number of problems with this approach:

- * The [RFC4282] ABNF is not aligned with internationalization of DNS.
- * The requirement in [RFC4282] Section 2.1 that realms are ASCII conflicts with the Extensible Authentication Protocol (EAP) defined in [RFC3748], and RADIUS, which are both 8-bit clean, and which both recommend the use of UTF-8 for identifiers.
- * [RFC4282] Section 2.4 required mappings that are language-specific, and which are nearly impossible for intermediate nodes to perform correctly without information about that language.
- * [RFC4282] Section 2.4 requires normalization of user names, which may conflict with local system or administrative requirements.
- * The recommendations in [RFC4282] Section 2.4 for treatment of bidirectional characters have proven to be unworkable.
- * The prohibition against use of unassigned code points in [RFC4282] Section 2.4 effectively prohibits support for new scripts.
- * No Authentication, Authorization, and Accounting (AAA) client, proxy, or server has implemented any of the requirements in [RFC4282] Section 2.4, among other sections.

With international roaming growing in popularity, it is important for these issues to be corrected in order to provide robust and inter-

operable network services.

Furthermore, this document was motivated by a desire to codify existing practice related to the use of the NAI format and to encourage widespread use of the format.

2. NAI Definition

2.1. UTF-8 Syntax and Normalization

UTF-8 characters can be defined in terms of octets using the following ABNF [RFC5234], taken from [RFC3629]:

UTF8-xtra-char = UTF8-2 / UTF8-3 / UTF8-4

UTF8-2 = %xC2-DF UTF8-tail

UTF8-3 = %xE0 %xA0-BF UTF8-tail /
 %xE1-EC 2(UTF8-tail) /
 %xED %x80-9F UTF8-tail /
 %xEE-EF 2(UTF8-tail)

UTF8-4 = %xF0 %x90-BF 2(UTF8-tail) /
 %xF1-F3 3(UTF8-tail) /
 %xF4 %x80-8F 2(UTF8-tail)

UTF8-tail = %x80-BF

These are normatively defined in [RFC3629], but are repeated in this document for reasons of convenience.

See [RFC5198] and section 2.6 of this specification for a discussion of normalization. Strings which are not in Normal Form Composed (NFC) are not valid NAIs and SHOULD NOT be treated as such.

Implementations which expect to receive a NAI, but which instead receive non-normalised (but otherwise valid) UTF-8 strings instead SHOULD attempt to create a local version of the NAI, which is normalized from the input identifier. This local version can then be used for local processing. This local version of the identifier MUST NOT be used outside of the local context.

Where protocols carry identifiers which are expected to be transported over an AAA protocol, it is RECOMMENDED that the identifiers be in NAI format. Where the identifiers are not in the NAI format, it is up to the AAA systems to discover this, and to process them. This document does not suggest how that is done. However, existing practice indicates that it is possible.

As internationalized domain names become more widely used, existing practices are likely to become inadequate. This document therefore defines the NAI, which is a user identifier format that can correctly deal with internationalized identifiers.

2.2. Formal Syntax

The grammar for the NAI is given below, described in Augmented Backus-Naur Form (ABNF) as documented in [RFC5234].

```
nai           = utf8-username
nai           =/ "@" utf8-realm
nai           =/ utf8-username "@" utf8-realm

utf8-username = dot-string

dot-string    = string *("." string)
string        = 1*utf8-atext

utf8-atext    = ALPHA / DIGIT /
               "!" / "#" /
               "$" / "%" /
               "&" / "'" /
               "*" / "+" /
               "-" / "/" /
               "=" / "?" /
               "^" / "_" /
               "`" / "{" /
               "|" / "}" /
               "~" /
               UTF8-xtra-char

utf8-realm    = 1*( label "." ) label

label         = utf8-rtext *(ldh-str)
ldh-str       = *( utf8-rtext / "-" ) utf8-rtext
utf8-rtext    = ALPHA / DIGIT / UTF8-xtra-char
```

2.3. NAI Length Considerations

Devices handling NAIs MUST support an NAI length of at least 72 octets. Devices SHOULD support an NAI length of 253 octets. However, the following implementation issues should be considered:

- * NAI octet length constraints may impose a more severe constraint on the number of UTF-8 characters.
- * NAIs are often transported in the User-Name attribute of the

Remote Authentication Dial-In User Service (RADIUS) protocol. Unfortunately, RFC 2865 [RFC2865], Section 5.1, states that "the ability to handle at least 63 octets is recommended." As a result, it may not be possible to transfer NAIs beyond 63 octets through all devices. In addition, since only a single User-Name attribute may be included in a RADIUS message and the maximum attribute length is 253 octets, RADIUS is unable to support NAI lengths beyond 253 octets.

- * NAIs can also be transported in the User-Name attribute of Diameter [RFC6733], which supports content lengths up to $2^{24} - 9$ octets. As a result, NAIs processed only by Diameter nodes can be very long. However, an NAI transported over Diameter may eventually be translated to RADIUS, in which case the above limitations will apply.

- * NAIs may be transported in other protocols. Each protocol can have its own limitations on maximum NAI length. The above criteria should permit the widest use, and widest possible inter-operability of the NAI.

2.4. Support for Username Privacy

Interpretation of the username part of the NAI depends on the realm in question. Therefore, the utf8-username portion SHOULD be treated as opaque data when processed by nodes that are not a part of the home domain for that realm.

That is, the only domain which is capable of interpreting the meaning of the utf8-username portion of the NAI is the home domain. Any third-party domains cannot form any conclusions about the utf8-username, and cannot decode it into sub-fields. For example, it may be used as "firstname.lastname", or it may be entirely digits, or it may be a random hex identifier. There is simply no way (and no reason) for any other domain to interpret the utf8-username field as having any meaning whatsoever.

In some situations, NAIs are used together with a separate authentication method that can transfer the username part in a more secure manner to increase privacy. In this case, NAIs MAY be provided in an abbreviated form by omitting the username part. Omitting the username part is RECOMMENDED over using a fixed username part, such as "anonymous", since including a fixed username part is ambiguous as to whether or not the NAI refers to a single user. However, current practice is to use the username "anonymous" instead of omitting the username part. This behavior is also permitted.

The most common use-case of omitting or obfuscating the username part

is with TLS-based EAP methods such as TTLS [RFC5281]. Those methods allow for an "outer" identifier, which is typically an anonymous "@realm". This outer identifier allows the authentication request to be routed from a visited domain to a home domain. At the same time, the username part is kept confidential from the visited network. The protocol provides for an "inner" authentication exchange, in which a full identifier is used to authenticate a user.

That scenario offers the best of both worlds. An anonymous NAI can be used to route authentication to the home domain, and the home domain has sufficient information to identify and authenticate users.

However, some protocols do not support authenticate methods which allow for "inner" and "outer" exchanges. Those protocols are limited to using an identifier which is publicly visible. It is therefore RECOMMENDED that such protocols use ephemeral identifiers. We recognize that this practice is not currently used, and will likely be difficult to implement.

Similarly to the anonymous user, there may be situations where portions of the realm are sensitive. For those situations, it is RECOMMENDED that the sensitive portion of the realm also be omitted. e.g. To use "@example.com" instead of "@sensitive.example.com", or "anonymous@sensitive.example.com". The home domain is authoritative for users in all subdomains, and can (if necessary) route the authentication request to the appropriate subsystem within the home domain.

For roaming purposes, it is typically necessary to locate the appropriate backend authentication server for the given NAI before the authentication conversation can proceed. As a result, authentication routing is impossible unless the realm portion is available, and in a well-known format.

2.5. International Character Sets

This specification allows both international usernames and realms. International usernames are based on the use of Unicode characters, encoded as UTF-8. Internationalization of the username portion of the NAI is based on the "Internationalized Email Headers" [RFC6532] extensions to the "local-part" portion of email addresses [RFC5322].

In order to ensure a canonical representation, characters of the realm portion in an NAI MUST match the ABNF in this specification as well as the requirements specified in [RFC5891]. In practice, these requirements consist of the following item:

- * Realms MUST be of the form that can be registered as a

Fully Qualified Domain Name (FQDN) within the DNS.

This list is significantly shorter and simpler than the list in Section 2.4 of [RFC4282]. The form suggested in [RFC4282] depended on intermediate nodes performing canonicalizations based on insufficient information, which meant that the form was not canonical.

Specifying the realm requirement as above means that the requirements depend on specifications that are referenced here, rather than copied here. This allows the realm definition to be updated when the referenced documents change, without requiring a revision of this specification.

One caveat on the above recommendation is the issues noted in [RFC6912]. That document notes that there are additional restrictions around DNS registration which forbid some code points from being valid in a DNS U-label. These restrictions cannot be expressed algorithmically.

For this specification, that caveat means the following. Realms not matching the above ABNF are not valid NAIs. However, some realms which do match the ABNF are still invalid NAIs. That is, matching the ABNF is a necessary, but not sufficient, requirement for an NAI.

In general, the above requirement means following the requirements specified in [RFC5891].

2.6. The Normalization Process

Conversion to Unicode as well as normalization SHOULD be performed by edge systems (e.g. laptops, desktops, smart phones, etc.) that take "local" text as input. These edge systems are best suited to determine the users intent, and can best convert from "local" text to a normalized form.

Other AAA systems such as proxies do not have access to locale and character set information that is available to edge systems. Therefore, they may not always be able to convert local input to Unicode.

That is, all processing of NAIs from "local" character sets and locales to UTF-8 SHOULD be performed by edge systems, prior to the NAIs entering the AAA system. Inside of an AAA system, NAIs are sent over the wire in their canonical form, and this canonical form is used for all NAI and/or realm comparisons.

Copying of localized text into fields that can subsequently be placed

into the RADIUS User-Name attribute is problematic. This practice can result in a AAA proxy encountering non-UTF8 characters within what it expects to be an NAI. An example of this requirement is [RFC3579] Section 2.1, which states:

the NAS MUST copy the contents of the Type-Data field of the EAP-Response/Identity received from the peer into the User-Name attribute

As a result, AAA proxies expect the contents of the EAP-Response/Identity sent by an EAP supplicant to consist of UTF-8 characters, not localized text. Using localized text in AAA username or identity fields means that realm routing becomes difficult or impossible.

In contrast to [RFC4282] Section 2.4, AAA systems are now expected to perform NAI comparisons, matching, and AAA routing based on the NAI as it is received. This specification provides a canonical representation, ensures that intermediate AAA systems such as proxies are not required to perform translations, and can be expected to work through AAA systems that are unaware of international character sets.

In an ideal world, the following requirements would be widely implemented:

- * Edge systems using "localized" text SHOULD normalize the NAI prior to it being used as an identifier in an authentication protocol.
- * AAA systems SHOULD NOT normalize the NAI, as they may not have sufficient information to perform the normalization.

There are issues with this approach, however.

2.6.1. Issues with the Normalization Process

The requirements in the preceding section are not implemented today. For example, most EAP implementations use a user identifier which is passed to them from some other local system. This identifier is treated as an opaque blob, and is placed as-is into the EAP Identity field. Any subsequent system which receives that identifier is assumed to be able to understand and process it.

This opaque blob unfortunately can contain localized text, which means that the AAA systems have to process that text.

These limitations have the following theoretical and practical implications.

- * edge systems used today generally do not normalize the NAI

- * Therefore AAA systems SHOULD attempt to normalize the NAI

The suggestion in the above sentence contradicts the suggestion in the previous section. This is the reality of imperfect protocols.

Where the user identifier can be normalized, or determined to be in normal form, the normal form MUST be used as the NAI. In all other circumstances, the user identifier MUST NOT be treated as an NAI. That data is still, however, a user identifier. AAA systems MUST NOT fail authentication simply because the user identifier is not an NAI.

That is, when the realm portion of the NAI is not recognized by an AAA server, it SHOULD try to normalize the NAI into NFC form. That normalized form can then be used to see if the realm matches a known realm. If no match is found, the original form of the NAI SHOULD be used in all subsequent processing.

The AAA server may also convert realms to punycode, and perform all realm comparisons on the resulting punycode strings. This conversion follows the recommendations above, but may have different operational effects and failure modes.

2.7. Use in Other Protocols

As noted earlier, the NAI format can be used in other, non-AAA protocols. It is RECOMMENDED that the definition given here be used unchanged. Using other definitions for user identifiers may hinder interoperability, along with the users ability to authenticate successfully. It is RECOMMENDED that protocols requiring the use of a user identifier use the NAI format.

This document cannot require other protocols to use the NAI format for user identifiers. Their needs are unknown, and at this time unknowable. This document suggests that interoperability and inter-domain authentication is useful, and should be encouraged.

Where a protocol is 8-bit clean, it can likely transport the NAI as-is, without further modification.

Where a protocol is not 8-bit clean, it cannot transport the NAI as-is. Instead, this document presumes that a protocol-specific transport layer takes care of encoding the NAI on input to the protocol, and decoding it when the NAI exits the protocol. The encoded or escaped version of the NAI is not a valid NAI, and MUST NOT be presented to the AAA system.

For example, HTTP carries user identifiers, but escapes the '.' character as "%2E" (among others). When HTTP is used to transport the NAI "fred@example.com", the data as transported will be in the form "fred@example%2Ecom". That data exists only within HTTP, and has no relevance to any AAA system.

Any comparison, validation, or use of the NAI MUST be done on its un-escaped (i.e. utf8-clean) form.

2.8. Using the NAI format for other identifiers

As discussed in Section 1, above, is RECOMMENDED that the NAI format be used as the standard format for user identifiers. This section discusses that use in more detail.

It is often useful to create new identifiers for use in specific contexts. These identifiers may have a number of different properties, most of which are unimportant to this document. The goal of this document is to create identifiers which are to be in a well-known format, and to have namespaces. The NAI format fits these requirements.

One example of such use is the "private user identity", which is an identifier defined by the 3rd-Generation Partnership Project (3GPP). That identifier is used to uniquely identify the user to the network. The identifier is used for authorization, authentication, accounting, administration, etc. The "private user identity" is globally unique, and is defined by the home network operator. The format of the identifier is explicitly the NAI, as stated by Section 13.3 of [3GPP]:

The private user identity shall take the form of an NAI, and shall have the form username@realm as specified in clause 2.1 of IETF RFC 4282

For 3GPP, the "username" portion is a unique identifier which is derived from device-specific information. The "realm" portion is composed of information about the home network, followed by the base string "3gppnetwork.org". e.g.
234150999999999@ims.mnc015.mcc234.3gppnetwork.org.

This format as defined by 3GPP ensures that the identifier is globally unique, as it is based off of the "3gppnetwork.org" domain. It ensures that the "realm" portion is specific to a particular home network (or organization), via the "ims.mnc015.mcc234" prefix to the realm. Finally, it ensures that the "username" portion follows a well-known format.

This document suggests that the NAI format be used for all new specifications and/or protocols where a user identifier is required. Where the username portions need to be created with subfields, a well-known and documented method as has been done with 3GPP is preferred to ad-hoc methods.

3. Routing inside of AAA Systems

Many AAA systems use the "utf8-realm" portion of the NAI to route requests within a AAA proxy network. The semantics of this operation involves a logical AAA routing table, where the "utf8-realm" portion acts as a key, and the values stored in the table are one or more "next hop" AAA servers.

Intermediate nodes **MUST** use the "utf8-realm" portion of the NAI without modification to perform this lookup. As noted earlier, intermediate nodes may not have access to the same locale information as the system which injected the NAI into the AAA routing systems. Therefore, almost all "case insensitive" comparisons can be wrong. Where the "utf8-realm" is entirely ASCII, current AAA systems sometimes perform case-insensitive matching on realms. This method **MAY** be continued, as it has been shown to work in practice.

Many existing non-AAA systems have user identifiers which are similar in format to the NAI, but which are not compliant with this specification. For example, they may use non-NFC form, or they may have multiple "@" characters in the user identifier. Intermediate nodes **SHOULD** normalize non-NFC identifiers to NFC, prior to looking up the "utf8-realm" in the logical routing table. Intermediate nodes **MUST NOT** modify the identifiers that they forward. The data as entered by the user is inviolate.

The "utf8-realm" provisioned in the logical AAA routing table **SHOULD** be provisioned to the proxy prior to it receiving any AAA traffic. The "utf8-realm" **SHOULD** be supplied by the "next hop" or "home" system that also supplies the routing information necessary for packets to reach the next hop.

This "next hop" information may be any of, or all of, the following information: IP address; port; RADIUS shared secret; TLS certificate; DNS host name; or instruction to use dynamic DNS discovery (i.e. look up a record in the "utf8-realm" domain). This list is not exhaustive, and may be extended by future specifications.

It is **RECOMMENDED** to use the entirety of the "utf8-realm" for the routing decisions. However, AAA systems **MAY** use a portion of the "utf8-realm" portion, so long as that portion is a valid "utf8-realm", and that portion is handled as above. For example,

routing "fred@example.com" to a "com" destination is forbidden, because "com" is not a valid "utf8-realm". However, routing "fred@sales.example.com" to the "example.com" destination is permissible.

Another reason to forbid the use of a single label (e.g. "fred@sales") is that many non-AAA systems treat a single label as being a local identifier within their realm. That is, a user logging in as "fred@sales" to a domain "example.com", would be treated as if the NAI was instead "fred@sales.example.com". Permitting the use of a single label would mean changing the interpretation and meaning of a single label, which cannot be done.

3.1. Compatibility with Email Usernames

As proposed in this document, the Network Access Identifier is of the form "user@realm". Please note that while the user portion of the NAI is based on the "Internet Message Format" [RFC5322] "local-part" portion of an email address as extended by "Internationalized Email Headers" [RFC6532], it has been modified for the purposes of Section 2.2. It does not permit quoted text along with "folding" or "non-folding" whitespace that is commonly used in email addresses. As such, the NAI is not necessarily equivalent to usernames used in e-mail.

However, it is a common practice to use email addresses as user identifiers in AAA systems. The ABNF in Section 2.2 is defined to be close to the "addr-spec" portion of [RFC5322] as extended by [RFC6532], while still being compatible with [RFC4282].

In contrast to [RFC4282] Section 2.5, this document state that the internationalization requirements for NAIs and email addresses are substantially similar. The NAI and email identifiers may be the same, and both need to be entered by the user and/or the operator supplying network access to that user. There is therefore good reason for the internationalization requirements to be similar.

3.2. Compatibility with DNS

The "utf8-realm" portion of the NAI is intended to be compatible with Internationalized Domain Names (IDNs) [RFC5890]. As defined above, the "utf8-realm" portion as transported within an 8-bit clean protocol such as RADIUS and EAP can contain any valid UTF8 character. There is therefore no reason for a NAS to convert the "utf8-realm" portion of an NAI into Punycode encoding form [RFC3492] prior to placing the NAI into a RADIUS User-Name attribute.

The NAI does not make a distinction between A-labels and U-labels, as

those are terms specific to DNS. It is instead an IDNA-valid label, as per the first item in Section 2.3.2.1 of [RFC5890]. As noted in that section, the term "IDNA-valid label" encompasses both of the terms A-label and U-label.

When the realm portion of the NAI is used as the basis for name resolution, it may be necessary to convert internationalized realm names to Punycode [RFC3492] encoding form as described in [RFC5891]. As noted in [RFC6055] Section 2, resolver Application Programming Interfaces (APIs) are not necessarily DNS-specific, so conversion to Punycode needs to be done carefully:

Applications which convert an IDN to A-label form before calling (for example) `getaddrinfo()` will result in name resolution failures if the Punycode name is directly used in such protocols. Having libraries or protocols to convert from A-labels to the encoding scheme defined by the protocol (e.g., UTF-8) would require changes to APIs and/or servers, which IDNA was intended to avoid.

As a result, applications SHOULD NOT assume that non-ASCII names are resolvable using the public DNS and blindly convert them to A-labels without knowledge of what protocol will be selected by the name resolution library.

3.3. Realm Construction

The home realm usually appears in the "utf8-realm" portion of the NAI, but in some cases a different realm can be used. This may be useful, for instance, when the home realm is reachable only via intermediate proxies.

Such usage may prevent interoperability unless the parties involved have a mutual agreement that the usage is allowed. In particular, NAIs MUST NOT use a different realm than the home realm unless the sender has explicit knowledge that (a) the specified other realm is available and (b) the other realm supports such usage. The sender may determine the fulfillment of these conditions through a database, dynamic discovery, or other means not specified here. Note that the first condition is affected by roaming, as the availability of the other realm may depend on the user's location or the desired application.

The use of the home realm MUST be the default unless otherwise configured.

3.3.1. Historical Practices

Some AAA systems have historically used NAI modifications with multiple "prefix" and "suffix" decorations to perform explicit routing through multiple proxies inside of a AAA network.

In RADIUS based environment, the use of decorated NAI is NOT RECOMMENDED for the following reasons:

- * Using explicit routing paths is fragile, and is unresponsive to changes in the network due to servers going up or down, or to changing business relationships.
- * There is no RADIUS routing protocol, meaning that routing paths have to be communicated "out of band" to all intermediate AAA nodes, and also to all edge systems (e.g. supplicants) expecting to obtain network access.
- * Using explicit routing paths requires thousands, if not millions of edge systems to be updated with new path information when a AAA routing path changes. This adds huge expense for updates that would be better done at only a few AAA systems in the network.
- * Manual updates to RADIUS paths are expensive, time-consuming, and prone to error.
- * Creating compatible formats for the NAI is difficult when locally-defined "prefixes" and "suffixes" conflict with similar practices elsewhere in the network. These conflicts mean that connecting two networks may be impossible in some cases, as there is no way for packets to be routed properly in a way that meets all requirements at all intermediate proxies.
- * Leveraging the DNS name system for realm names establishes a globally unique name space for realms.

In summary, network practices and capabilities have changed significantly since NAIs were first overloaded to define AAA routes through a network. While manually managed explicit path routing was once useful, the time has come for better methods to be used.

Notwithstanding the above recommendations, the above practice is widely used for Diameter routing [RFC5729]. The routes described there are managed automatically, for both credential provisioning and routing updates. Those routes also exist within a particular framework (typically 3G), where membership is controlled and system behavior is standardized. There are no known issues with using

explicit routing in such an environment.

However, if decorated identifiers are used, such as:

```
homerealm.example.org!user@otherrealm.example.net
```

Then the part before the (non-escaped) '!' MUST be a "utf8-realm" as defined in the ABNF in Section 2.2. When receiving such an identifier, the "otherrealm.example.net" system MUST convert the identifier to "user@homerealm.example.org" before forwarding the request. The forwarding system MUST then apply normal AAA routing for the transaction, based on the updated identifier.

3.4. Examples

Examples of valid Network Access Identifiers include the following:

```
bob
joe@example.com
fred@foo-9.example.com
jack@3rd.depts.example.com
fred.smith@example.com
fred_smith@example.com
fred$@example.com
fred=?#&*+~/^smith@example.com
nancy@eng.example.net
eng.example.net!nancy@example.net
eng%nancy@example.net
@privatecorp.example.net
\ (user\)@example.net
```

An additional valid NAI is the following, given as a hex string, as this document can only contain ASCII characters.

```
626f 6240 ceb4 cebf ceba ceb9 cebc ceae 2e63 6f6d
```

Examples of invalid Network Access Identifiers include the following:

```
fred@example
fred@example_9.com
fred@example.net@example.net
fred.@example.net
eng:nancy@example.net
eng;nancy@example.net
(user)@example.net
<nancy>@example.net
```

One example given in [RFC4282] is still permitted by the ABNF, but it

is NOT RECOMMENDED because of the use of the Punycode [RFC3492] encoding form for what is now a valid UTF-8 string.

alice@xn--tmonesimerkki-bfbb.example.net

4. Security Considerations

Since an NAI reveals the home affiliation of a user, it may assist an attacker in further probing the username space. Typically, this problem is of most concern in protocols that transmit the username in clear-text across the Internet, such as in RADIUS, described in [RFC2865] and [RFC2866]. In order to prevent snooping of the username, protocols may use confidentiality services provided by protocols transporting them, such as RADIUS protected by IPsec [RFC3579] or Diameter protected by TLS [RFC6733].

This specification adds the possibility of hiding the username part in the NAI, by omitting it. As discussed in Section 2.4, this is possible only when NAIs are used together with a separate authentication method that can transfer the username in a secure manner. In some cases, application-specific privacy mechanism have also been used with NAIs. For instance, some EAP methods apply method-specific pseudonyms in the username part of the NAI [RFC3748]. While neither of these approaches can protect the realm part, their advantage over transport protection is that privacy of the username is protected, even through intermediate nodes such as NASes.

4.1. Correlation of Identities over Time and Protocols

The recommendations in Section 2.7 and Section 2.8 for using the NAI in other protocols has implications for privacy. Any attacker who is capable of observing traffic containing the NAI can track the user, and correlate his activity across time and across multiple protocols. The authentication credentials therefore SHOULD be transported over channels which permit private communications, or multiple identifiers SHOULD be used, so that user tracking is impossible.

It is RECOMMENDED that user privacy be enhanced by configuring multiple identifiers for one user. These identifiers can be changed over time, in order to make user tracking more difficult for a malicious observer. However, provisioning and management of the identifiers may be difficult in to do in practice, which is likely why multiple identifiers are rarely used today.

4.2. Multiple Identifiers

Section 1.3 states that multiple identifier formats allow attackers to make contradictory claims without being detected. This statement

deserves further discussion.

Section 2.4 discussed "inner" and "outer" identifiers in the context of TTLS [RFC5281]. A close reading of that specification shows there is no requirement that the inner and outer identifiers be in any way related. That is, it is perfectly valid to use "@example.com" for an outer identifier, and "user@example.org" as an inner identifier. The authentication request will then be routed to "example.com", which will likely be unable to authenticate "user@example.org".

Even worse, a misconfiguration of "example.com" means that it may in turn proxy the inner authentication request to the "example.org" domain. Such cross-domain authentication is highly problematic, and there are few good reasons to allow it.

It is therefore RECOMMENDED that systems which permit anonymous "outer" identifiers require that the "inner" domain be the same as, or a sub-domain of the "outer" domain. An authentication request using disparate realms is a security violation, and the request SHOULD be rejected.

The situation gets worse when multiple protocols are involved. The TTLS protocol permits MS-CHAP [RFC2433] to be carried inside of the TLS tunnel. MS-CHAP defines its own identifier which is encapsulated inside of the MS-CHAP exchange. That identifier is not required to be any particular format, is not required to be in UTF-8, and in practice, can be one of many unknown character sets. There is no way in practice to determine which character set was used for that identifier.

The result is that the "outer" EAP Identity carried by TTLS is likely to not even share the same character set as the "inner" identifier used by MS-CHAP. The two identifiers are entirely independent, and fundamentally incomparable.

Such protocol design is NOT RECOMMENDED.

5. Administration of Names

In order to avoid creating any new administrative procedures, administration of the NAI realm namespace piggybacks on the administration of the DNS namespace.

NAI realm names are required to be unique, and the rights to use a given NAI realm for roaming purposes are obtained coincident with acquiring the rights to use a particular Fully Qualified Domain Name (FQDN). Those wishing to use an NAI realm name should first acquire the rights to use the corresponding FQDN. Administrators MUST NOT

publicly use an NAI realm without first owning the corresponding FQDN. Private use of unowned NAI realms within an administrative domain is allowed, though it is RECOMMENDED that example names be used, such as "example.com".

Note that the use of an FQDN as the realm name does not require use of the DNS for location of the authentication server. While Diameter [RFC6733] supports the use of DNS for location of authentication servers, existing RADIUS implementations typically use proxy configuration files in order to locate authentication servers within a domain and perform authentication routing. The implementations described in [RFC2194] did not use DNS for location of the authentication server within a domain. Similarly, existing implementations have not found a need for dynamic routing protocols or propagation of global routing information. Note also that there is no requirement that the NAI represent a valid email address.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997.

[RFC3629]

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5198]

Klensin J., and Padlipsky M., "Unicode Format for Network Interchange", RFC 5198, March 2008

[RFC5234]

Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008.

[RFC5890]

Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 5890, August 2010

[RFC5891]

Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010

[RFC6365]

Hoffman, P., and Klensin, J., "Terminology Used in Internationalization in the IETF", RFC 6365, September 2011

7.2. Informative References

[RFC2194]

Aboba, B., Lu, J., Alsop, J., Ding, J., and W. Wang, "Review of Roaming Implementations", RFC 2194, September 1997.

[RFC2341]

Valencia, A., Littlewood, M., and T. Kolar, "Cisco Layer Two Forwarding (Protocol) "L2F"", RFC 2341, May 1998.

[RFC2433]

Zorn G., and Cobb, S. "Microsoft PPP CHAP Extensions", RFC 2433, October 1998.

[RFC2637]

Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol", RFC 2637, July 1999.

[RFC2661]

Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, August 1999.

[RFC2865]

Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC2866]

Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

[RFC3492]

Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

[RFC3579]

Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RFC3748]

Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

- [RFC4282]
Aboba, B. et al., "The Network Access Identifier", RFC 4282, December 2005.
- [RFC4301]
Kent, S. and S. Keo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC5281]
Funk, P., and Blake-Wilson, S., "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.
- [RFC5322]
Resnick, P. (Ed), "Internet Message Format", RFC 5322, October 2008.
- [RFC5335]
Y. Abel, Ed., "Internationalized Email Headers", RFC 5335, September 2008.
- [RFC5729]
Korhonen, J. (Ed) et. al., "Clarifications on the Routing of Diameter Requests Based on the Username and the Realm", RFC 5729, December 2009
- [RFC6055]
Thaler, D., et al, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.
- [RFC6532]
Yang, A., et al, "Internationalized Email Headers", RFC 6532, February 2012.
- [RFC6733]
V. Fajardo, Ed., et al, "Diameter Base Protocol", RFC 6733, October 2012.
- [RFC6912]
Sullivan, A., et al, "Principles for Unicode Code Point Inclusion in Labels in the DNS", RFC 6912, April 2013.
- [EDUROAM]
<http://eduroam.org>, "eduroam (EDUcational ROAMing)"
- [3GPP]
3GPP, "TS 23.003 Numbering, addressing, and Identification (Release 12)", July 2014,

ftp://ftp.3gpp.org/Specs/archive/23_series/23.003/.

Acknowledgments

The initial text for this document was [RFC4282], which was then heavily edited. The original authors of [RFC4282] were Bernard Aboba, Mark A. Beadles, Jari Arkko, and Pasi Eronen.

The ABNF validator at <http://www.apps.ietf.org/abnf.html> was used to verify the syntactic correctness of the ABNF in Section 2.

Appendix A - Changes from RFC4282

This document contains the following updates with respect to the previous NAI definition in RFC 4282 [RFC4282]:

- * The formal syntax in Section 2.1 has been updated to forbid non-UTF8 characters. e.g. characters with the "high bit" set.
- * The formal syntax in Section 2.1 has been updated to allow UTF-8 in the "realm" portion of the NAI.
- * The formal syntax in [RFC4282] Section 2.1 applied to the NAI after it was "internationalized" via the ToAscii function. The contents of the NAI before it was "internationalized" were left indeterminate. This document updates the formal syntax to define an internationalized form of the NAI, and forbids the use of the ToAscii function for NAI "internationalization".
- * The grammar for the user and realm portion is based on a combination of the "nai" defined in [RFC4282] Section 2.1, and the "utf8-addr-spec" defined in [RFC5335] Section 4.4.
- * All use of the ToAscii function has been moved to normal requirements on DNS implementations when realms are used as the basis for DNS lookups. This involves no changes to the existing DNS infrastructure.
- * The discussions on internationalized character sets in Section 2.4 have been updated. The suggestion to use the ToAscii function for realm comparisons has been removed. No AAA system has implemented these suggestions, so this change should have no operational impact.
- * The section "Routing inside of AAA Systems" section is new in this document. The concept of a "local AAA routing table" is also new, although it accurately describes the functionality of wide-spread implementations.
- * The "Compatibility with EMail Usernames" and "Compatibility with DNS" sections have been revised and updated. The Punycode transformation is suggested to be used only when a realm name is used for DNS lookups, and even then the function is only used by a resolving API on the local system, and even then it is recommended that only the home network perform this conversion.
- * The "Realm Construction" section has been updated to note that editing of the NAI is NOT RECOMMENDED.

- * The "Examples" section has been updated to remove the instance of the IDN being converted to ASCII. This behavior is now forbidden.

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org

Network Working Group
INTERNET-DRAFT
Category: Proposed Standard
Updates: 2865, 2866, 3575, 5176, 6158
<draft-ietf-radext-radius-extensions-13.txt>
Expires: August 25, 2013
25 February 2013

Alan DeKok
Network RADIUS
Avi Lior

Remote Authentication Dial In User Service (RADIUS) Protocol
Extensions
draft-ietf-radext-radius-extensions-13.txt

Abstract

The Remote Authentication Dial In User Service (RADIUS) protocol is nearing exhaustion of its current 8-bit Attribute Type space. In addition, experience shows a growing need for complex grouping, along with attributes which can carry more than 253 octets of data. This document defines changes to RADIUS which address all of the above problems.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on June 26, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info/>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	5
1.1.	Caveats and Limitations	6
1.1.1.	Failure to Meet Certain Goals	6
1.1.2.	Implementation Recommendations	6
1.2.	Terminology	7
1.3.	Requirements Language	8
2.	Extensions to RADIUS	9
2.1.	Extended Type	10
2.2.	Long Extended Type	11
2.3.	TLV Data Type	14
2.3.1.	TLV Nesting	16
2.4.	EVS Data Type	16
2.5.	Integer64 Data Type	18
2.6.	Vendor-ID Field	18
2.7.	Attribute Naming and Type Identifiers	19
2.7.1.	Attribute and TLV Naming	19
2.7.2.	Attribute Type Identifiers	19
2.7.3.	TLV Identifiers	20
2.7.4.	VSA Identifiers	20
2.8.	Invalid Attributes	21
3.	Attribute Definitions	22
3.1.	Extended-Type-1	23
3.2.	Extended-Type-2	23
3.3.	Extended-Type-3	24
3.4.	Extended-Type-4	25
3.5.	Long-Extended-Type-1	26
3.6.	Long-Extended-Type-2	27
4.	Vendor Specific Attributes	28
4.1.	Extended-Vendor-Specific-1	29
4.2.	Extended-Vendor-Specific-2	30
4.3.	Extended-Vendor-Specific-3	31
4.4.	Extended-Vendor-Specific-4	32
4.5.	Extended-Vendor-Specific-5	33
4.6.	Extended-Vendor-Specific-6	35
5.	Compatibility with traditional RADIUS	36
5.1.	Attribute Allocation	36
5.2.	Proxy Servers	37
6.	Guidelines	38
6.1.	Updates to RFC 6158	38
6.2.	Guidelines for Simple Data Types	38
6.3.	Guidelines for Complex Data Types	39
6.4.	Design Guidelines For the New Types	40
6.5.	TLV Guidelines	41
6.6.	Allocation Request Guidelines	41
6.7.	Allocation Requests Guidelines for TLVs	42
6.8.	Implementation Guidelines	43

6.9. Vendor Guidelines	43
7. Rationale for This Design	43
7.1. Attribute Audit	44
8. Diameter Considerations	45
9. Examples	45
9.1. Extended Type	46
9.2. Long Extended Type	47
10. IANA Considerations	50
10.1. Attribute Allocations	50
10.2. RADIUS Attribute Type Tree	50
10.3. Allocation Instructions	51
10.3.1. Requested Allocation from the Standard Space ..	52
10.3.2. Requested Allocation from the short extended sp	52
10.3.3. Requested Allocation from the long extended spa	52
10.3.4. Allocation Preferences	52
10.3.5. Extending the Type Space via TLV Data Type	53
10.3.6. Allocation within a TLV	53
10.3.7. Allocation of Other Data Types	54
11. Security Considerations	54
12. References	54
12.1. Normative references	54
12.2. Informative references	55
Appendix A - Extended Attribute Generator Program	56

1. Introduction

Under current allocation pressure, we expect that the RADIUS Attribute Type space will be exhausted by 2014 or 2015. We therefore need a way to extend the type space, so that new specifications may continue to be developed. Other issues have also been shown with RADIUS. The attribute grouping method defined in [RFC2868] has been shown to be impractical, and a more powerful mechanism is needed. Multiple attributes have been defined which transport more than the 253 octets of data originally envisioned with the protocol. Each of these attributes is handled as a "special case" inside of RADIUS implementations, instead of as a general method. We therefore also need a standardized method of transporting large quantities of data. Finally, some vendors are close to allocating all of the Attributes within their Vendor-Specific Attribute space. It would be useful to leverage changes to the base protocol for extending the Vendor-Specific Attribute space.

We satisfy all of these requirements through the following changes given in this document:

- * defining an "Extended Type" format, which adds 8 bits of "Extended Type" to the RADIUS Attribute Type space, by using one octet of the "Value" field. This method gives us a general way of extending the Attribute Type Space. (Section 2.1)
- * allocating 4 attributes as using the format of "Extended Type". This allocation extends the RADIUS Attribute Type Space by approximately 1000 values. (Sections 3.1, 3.2, 3.3, and 3.4)
- * defining a "Long Extended Type" format, which inserts an additional octet between the "Extended Type" octet, and the "Value" field. This method gives us a general way of adding additional functionality to the protocol. (Section 2.2)
- * defining a method which uses the additional octet in the "Long Extended Type" to indicate data fragmentation across multiple Attributes. This method provides a standard way for an Attribute to carry more than 253 octets of data. (Section 2.2)
- * allocating 2 attributes as using the format "Long Extended Type". This allocation extends the RADIUS Attribute Type Space by an additional 500 values. (Sections 3.5 and 3.6)
- * defining a new "Type Length Value" (TLV) data type. The data type allows an attribute to carry TLVs as "sub-attributes", which can in turn encapsulate other TLVs as "sub-sub-attributes." This change creates a standard way to group a set of Attributes. (Section 2.3)

- * defining a new "extended Vendor-Specific" (EVS) data type. The data type allows an attribute to carry Vendor-Specific Attributes (VSAs) inside of the new attribute formats. (Section 2.4)
- * defining a new "integer64" data type. The data type allows counters which track more than 2^{32} octets of data. (Section 2.5)
- * allocating 6 attributes using the new EVS data type. This allocation extends the Vendor-Specific Attribute Type space by over 1500 values. (Sections 4.1 through 4.6)
- * Define the "Vendor-ID" for Vendor-Specific attributes to encompass the entire 4 octets of the Vendor field. [RFC2865] Section 5.26 defined it to be 3 octets, with the high octet being zero. (Section 2.5)
- * Describing compatibility with existing RADIUS systems. (Section 5)
- * Defining guidelines for the use of these changes for IANA, implementations of this specification, and for future RADIUS specifications. (Section 6)

As with any protocol change, the changes defined here are the result of a series of compromises. We have tried to find a balance between flexibility, space in the RADIUS message, compatibility with existing deployments, and implementation difficulty.

1.1. Caveats and Limitations

This section describes some caveats and limitations with the proposal.

1.1.1. Failure to Meet Certain Goals

One goal which was not met by the above modifications is to have an incentive for standards to use the new space. That incentive is being provided by the exhaustion of the standard space.

1.1.2. Implementation Recommendations

It is RECOMMENDED that implementations support this specification. It is RECOMMENDED that new specifications use the formats defined in this specification.

The alternative to the above recommendations is a circular argument of not implementing this specification because no other standards reference it, and also not defining new standards referencing this specification because no implementations exist.

As noted earlier, the "standard space" is almost entirely allocated. Ignoring the looming crisis benefits no one.

1.2. Terminology

This document uses the following terms:

Silently discard

This means the implementation discards the packet without further processing. The implementation MAY provide the capability of logging the error, including the contents of the silently discarded packet, and SHOULD record the event in a statistics counter.

Invalid attribute

This means that the Length field of an Attribute is valid (as per [RFC2865], Section 5, top of page 25), but the contents of the Attribute do not follow the correct format. For example, an Attribute of type "address" which encapsulates more than four, or less than four, octets of data. See Section 2.8 for a more complete definition.

Standard space

Codes in the RADIUS Attribute Type Space that are allocated by IANA and that follow the format defined in Section 5 of [RFC2865].

Extended space

Codes in the RADIUS Attribute Type Space that require the extensions defined in this document, and are an extension of the standard space, but which cannot be represented within the standard space.

Short extended space

Codes in the extended space which use the "Extended Type" format.

Long extended space

Codes in the extended space which use the "Long Extended Type" format.

The following terms are used here with the meanings defined in BCP 26 [RFC5226]: "name space", "assigned value", "registration", "Private Use", "Reserved", "Unassigned", "IETF Review", and "Standards Action".

1.3. Requirements Language

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Extensions to RADIUS

This section defines two new attribute formats; "Extended Type"; and "Long Extended Type". It defines a new Type-Length-Value (TLV) data type, an Extended-Vendor-Specific (EVS) data type, and an Integer64 data type. It defines a new method for naming attributes and identifying Attributes using the new attribute formats. It finally defines the new term "invalid attribute", and describes how it affects implementations.

The new attribute formats are designed to be compatible with the attribute format given in [RFC2865] Section 5. The meaning and interpretation of the Type and Length fields is unchanged from that specification. This reuse allows the new formats to be compatible with RADIUS implementations which do not implement this specification. Those implementations can simply ignore the Value field of an attribute, or forward it verbatim.

The changes to the attribute format come about by "stealing" one or more octets from the Value field. This change has the effect that the Value field of [RFC2865] Section 5 contains both the new octets given here, and any attribute-specific Value. The result is that Values in this specification are limited to less than 253 octets in size. This limitation is overcome through the use of the "Long Extended Type" format.

We reiterate that the formats given in this document do not insert new data into an attribute. Instead, we "steal" one octet of Value, so that the definition of the Length field remains unchanged. The new attribute formats are designed to be compatible with the attribute format given in [RFC2865] Section 5. The meaning and interpretation of the Type and Length fields is unchanged from that specification. This reuse allows the new formats to be compatible RADIUS implementations which do not implement this specification. Those implementations can simply ignore the Value field of an attribute, or forward it verbatim.

The changes to the attribute format come about by "stealing" one or more octets from the Value field. This change has the effect that the Value field of [RFC2865] Section 5 contains both the new octets given here, and any attribute-specific Value. The result is that Values in this specification are limited to less than 253 octets in size. This limitation is overcome through the use of the "Long Extended Type" format.

2.1. Extended Type

This section defines a new attribute format, called "Extended Type". A summary of the Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      | Extended-Type | Value ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

This field is identical to the Type field of the Attribute format defined in [RFC2865] Section 5.

Length

The Length field is one octet, and indicates the length of this Attribute including the Type, Length, Extended-Type, and Value fields. Permitted values are between 4 and 255. If a client or server receives an Extended Attribute with a Length of 2 or 3, then that Attribute MUST be considered to be an "invalid attribute", and handled as per Section 2.8, below.

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified according to the policies and rules described in Section 10. Unlike the Type field defined in [RFC2865] Section 5, no values are allocated for experimental or implementation-specific use. Values 241-255 are reserved and MUST NOT be used.

The Extended-Type is meaningful only within a context defined by the Type field. That is, this field may be thought of as defining a new type space of the form "Type.Extended-Type". See Section 2.5, below, for additional discussion.

A RADIUS server MAY ignore Attributes with an unknown "Type.Extended-Type".

A RADIUS client MAY ignore Attributes with an unknown "Type.Extended-Type".

Value

This field is similar to the Value field of the Attribute format

defined in [RFC2865] Section 5. The format of the data MUST be a valid RADIUS data type.

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

The addition of the Extended-Type field decreases the maximum length for attributes of type "text" or "string" from 253 to 252 octets. Where an Attribute needs to carry more than 252 octets of data, the "Long Extended Type" format MUST be used.

Experience has shown that the "experimental" and "implementation specific" attributes defined in [RFC2865] Section 5 have had little practical value. We therefore do not continue that practice here with the Extended-Type field.

2.2. Long Extended Type

This section defines a new attribute format, called "Long Extended Type". It leverages the "Extended Type" format in order to permit the transport of attributes encapsulating more than 253 octets of data. A summary of the Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
Type										Length										Extended-Type										M										Reserved									
Value ...																																																	

Type

This field is identical to the Type field of the Attribute format defined in [RFC2865] Section 5.

Length

The Length field is one octet, and indicates the length of this Attribute including the Type, Length, Extended-Type, and Value fields. Permitted values are between 5 and 255. If a client or server receives a "Long Extended Type" with a Length of 2, 3, or 4, then that Attribute MUST be considered to be an "invalid attribute", and be handled as per Section 2.8, below.

Note that this Length is limited to the length of this fragment. There is no field which gives an explicit value for the total size of the fragmented attribute.

Extended-Type

This field is identical to the Extended-Type field defined above in Section 2.1.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. The More field MUST be clear (0) if the Length field has value less than 255. The More field MAY be set (1) if the Length field has value of 255.

If the More field is set (1), it indicates that the Value field has been fragmented across multiple RADIUS attributes. When the More field is set (1), the attribute MUST have a Length field of value 255; there MUST be an attribute following this one; and the next attribute MUST have both the same Type and Extended Type. That is, multiple fragments of the same value MUST be in order and MUST be consecutive attributes in the packet, and the last attribute in a packet MUST NOT have the More field set (1).

That is, a packet containing a fragmented attribute needs to contain all fragments of the attribute, and those fragments need to be contiguous in the packet. RADIUS does not support inter-packet fragmentation, which means that fragmenting an attribute across multiple packets is impossible.

If a client or server receives an attribute fragment with the "More" field set (1), but for which no subsequent fragment can be found, then the fragmented attribute is considered to be an "invalid attribute", and handled as per Section 2.8, below.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Future specifications may define additional meaning for this field. Implementations therefore MUST NOT treat this field as invalid if it is non-zero.

Value

This field is similar to the Value field of the Attribute format defined in [RFC2865] Section 5. It may contain a complete set of data (when the Length field has value less than 255), or it may contain a fragment of data.

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

Any interpretation of the resulting data MUST occur after the fragments have been reassembled. The length of the data MUST be taken as the sum of the lengths of the fragments (i.e. Value fields) from which it is constructed. The format of the data SHOULD be a valid RADIUS data type. If the reassembled data does not match the expected format, all fragments MUST be treated as "invalid attributes", and the reassembled data MUST be discarded.

We note that the maximum size of a fragmented attribute is limited only by the RADIUS packet length limitation (i.e. 4096 octets, not counting various headers and overhead). Implementations MUST be able to handle the case where one fragmented attribute completely fills the packet.

This definition increases the RADIUS Attribute Type space as above, but also provides for transport of Attributes which could contain more than 253 octets of data.

Note that [RFC2865] Section 5 says:

If multiple Attributes with the same Type are present, the order of Attributes with the same Type MUST be preserved by any proxies. The order of Attributes of different Types is not required to be preserved. A RADIUS server or client MUST NOT have any dependencies on the order of attributes of different types. A RADIUS server or client MUST NOT require attributes of the same type to be contiguous.

These requirements also apply to the "Long Extended Type" attribute, including fragments. Implementations MUST be able to process non-contiguous fragments -- that is, fragments which are mixed together with other attributes of a different Type. This will allow them to accept packets, so long as the attributes can be correctly decoded.

2.3. TLV Data Type

We define a new data type in RADIUS, called "tlv". The "tlv" data type is an encapsulation layer which permits the "Value" field of an Attribute to contain new sub-Attributes. These sub-Attributes can in turn contain "Value"s of data type TLV. This capability both extends the attribute space, and permits "nested" attributes to be used. This nesting can be used to encapsulate or group data into one or more logical containers.

The "tlv" data type re-uses the RADIUS attribute format, as given below:

```

      0                               1                               2                               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|  TLV-Type   |  TLV-Length   |  TLV-Value ...  |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

TLV-Type

The Type field is one octet. Up-to-date values of this field are specified according to the policies and rules described in Section 10. Values 254-255 are "Reserved" for use by future extensions to RADIUS. The value 26 has no special meaning, and MUST NOT be treated as a Vendor Specific attribute.

As with Extended-Type above, the TLV-Type is meaningful only within the context defined by "Type" fields of the encapsulating Attributes. That is, the field may be thought of as defining a new type space of the form "Type.Extended-Type.TLV-Type". Where TLVs are nested, the type space is of the form "Type.Extended-Type.TLV-Type.TLV-Type", etc.

A RADIUS server MAY ignore Attributes with an unknown "TLV-Type".

A RADIUS client MAY ignore Attributes with an unknown "TLV-Type".

A RADIUS proxy SHOULD forward Attributes with an unknown "TLV-Type" verbatim.

TLV-Length

The TLV-Length field is one octet, and indicates the length of this TLV including the TLV-Type, TLV-Length and TLV-Value fields. It MUST have a value between 3 and 255. If a client or server receives a TLV with an invalid TLV-Length, then the attribute which encapsulates that TLV MUST be considered to be an "invalid

attribute", and handled as per Section 2.8, below.

TLV-Value

The TLV-Value field is one or more octets and contains information specific to the Attribute. The format and length of the TLV-Value field is determined by the TLV-Type and TLV-Length fields.

The TLV-Value field SHOULD encapsulate a standard RADIUS data type. Non-standard data types SHOULD NOT be used within TLV-Value fields. We note that the TLV-Value field MAY also contain one or more attributes of data type "tlv", which allows for simple grouping and multiple layers of nesting.

The TLV-Value field is limited to containing 253 or fewer octets of data. Specifications which require a TLV to contain more than 253 octets of data are incompatible with RADIUS, and need to be redesigned. Specifications which require the transport of empty Values (i.e. Length = 2) are incompatible with RADIUS, and need to be redesigned.

The TLV-Value field MUST NOT contain data using the "Extended Type" formats defined in this document. The base Extended Attributes format allows for sufficient flexibility that nesting them inside of a TLV offers little additional value.

This TLV definition is compatible with the suggested format of the "String" field of the Vendor-Specific attribute, as defined in [RFC2865] Section 5.26, though that specification does not discuss nesting.

Vendors MAY use attributes of type "tlv" in any Vendor Specific Attribute. It is RECOMMENDED to use type "tlv" for VSAs, in preference to any other format.

If multiple TLVs with the same TLV-Type are present, the order of TLVs with the same TLV-Type MUST be preserved by any proxies. The order of TLVs of different TLV-Types is not required to be preserved. A RADIUS server or client MUST NOT have any dependencies on the order of TLVs of different TLV-Types. A RADIUS server or client MUST NOT require TLVs of the same TLV-type to be contiguous.

The interpretation of multiple TLVs of the same TLV-Type MUST be that of a logical "and", unless otherwise specified. That is, multiple TLVs are interpreted as specifying an unordered set of values. Specifications SHOULD NOT define TLVs to be interpreted as a logical "or". Doing so would mean that a RADIUS client or server would make an arbitrary, and non-deterministic choice among the values.

2.3.1. TLV Nesting

TLVs may contain other TLVs. When this occurs, the "container" TLV MUST be completely filled by the "contained" TLVs. That is, the "container" TLV-Length field MUST be exactly two (2) more than the sum of the "contained" TLV-Length fields. If the "contained" TLVs over-fill the "container" TLV, the "container" TLV MUST be considered to be an "invalid attribute", and handled as described in Section 2.8, below.

The depth of TLV nesting is limited only by the restrictions on the TLV-Length field. The limit of 253 octets of data results in a limit of 126 levels of nesting. However, nesting depths of more than 4 are NOT RECOMMENDED. They have not been demonstrated to be necessary in practice, and they appear to make implementations more complex. Reception of packets with such deeply nest TLVs may indicate implementation errors or deliberate attacks. Where implementations do not support deep nesting of TLVs, it is RECOMMENDED that the unsupported layers are treated as "invalid attributes".

2.4. EVS Data Type

We define a new data type in RADIUS, called "evs", for "Extended Vendor-Specific". The "evs" data type is an encapsulation layer which permits the "Value" field of an Attribute to contain a Vendor-Id, followed by a Vendor-Type, and then vendor-defined data. This data can in turn contain valid RADIUS data types, or any other data as determined by the vendor.

This data type is intended use in attributes which carry Vendor-Specific information, as is done with the Vendor-Specific Attribute (26). It is RECOMMENDED that this data type be used by a vendor only when the Vendor-Specific Attribute Type space has been fully allocated.

Where [RFC2865] Section 5.26 makes a recommendation for the format of the data following the Vendor-Id, we give a strict definition. Experience has shown that many vendors have not followed the [RFC2865] recommendations, leading to interoperability issues. We hope here to give vendors sufficient flexibility as to meet their needs, while minimizing the use of non-standard VSA formats.

The "evs" data type MAY be used in Attributes having the format of "Extended Type" or "Long Extended Type". It MUST NOT be used in any other Attribute definition, including standard RADIUS Attributes, TLVs, and VSAs.

A summary of the "evs" data type format is shown below. The fields

are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                           Vendor-Id                                           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Vendor-Type | Vendor-Value .... |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

Vendor-Value

The Vendor-Value field is one or more octets. It SHOULD encapsulate a standard RADIUS data type. Using non-standard data types is NOT RECOMMENDED. We note that the Value field may be of data type "tlv". However, it MUST NOT be of data type "evs", as the use cases are unclear for one vendor delegating Attribute Type space to another vendor.

The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets. We recognise that Vendors have complete control over the contents and format of the Value field, while at the same time recommending that good practices be followed.

Further codification of the range of allowed usage of this field is outside the scope of this specification.

Note that unlike the format described in [RFC2865] Section 5.26, this data type has no "Vendor length" field. The length of the Vendor-Value field is implicit, and is determined by taking the "Length" of the encapsulating RADIUS Attribute, and subtracting the length of the attribute header (2 octets), the extended type (1 octet), the Vendor-Id (4 octets), and the Vendor-type (1 octet). i.e. For "Extended Type" attributes, the length of the Vendor-Value field is eight (8) less than the value of the Length field. For "Long Extended Type" attributes, the length of the Vendor-Value field is nine (9) less than the value of the Length field.

2.5. Integer64 Data Type

We define a new data type in RADIUS, called "integer64", which carries a 64-bit unsigned integer in network byte order.

This data type is intended to be used in any situation where there is a need to have counters which can count past 2^{32} . The expected use of this data type is within Accounting-Request packets, but this data type SHOULD be used in any packet where 32-bit integers are expected to be insufficient.

The "integer64" data type can be used in Attributes of any format, standard space, extended attributes, TLVs, and VSAs.

A summary of the "integer64" data type format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Value ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                                                                                                                 |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Attributes having data type "integer64" MUST have the relevant Length field set to eight more than the length of the Attribute header. For standard space Attributes and TLVs, this means that the Length field MUST be set to ten (10). For "Extended Type" Attributes, the Length field MUST be set to eleven (11). For "Long Extended Type" Attributes, the Length field MUST be set to twelve (12).

2.6. Vendor-ID Field

We define the Vendor-ID field of Vendor-Specific Attributes to encompass the entire 4 octets of the Vendor field. [RFC2865] Section 5.26 defined it to be 3 octets, with the high octet being zero. This change has no immediate impact on RADIUS, as the maximum Private Enterprise Code defined is still within 16 bits.

However, it is best to make advance preparations for changes in the protocol. As such, it is RECOMMENDED that all implementations support four (4) octets for the Vendor-ID field, instead of three (3).

2.7. Attribute Naming and Type Identifiers

Attributes have traditionally been identified by a unique name and number. For example, the attribute named "User-Name" has been allocated number one (1). This scheme needs to be extended in order to be able to refer to attributes of Extended Type, and to TLVs. It will also be used by IANA for allocating RADIUS Attribute Type values.

The names and identifiers given here are intended to be used only in specifications. The system presented here may not be useful when referring to the contents of a RADIUS packet. It imposes no requirements on implementations, as implementations are free to reference RADIUS Attributes via any method they choose.

2.7.1. Attribute and TLV Naming

RADIUS specifications traditionally use names consisting of one or more words, separated by hyphens, e.g. "User-Name". However, these names are not allocated from a registry, and there is no restriction other than convention on their global uniqueness.

Similarly, vendors have often used their company name as the prefix for VSA names, though this practice is not universal. For example, for a vendor named "Example", the name "Example-Attribute-Name" SHOULD be used instead of "Attribute-Name". The second form can conflict with attributes from other vendors, whereas the first form cannot.

It is therefore RECOMMENDED that specifications give names to Attributes which attempt to be globally unique across all RADIUS Attributes. It is RECOMMENDED that vendors use their name as a unique prefix for attribute names, e.g. Livingston-IP-Pool instead of IP-Pool. It is RECOMMENDED that implementations enforce uniqueness on names, which would otherwise lead to ambiguity and problems.

We recognise that these suggestions may sometimes be difficult to implement in practice.

TLVs SHOULD be named with a unique prefix that is shared among related attributes. For example, a specification that defines a set of TLVs related to time could create attributes named "Time-Zone", "Time-Day", "Time-Hour", "Time-Minute", etc.

2.7.2. Attribute Type Identifiers

The RADIUS Attribute Type space defines a context for a particular "Extended-Type" field. The "Extended-Type" field allows for 256

possible type code values, with values 1 through 240 available for allocation. We define here an identification method that uses a "dotted number" notation similar to that used for Object Identifiers (OIDs), formatted as "Type.Extended-Type".

For example, an attribute within the Type space of 241, having Extended-Type of one (1), is uniquely identified as "241.1". Similarly, an attribute within the Type space of 246, having Extended-Type of ten (10), is uniquely identified as "246.10".

2.7.3. TLV Identifiers

We can extend the Attribute reference scheme defined above for TLVs. This is done by leveraging the "dotted number" notation. As above, we define an additional TLV type space, within the "Extended Type" space, by appending another "dotted number" in order to identify the TLV. This method can be repeated in sequence for nested TLVs.

For example, let us say that "245.1" identifies RADIUS Attribute Type 245, containing an "Extended Type" of one (1), which is of type "tlv". That attribute will contain 256 possible TLVs, one for each value of the TLV-Type field. The first TLV-Type value of one (1) can then be identified by appending a ".1" to the number of the encapsulating attribute ("241.1"), to yield "241.1.1". Similarly, the sequence "245.2.3.4" identifies RADIUS attribute 245, containing an "Extended Type" of two (2) which is of type "tlv", which in turn contains a TLV with TLV-Type number three (3), which in turn contains another TLV, with TLV-Type number four (4).

2.7.4. VSA Identifiers

There has historically been no method for numerically addressing VSAs. The "dotted number" method defined here can also be leveraged to create such an addressing scheme. However, as the VSAs are completely under the control of each individual vendor, this section provides a suggested practice, but does not define a standard of any kind.

The Vendor-Specific Attribute has been assigned the Attribute number 26. It in turn carries a 24-bit Vendor-Id, and possibly additional VSAs. Where the VSAs follow the [RFC2865] Section 5.26 recommended format, a VSA can be identified as "26.Vendor-Id.Vendor-Type".

For example, Livingston has Vendor-Id 307, and has defined an attribute "IP-Pool" as number 6. This VSA can be uniquely identified as 26.307.6, but it cannot be uniquely identified by name, as other vendors may have used the same name.

Note that there are few restrictions on the size of the numerical values in this notation. The Vendor-Id is a 24-bit number, and the VSA may have been assigned from a 16-bit vendor-specific Attribute Type space. Implementations SHOULD be capable of handling 32-bit numbers at each level of the "dotted number" notation.

For example, the company USR has been allocated Vendor-Id 429, and has defined a "Version-Id" attribute as number 32768. This VSA can be uniquely identified as 26.429.32768, and again cannot be uniquely identified by name.

Where a VSA is a TLV, the "dotted number" notation can be used as above: 26.Vendor-Id.Vendor-Type.TLV1.TLV2.TLV3 where "TLVn" are the numerical values assigned by the vendor to the different nested TLVs.

2.8. Invalid Attributes

The term "invalid attribute" is new to this specification. It is defined to mean that the Length field of an Attribute permits the packet to be accepted as not being "malformed". However, the Value field of the attribute does not follow the format required by the data type defined for that Attribute, and therefore the attribute is "malformed". In order to distinguish the two cases, we refer to "malformed" packets, and "invalid attributes".

For example, an implementation receives a packet which is well-formed. That packet contains an Attribute allegedly of data type "address", but which has Length not equal to four. In that situation, the packet is well formed, but the attribute is not. Therefore, it is an "invalid attribute".

A similar analysis can be performed when an attribute carries TLVs. The encapsulating attribute may be well formed, but the TLV may be an "invalid attribute". The existence of an "invalid attribute" in a packet or attribute MUST NOT result in the implementation discarding the entire packet, or treating the packet as a negative acknowledgment. Instead, only the "invalid attribute" is treated specially.

When an implementation receives an "invalid attribute", it SHOULD be silently discarded, except when the implementation is acting as a proxy (see Section 5.2 for discussion of proxy servers). If it is not discarded, it MUST NOT be handled in the same manner as a well-formed attribute. For example, receiving an Attribute of data type "address" containing less than four octets, or more than four octets of data means that the Attribute MUST NOT be treated as being of data type "address". The reason here is that if the attribute does not carry an IPv4 address, the receiver has no idea what format the data

is in, and it is therefore not an IPv4 address.

For Attributes of type "Long Extended Type", an Attribute is considered to be an "invalid attribute" when it does not match the criteria set out in Section 2.2, above.

For Attributes of type "TLV", an Attribute is considered to be an "invalid attribute" when the TLV-Length field allows the encapsulating Attribute to be parsed, but the TLV-Value field does not match the criteria for that TLV. Implementations SHOULD NOT treat the "invalid attribute" property as being transitive. That is, the Attribute encapsulating the "invalid attribute" SHOULD NOT be treated as an "invalid attribute". That encapsulating Attribute might contain multiple TLVs, only one of which is an "invalid attribute".

However, a TLV definition may require particular sub-TLVs to be present, and/or to have specific values. If a sub-TLV is missing, or contains incorrect value(s), or is an "invalid attribute", then the encapsulating TLV SHOULD be treated as an "invalid attribute". This requirement ensures that strongly connected TLVs are handled either as a coherent whole, or are ignored entirely.

It is RECOMMENDED that Attributes with unknown Type, Ext-Type, TLV-Type, or VSA-Type are treated as "invalid attributes". This recommendation is compatible with the suggestion in [RFC2865] Section 5, that implementations "MAY ignore Attributes with an unknown Type".

3. Attribute Definitions

We define four (4) attributes of "Extended Type", which are allocated from the "Reserved" Attribute Type codes of 241, 242, 243, and 244. We also define two (2) attributes of "Long Extended Type", which are allocated from the "Reserved" Attribute Type codes of 245 and 246.

Type	Name
----	----
241	Extended-Type-1
242	Extended-Type-2
243	Extended-Type-3
244	Extended-Type-4
245	Long-Extended-Type-1
246	Long-Extended-Type-2

The rest of this section gives a detailed definition for each Attribute based on the above summary.

3.1. Extended-Type-1

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 241.{1-255}.

A summary of the Extended-Type-1 Attribute format is shown below. The fields are transmitted from left to right.

										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Type										Length										Extended-Type										Value ...									

Type

241 for Extended-Type-1.

Length

$$\geq 4$$

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 241.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

3.2. Extended-Type-2

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 242.{1-255}.

A summary of the Extended-Type-2 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      | Extended-Type | Value ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

242 for Extended-Type-2.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 242.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field

3.3. Extended-Type-3

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 243.{1-255}.

A summary of the Extended-Type-3 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      | Extended-Type | Value ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

243 for Extended-Type-3.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 243.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

3.4. Extended-Type-4**Description**

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 244.{1-255}.

A summary of the Extended-Type-4 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      | Extended-Type | Value ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

244 for Extended-Type-4.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 244.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value Field.

3.5. Long-Extended-Type-1

Description

This attribute encapsulates attributes of the "Long Extended Type" format, in the RADIUS Attribute Type Space of 245.{1-255}.

A summary of the Long-Extended-Type-1 Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										Length										Extended-Type										M Reserved									
Value ...																																							

Type

245 for Long-Extended-Type-1

Length

>= 5

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this

field are specified in the 245.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Value

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

3.6. Long-Extended-Type-2

Description

This attribute encapsulates attributes of the "Long Extended Type" format, in the RADIUS Attribute Type Space of 246.{1-255}.

A summary of the Long-Extended-Type-2 Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3										
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1									
Type										Length										Extended-Type										M	Reserved									
Value ...																																								

Type

246 for Long-Extended-Type-2

Length

>= 5

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 246.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Value

The Value field is one or more octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

4. Vendor Specific Attributes

We define six new attributes which can carry Vendor Specific information. We define four (4) attributes of the "Extended Type" format, with Type codes (241.26, 242.26, 243.26, 244.26), using the "evs" data type. We also define two (2) attributes using "Long Extended Type" format, with Type codes (245.26, 246.26), which are of the "evs" data type.

Type.Extended-Type	Name
-----	----
241.26	Extended-Vendor-Specific-1
242.26	Extended-Vendor-Specific-2
243.26	Extended-Vendor-Specific-3
244.26	Extended-Vendor-Specific-4

245.26 Extended-Vendor-Specific-5
 246.26 Extended-Vendor-Specific-6

The rest of this section gives a detailed definition for each Attribute based on the above summary.

4.1. Extended-Vendor-Specific-1

Description

This attribute defines a RADIUS Type Code of 241.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-1 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |      Length      | Extended-Type | Vendor-Id ...
+-----+-----+-----+-----+-----+-----+-----+-----+
| ... Vendor-Id (cont) | Vendor-Type |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value ....
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type.Extended-Type

241.26 for Extended-Vendor-Specific-1

Length

>= 9

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust

implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the Value field is eight (8) less than the value of the Length field.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

4.2. Extended-Vendor-Specific-2

Description

This attribute defines a RADIUS Type Code of 242.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-2 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |      Length      | Extended-Type | Vendor-Id ...
+-----+-----+-----+-----+-----+-----+-----+-----+
| ... Vendor-Id (cont) | Vendor-Type |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value ....
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type.Extended-Type

242.26 for Extended-Vendor-Specific-2

Length

>= 9

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the

sole discretion of the Vendor.

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the Value field is eight (8) less than the value of the Length field.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

4.3. Extended-Vendor-Specific-3

Description

This attribute defines a RADIUS Type Code of 243.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-3 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      | Extended-Type | Vendor-Id ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ... Vendor-Id (cont) | Vendor-Type |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Value ....
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type.Extended-Type

243.26 for Extended-Vendor-Specific-3

Length

>= 9

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the Value field is eight (8) less than the value of the Length field.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

4.4. Extended-Vendor-Specific-4

Description

This attribute defines a RADIUS Type Code of 244.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-3 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Type   |   Length   | Extended-Type | Vendor-Id ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ... Vendor-Id (cont) | Vendor-Type |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Value ....
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type.Extended-Type

244.26 for Extended-Vendor-Specific-4

Length

>= 9

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the Value field is eight (8) less than the value of the Length field.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

4.5. Extended-Vendor-Specific-5

Description

This attribute defines a RADIUS Type Code of 245.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-5 Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
Type										Length										Extended-Type										M										Reserved									
										Vendor-Id																																							
Vendor-Type																				Value																													

+-----+

Type.Extended-Type

245.26 for Extended-Vendor-Specific-5

Length

>= 10 (first fragment)
>= 5 (subsequent fragments)

When a VSA is fragmented across multiple Attributes, only the first Attribute contains the Vendor-Id and Vendor-Type fields. Subsequent Attributes contain fragments of the Value field only.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

Implementations supporting this specification MUST use the

Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

4.6. Extended-Vendor-Specific-6

Description

This attribute defines a RADIUS Type Code of 246.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-6 Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
Type										Length										Extended-Type										M										Reserved									
										Vendor-Id																																							
Vendor-Type										Value																																							

Type.Extended-Type

246.26 for Extended-Vendor-Specific-6

Length

>= 10 (first fragment)
>= 5 (subsequent fragments)

When a VSA is fragmented across multiple Attributes, only the first Attribute contains the Vendor-Id and Vendor-Type fields. Subsequent Attributes contain fragments of the Value field only.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Vendor-Id

The 4 octets are the Network Management Private Enterprise Code [PEN] of the Vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the Vendor.

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

5. Compatibility with traditional RADIUS

There are a number of potential compatibility issues with traditional RADIUS, as defined in [RFC6158] and earlier. This section describes them.

5.1. Attribute Allocation

Some vendors have used Attribute Type codes from the "Reserved" space, as part of vendor-defined dictionaries. This practice is considered anti-social behavior, as noted in [RFC6158]. These vendor definitions conflict with the attributes in the RADIUS Attribute Type space. The conflicting definitions may make it difficult for implementations to support both those Vendor Attributes, and the new Extended Attribute formats.

We RECOMMEND that RADIUS client and server implementations delete all references to these improperly defined attributes. Failing that, we RECOMMEND that RADIUS server implementations have a per-client configurable flag which indicates which type of attributes are being sent from the client. If the flag is set to "Non-Standard Attributes", the conflicting attributes can be interpreted as being improperly defined Vendor Specific Attributes. If the flag is set the "IETF Attributes", the attributes MUST be interpreted as being of the Extended Attributes format. The default SHOULD be to interpret the

attributes as being of the Extended Attributes format.

Other methods of determining how to decode the attributes into a "correct" form are NOT RECOMMENDED. Those methods are likely to be fragile and prone to error.

We RECOMMEND that RADIUS server implementations re-use the above flag to determine which type of attributes to send in a reply message. If the request is expected to contain the improperly defined attributes, the reply SHOULD NOT contain Extended Attributes. If the request is expected to contain Extended Attributes, the reply MUST NOT contain the improper Attributes.

RADIUS clients will have fewer issues than servers. Clients MUST NOT send improperly defined Attributes in a request. For replies, clients MUST interpret attributes as being of the Extended Attributes format, instead of the improper definitions. These requirements impose no change in the RADIUS specifications, as such usage by vendors has always been in conflict with the standard requirements and the standards process.

Existing clients that send these improperly defined attributes usually have a configuration setting which can disable this behavior. We RECOMMEND that vendors ship products with the default set to "disabled". We RECOMMEND that administrators set this flag to "disabled" on all equipment that they manage.

5.2. Proxy Servers

RADIUS Proxy servers will need to forward Attributes having the new format, even if they do not implement support for the encoding and decoding of those attributes. We remind implementers of the following text in [RFC2865] Section 2.3:

The forwarding server MUST NOT change the order of any attributes of the same type, including Proxy-State.

This requirement solves some of the issues related to proxying of the new format, but not all. The reason is that proxy servers are permitted to examine the contents of the packets that they forward. Many proxy implementations not only examine the attributes, but they refuse to forward attributes which they do not understand (i.e. attributes for which they have no local dictionary definitions).

This practice is NOT RECOMMENDED. Proxy servers SHOULD forward attributes, even ones which they do not understand, or which 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 which understands them.

The only exception to this recommendation is when local site policy dictates that filtering of attributes has to occur. For example, a filter at a visited network may require removal of certain authorization rules which apply to the home network, but not to the visited network. This filtering can sometimes be done even when the contents of the attributes are unknown, such as when all Vendor-Specific Attributes are designated for removal.

As seen in [EDUROAM] many proxies do not follow these practices for unknown Attributes. Some proxies filter out unknown attributes or attributes which have unexpected lengths (24%, 17/70), some truncate the attributes to the "expected" length (11%, 8/70), some discard the request entirely (1%, 1/70), with the rest (63%, 44/70) following the recommended practice of passing the attributes verbatim. It will be difficult to widely use the Extended Attributes format until all non-conformant proxies are fixed. We therefore RECOMMEND that all proxies which do not support the Extended Attributes (241 through 246) define them as being of data type "string", and delete all other local definitions for those attributes.

This last change should enable wider usage of the Extended Attributes format.

6. Guidelines

This specification proposes a number of changes to RADIUS, and therefore requires a set of guidelines, as has been done in [RFC6158]. These guidelines include suggestions around design, interaction with IANA, usage, and implementation of attributes using the new formats.

6.1. Updates to RFC 6158

This specification updates [RFC6158] by adding the data types "evs", "tlv" and "integer64"; defining them to be "basic" data types; and permitting their use subject to the restrictions outlined below.

The recommendations for the use of the new data types and attribute formats are given below.

6.2. Guidelines for Simple Data Types

[RFC6158] Section A.2.1 says in part:

- * Unsigned integers of size other than 32 bits.

SHOULD be replaced by an unsigned integer of 32 bits. There is insufficient justification to define a new size of integer.

We update that specification to permit unsigned integers of 64 bits, for the reasons defined above in Section 2.5. The updated text is as follows:

- * Unsigned integers of size other than 32 or 64 bits.
SHOULD be replaced by an unsigned integer of 32 or 64 bits.
There is insufficient justification to define a new size of integer.

That section later continues with the following list item:

- * Nested attribute-value pairs (AVPs).
Attributes should be defined in a flat typespace.

We update that specification to permit nested TLVs, as defined in this document:

- * Nested attribute-value pairs (AVPs) using the extended attribute format MAY be used. All other nested AVP or TLV formats MUST NOT be used.

The [RFC6158] recommendations for "basic" data types apply to the three types listed above. All other recommendations given in [RFC6158] for "basic" data types remain unchanged.

6.3. Guidelines for Complex Data Types

[RFC6158] Section 2.1 says:

Complex data types MAY be used in situations where they reduce complexity in non-RADIUS systems or where using the basic data types would be awkward (such as where grouping would be required in order to link related attributes).

Since the extended attribute format allows for grouping of complex types via TLVs, the guidelines for complex data types need to be updated as follows:

[RFC6158], Section 3.2.4, describes situations in which complex data types might be appropriate. They SHOULD NOT be used even in those situations, without careful consideration of the described limitations. In all other cases not covered by the complex data type exceptions, complex data types MUST NOT be used. Instead,

complex data types MUST be decomposed into TLVs.

The checklist in Appendix A.2.2 is similarly updated to add a new requirement at the top of that section,

Does the attribute:

- * define a complex type which can be represented via TLVs?

If so, this data type MUST be represented via TLVs.

Note that this requirement does not over-ride Section A.1, which permits the transport of complex types in certain situations.

All other recommendations given in [RFC6158] for "complex" data types remain unchanged.

6.4. Design Guidelines For the New Types

This section gives design guidelines for specifications defining attributes using the new format. The items listed below are not exhaustive. As experience is gained with the new formats, later specifications may define additional guidelines.

- * The data type "evs" MUST NOT be used for standard RADIUS Attributes, or for TLVs, or for VSAs.
- * The data type "tlv" SHOULD NOT be used for standard RADIUS attributes.
- * [RFC2866] "tagged" attributes MUST NOT be defined in the Extended-Type space. The "tlv" data type should be used instead to group attributes.
- * The "integer64" data type MAY be used in any RADIUS attribute. The use of 64-bit integers was not recommended in [RFC6158], but their utility is now evident.
- * Any attribute which is allocated from the "long extended space" of data type "text", "string", or "tlv" can potentially carry more than 251 octets of data. Specifications defining such attributes SHOULD define a maximum length to guide implementations.

All other recommendations given in [RFC6158] for attribute design guidelines apply to attributes using the "short extended space" and "long extended space".

6.5. TLV Guidelines

The following items give design guidelines for specifications using TLVs.

- * when multiple attributes are intended to be grouped or managed together, the use of TLVs to group related attributes is RECOMMENDED.
- * more than 4 layers (depth) of TLV nesting is NOT RECOMMENDED.
- * Interpretation of an attribute depends only on its type definition (e.g. Type.Extended-Type.TLV-Type), and not on its encoding or location in the RADIUS packet.
- * Where a group of TLVs is strictly defined, and not expected to change, and totals less than 247 octets of data, they SHOULD request allocation from the "short extended space".
- * Where a group of TLVs is loosely defined, or is expected to change, they SHOULD request allocation from the "long extended space".

All other recommendations given in [RFC6158] for attribute design guidelines apply to attributes using the TLV format.

6.6. Allocation Request Guidelines

The following items give guidelines for allocation requests made in a RADIUS specification.

- * Discretion is recommended when requesting allocation of attributes. The new space is much larger than the old one, but it is not infinite.
- * Specifications which allocate many attributes MUST NOT request that allocation be made from the standard space. That space is under allocation pressure, and the extended space is more suitable for large allocations. As a guideline, we suggest that one specification allocating twenty percent (20%) or more of the standard space would meet the above criteria.
- * Specifications which allocate many related attributes SHOULD define one or more TLVs to contain related attributes.
- * Specifications SHOULD request allocation from a specific space. The IANA considerations given in Section 9, below, give instruction to IANA, but authors should assist IANA where possible.

- * Specifications of an attribute which encodes 252 octets or less of data MAY request allocation from the "short extended space".
- * Specifications of an attribute which always encode less than 253 octets of data MUST NOT request allocation from the long extended space. The standard space, or the short extended space MUST be used instead.
- * Specifications of an attribute which encodes 253 octets or more of data MUST request allocation from the "long extended space".
- * When the extended space is nearing exhaustion, a new specification will have to be written which requests allocation of one or more RADIUS Attributes from the "Reserved" portion of the standard space, values 247-255, using an appropriate format ("Short Extended Type", or "Long Extended Type")

An allocation request made in a specification SHOULD use one of the following formats when allocating an attribute type code:

- * TBDn - request allocation of an attribute from the "standard space". The value "n" should be "1" or more, to track individual attributes which are to be allocated.
- * SHORT-TBDn - request allocation of an attribute from the "short extended space". The value "n" should be "1" or more, to track individual attributes which are to be allocated.
- * LONG-TBDn - request allocation of an attribute from the "long extended space". The value "n" should be "1" or more, to track individual attributes which are to be allocated.

These guidelines should help specification authors and IANA communicate effectively and clearly.

6.7. Allocation Requests Guidelines for TLVs

Specifications may allocate a new attribute of type TLV, and at the same time, allocate sub-attributes within that TLV. These specifications SHOULD request allocation of specific values for the sub-TLV. The "dotted number" notation MUST be used.

For example, a specification may request allocation of a TLV as SHORT-TBD1. Within that attribute, it could request allocation of three sub-TLVs, as SHORT-TBD1.1, SHORT-TBD1.2, and SHORT-TBD1.3.

Specifications may request allocation of additional sub-TLVs within an existing attribute of type TLV. Those specifications SHOULD use

the "TBDn" format for every entry in the "dotted number" notation.

For example, a specification may request allocation within an existing TLV, with "dotted number" notation MM.NN. Within that attribute, the specification could request allocation of three sub-TLVs, as MM.NN.TBD1, MM.NN.TBD2, and MM.NN.TBD3.

6.8. Implementation Guidelines

- * RADIUS client implementations SHOULD support this specification, in order to permit the easy deployment of specifications using the changes defined herein.
- * RADIUS server implementations SHOULD support this specification, in order to permit the easy deployment of specifications using the changes defined herein.
- * RADIUS proxy servers MUST follow the specifications in section 5.2

6.9. Vendor Guidelines

- * Vendors SHOULD use the existing Vendor-Specific Attribute Type space in preference to the new Extended-Vendor-Specific attributes, as this specification may take time to become widely deployed.
- * Vendors SHOULD implement this specification. The changes to RADIUS are relatively small, and are likely to quickly be used in new specifications.

7. Rationale for This Design

The path to extending the RADIUS protocol has been long and arduous. A number of proposals have been made and discarded by the RADEXT working group. These proposals have been judged to be either too bulky, too complex, too simple, or to be unworkable in practice. We do not otherwise explain here why earlier proposals did not obtain working group consensus.

The changes outlined here have the benefit of being simple, as the "Extended Type" format requires only a one octet change to the Attribute format. The downside is that the "Long Extended Type" format is awkward, and the 7 Reserved bits will likely never be used for anything.

7.1. Attribute Audit

An audit of almost five thousand publicly available attributes [ATTR] (2010), shows the statistics summarized below. The attributes include over 100 Vendor dictionaries, along with the IANA assigned attributes:

Count	Data Type
-----	-----
2257	integer
1762	text
273	IPv4 Address
225	string
96	other data types
35	IPv6 Address
18	date
10	integer64
4	Interface Id
3	IPv6 Prefix
4683	Total

The entries in the "Data Type" column are data types recommended by [RFC6158], along with "integer64". The "other data types" row encompasses all other data types, including complex data types and data types transporting opaque data.

We see that over half of the attributes encode less than 16 octets of data. It is therefore important to have an extension mechanism which adds as little as possible to the size of these attributes. Another result is that the overwhelming majority of attributes use simple data types.

Of the attributes defined above, 177 were declared as being inside of a TLV. This is approximately 4% of the total. We did not investigate whether additional attributes were defined in a flat name space, but could have been defined as being inside of a TLV. We expect that the number could be as high as 10% of attributes.

Manual inspection of the dictionaries shows that approximately 20 (or 0.5%) attributes have the ability to transport more than 253 octets of data. These attributes are divided between VSAs, and a small number of standard Attributes such as EAP-Message.

The results of this audit and analysis is reflected in the design of the extended attributes. The extended format has minimal overhead, it permits TLVs, and it has support for "long" attributes.

8. Diameter Considerations

The attribute formats defined in this specification need to be transported in Diameter. While Diameter supports attributes longer than 253 octets and grouped attributes, we do not use that functionality here. Instead, we define the simplest possible encapsulation method.

The new formats MUST be treated the same as traditional RADIUS attributes when converting from RADIUS to Diameter, or vice versa. That is, the new attribute space is not converted to any "extended" Diameter attribute space. Fragmented attributes are not converted to a single long Diameter attribute. The new EVS data types are not converted to Diameter attributes with the "V" bit set.

In short, this document mandates no changes for existing RADIUS to Diameter, or Diameter to RADIUS gateways.

9. Examples

A few examples are presented here in order to illustrate the encoding of the new attribute formats. These examples are not intended to be exhaustive, as many others are possible. For simplicity, we do not show complete packets, only attributes.

The examples are given using a domain-specific language implemented by the program given in Appendix A. The language is line oriented, and composed of a sequence of lines matching the grammar ([RFC5234]) given below:

```
Identifier = 1*DIGIT *( "." 1*DIGIT )

HEXCHAR = HEXDIG HEXDIG

STRING = DQUOTE 1*CHAR DQUOTE

TLV = "{" SP 1*DIGIT SP DATA SP "}"

DATA = (HEXCHAR *(SP HEXCHAR)) / (TLV *(SP TLV)) / STRING

LINE = Identifier SP DATA
```

The program has additional restrictions on its input that are not reflected in the above grammar. For example, the portions of the Identifier which refer to Type and Extended-Type are limited to values between 1 and 255. We trust that the source code in Appendix A is clear, and that these restrictions do not negatively affect the comprehensibility of the examples.

The program reads the input text, and interprets it as a set of instructions to create RADIUS Attributes. It then prints the hex encoding of those attributes. It implements the minimum set of functionality which achieves that goal. This minimalism means that it does not use attribute dictionaries; it does not implement support for RADIUS data types; it can be used to encode attributes with invalid data fields; and there is no requirement for consistency from one example to the next. For example, it can be used to encode a User-Name attribute which contains non-UTF8 data, or a Framed-IP-Address which contains 253 octets of ASCII data. As a result, it MUST NOT be used to create RADIUS Attributes for transport in a RADIUS message.

However, the program correctly encodes the RADIUS attribute fields of "Type", "Length", "Extended-Type", "More", "Reserved", "Vendor-Id", "Vendor-Type", and "Vendor-Length". It encodes RADIUS attribute data types "evs" and TLV. It can therefore be used to encode example attributes from inputs which are humanly readable.

We do not give examples of "malformed" or "invalid attributes". We also note that the examples show format, rather than consistent meaning. A particular Attribute Type code may be used to demonstrate two different formats. In real specifications, attributes have a static definitions based on their type code.

The examples given below are strictly for demonstration purposes only, and do not provide a standard of any kind.

9.1. Extended Type

The following are a series of examples of the "Extended Type" format.

Attribute encapsulating textual data.

```
241.1 "bob"
-> f1 06 01 62 6f 62
```

Attribute encapsulating a TLV with TLV-Type of one (1).

```
241.2 { 1 23 45 }
-> f1 07 02 01 04 23 45
```

Attribute encapsulating two TLVs, one after the other.

```
241.2 { 1 23 45 } { 2 67 89 }
-> f1 0b 02 01 04 23 45 02 04 67 89
```

Attribute encapsulating two TLVs, where the second TLV is itself

encapsulating a TLV.

```
241.2 { 1 23 45 } { 3 { 1 ab cd } }  
-> f1 0d 02 01 04 23 45 03 06 01 04 ab cd
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating two TLVs.

```
241.2 { 1 23 45 } { 3 { 1 ab cd } { 2 "foo" } }  
-> f1 12 02 01 04 23 45 03 0b 01 04 ab cd 02 05 66 6f 6f
```

Attribute encapsulating a TLV, which in turn encapsulates a TLV, to a depth of 5 nestings.

```
241.1 { 1 { 2 { 3 { 4 { 5 cd ef } } } } }  
-> f1 0f 01 01 0c 02 0a 03 08 04 06 05 04 cd ef
```

Attribute encapsulating an extended Vendor Specific attribute, with Vendor-Id of 1, and Vendor-Type of 4, which in turn encapsulates textual data.

```
241.26.1.4 "test"  
-> f1 0c 1a 00 00 00 01 04 74 65 73 74
```

Attribute encapsulating an extended Vendor Specific attribute, with Vendor-Id of 1, and Vendor-Type of 5, which in turn encapsulates a TLV with TLV-Type of 3, which encapsulates textual data.

```
241.26.1.5 { 3 "test" }  
-> f1 0e 1a 00 00 00 01 05 03 06 74 65 73 74
```

9.2. Long Extended Type

The following are a series of examples of the "Long Extended Type" format.

Attribute encapsulating textual data.

```
245.1 "bob"  
-> f5 07 01 00 62 6f 62
```

Attribute encapsulating a TLV with TLV-Type of one (1).

```
245.2 { 1 23 45 }  
-> f5 08 02 00 01 04 23 45
```

Attribute encapsulating two TLVs, one after the other.


```

          "cccccccccccc"
-> f5 ff 04 80 aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa ab bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc

```

Attribute encapsulating an extended Vendor Specific attribute, with Vendor-Id of 1, and Vendor-Type of 6, which in turn encapsulates more than 251 octets of data.

As the VSA encapsulates more than 251 octets of data, it is split into two RADIUS attributes. The first attribute has the More field set, and carries the Vendor-Id and Vendor-Type. The second attribute has the More field clear, and carries the rest of the data portion of the VSA. Note that the second attribute does not include the Vendor-Id and Vendor-Type fields.

The "Data" portions are indented for readability.

```

245.26.1.6 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbccccc
            ccccccccccccccc"
-> f5 ff 1a 80 00 00 00 01 06 aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
    aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa ab bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
    bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb

```

```

bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb f5 18 1a 00 bb
bb bb bb bb cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc

```

10. IANA Considerations

This document updates [RFC3575] in that it adds new IANA considerations for RADIUS Attributes. These considerations modify and extend the IANA considerations for RADIUS, rather than replacing them.

The IANA considerations of this document are limited to the "RADIUS Attribute Types" registry. Some Attribute Type values which were previously marked "Reserved" are now allocated, and the registry is extended from a simple 8-bit array to a tree-like structure, up to a maximum depth of 125 nodes. Detailed instructions are given below.

10.1. Attribute Allocations

IANA is requested to move the following Attribute Type values from "Reserved", to "Allocated", with the corresponding names:

- * 241 Extended-Type-1
- * 242 Extended-Type-2
- * 243 Extended-Type-3
- * 244 Extended-Type-4
- * 245 Long-Extended-Type-1
- * 246 Long-Extended-Type-2

These values serve as an encapsulation layer for the new RADIUS Attribute Type tree.

10.2. RADIUS Attribute Type Tree

Each of the Attribute Type values allocated above extends the "RADIUS Attribute Types" to an N-ary tree, via a "dotted number" notation. Allocation of an Attribute Type value "TYPE" using the new Extended type format results in allocation of 255 new Attribute Type values, of format "TYPE.1" through "TYPE.255". Value twenty-six (26) is assigned as "Extended-Vendor-Specific-*". Values "TYPE.241" through "TYPE.255" are marked "Reserved". All other values are "Unassigned".

The initial set of Attribute Type values and names assigned by this document is given below.

* 241	Extended-Attribute-1
* 241.{1-25}	Unassigned
* 241.26	Extended-Vendor-Specific-1
* 241.{27-240}	Unassigned
* 241.{241-255}	Reserved
* 242	Extended-Attribute-2
* 242.{1-25}	Unassigned
* 242.26	Extended-Vendor-Specific-2
* 242.{27-240}	Unassigned
* 243	Extended-Attribute-3
* 242.{241-255}	Reserved
* 243.{1-25}	Unassigned
* 243.26	Extended-Vendor-Specific-3
* 243.{27-240}	Unassigned
* 243.{241-255}	Reserved
* 244	Extended-Attribute-4
* 244.{1-25}	Unassigned
* 244.26	Extended-Vendor-Specific-4
* 244.{27-240}	Unassigned
* 244.{241-255}	Reserved
* 245	Extended-Attribute-5
* 245.{1-25}	Unassigned
* 245.26	Extended-Vendor-Specific-5
* 245.{27-240}	Unassigned
* 245.{241-255}	Reserved
* 246	Extended-Attribute-6
* 246.{1-25}	Unassigned
* 245.26	Extended-Vendor-Specific-6
* 246.{27-240}	Unassigned
* 246.{241-255}	Reserved

As per [RFC5226], the values marked "Unassigned" above are available via for assignment by IANA in future RADIUS specifications. The values marked "Reserved" are reserved for future use.

The Extended-Vendor-Specific spaces (TYPE.26) are for Private Use, and allocations are not managed by IANA.

Allocation of Reserved entries in the extended space requires Standards Action.

All other allocations in the extended space require IETF Review.

10.3. Allocation Instructions

This section defines what actions IANA needs to take when allocating new attributes. Different actions are required when allocating attributes from the standard space, attributes of Extended Type

format, attributes of the "Long Extended Type" format, preferential allocations, attributes of data type TLV, attributes within a TLV, and attributes of other data types.

10.3.1. Requested Allocation from the Standard Space

Specifications can request allocation of an Attribute from within the standard space (e.g. Attribute Type Codes 1 through 255), subject to the considerations of [RFC3575] and this document.

10.3.2. Requested Allocation from the short extended space

Specifications can request allocation of an Attribute which requires the format Extended Type, by specifying the short extended space. In that case, IANA should assign the lowest Unassigned number from the Attribute Type space with the relevant format.

10.3.3. Requested Allocation from the long extended space

Specifications can request allocation of an Attribute which requires the format "Long Extended Type", by specifying the extended space (long). In that case, IANA should assign the lowest Unassigned number from the Attribute Type space with the relevant format.

10.3.4. Allocation Preferences

Specifications which make no request for allocation from a specific Type Space should have Attributes allocated using the following criteria:

- * when the standard space has no more Unassigned attributes, all allocations should be performed from the extended space.
- * specifications which allocate a small number of attributes (i.e. less than ten) should have all allocations made from the standard space.
- * specifications which would allocate a more than twenty percent of the remaining standard space attributes should have all allocations made from the extended space.
- * specifications which request allocation of an attribute of data type TLV should have that attribute allocated from the extended space.
- * specifications which request allocation of an attribute which can transport 253 or more octets of data should have

that attribute allocated from within the long extended space,
We note that Section 6.5, above requires specifications to
request this allocation.

There is otherwise no requirement that all attributes within a
specification be allocated from one type space or another.
Specifications can simultaneously allocate attributes from both the
standard space and the extended space.

10.3.5. Extending the Type Space via TLV Data Type

When specifications request allocation of an attribute of data type
"tlv", that allocation extends the Attribute Type Tree by one more
level. Allocation of an Attribute Type value "TYPE.TLV", with Data
Type TLV, results in allocation of 255 new Attribute Type values, of
format "TYPE.TLV.1" through "TYPE.TLV.255". Values 254-255 are
marked "Reserved". All other values are "Unassigned". Value 26 has
no special meaning.

For example, if a new attribute "Example-TLV" of data type "tlv" is
assigned the identifier "245.1", then the extended tree will be
allocated as below:

```
* 245.1           Example-TLV
* 245.1.{1-253}   Unassigned
* 245.1.{254-255} Reserved
```

Note that this example does not define an "Example-TLV" attribute.

The Attribute Type Tree can be extended multiple levels in one
specification when the specification requests allocation of nested
TLVs, as discussed below.

10.3.6. Allocation within a TLV

Specifications can request allocation of Attribute Type values within
an Attribute of Data Type TLV. The encapsulating TLV can be
allocated in the same specification, or it can have been previously
allocated.

Specifications need to request allocation within a specific Attribute
Type value (e.g. "TYPE.TLV.*"). Allocations are performed from the
smallest Unassigned value, proceeding to the largest Unassigned
value.

Where the Attribute being allocated is of Data Type TLV, the
Attribute Type tree is extended by one level, as given in the

previous section. Allocations can then be made within that level.

10.3.7. Allocation of Other Data Types

Attribute Type value allocations are otherwise allocated from the smallest Unassigned value, proceeding to the largest Unassigned value. e.g. Starting from 241.1, proceeding through 241.255, then to 242.1, through 242.255, etc.

11. Security Considerations

This document defines new formats for data carried inside of RADIUS, but otherwise makes no changes to the security of the RADIUS protocol.

Attacks on cryptographic hashes are well known, and are getting better with time, as discussed in [RFC4270]. The security of the RADIUS protocol is dependent on MD5 [RFC1311], which has security issues as discussed in [RFC6151]. It is not known if the issues described in [RFC6151] apply to RADIUS. For other issues, we incorporate by reference the security considerations of [RFC6158] Section 5.

As with any protocol change, code changes are required in order to implement the new features. These code changes have the potential to introduce new vulnerabilities in the software. Since the RADIUS server performs network authentication, it is an inviting target for attackers. We RECOMMEND that access to RADIUS servers be kept to a minimum.

12. References

12.1. Normative references

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.
- [RFC5176] Chiba, M, et. al., "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176,

January 2008.

[RFC5226] Narten, T. and Alvestrand, H, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, May 2008.

[RFC6158] DeKok, A., and Weber, G., "RADIUS Design Guidelines", RFC 6158, March 2011.

[PEN] <http://www.iana.org/assignments/enterprise-numbers>

12.2. Informative references

[RFC1321] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April, 1992

[RFC2868] Zorn, G., et al, " RADIUS Attributes for Tunnel Protocol Support", RFC 2868, June 2000.

[RFC4270] Hoffman, P, and Schneier, B, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[RFC5234] Crocker, D. (Ed.), and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, October 2005.

[RFC6151] Turner, S. and Chen, L., "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

[EDUROAM] Internal Eduroam testing page, data retrieved 04 August 2010.

[ATTR] <http://github.com/alandekok/freeradius-server/tree/master/share/>, data retrieved September 2010.

Acknowledgments

This document is the result of long discussions in the IETF RADEXT working group. The authors would like to thank all of the participants who contributed various ideas over the years. Their feedback has been invaluable, and has helped to make this specification better.

Appendix A - Extended Attribute Generator Program

This section contains "C" program source which can be used for testing. It reads a line-oriented text file, parses it to create RADIUS formatted attributes, and prints the hex version of those attributes to standard output.

The input accepts a grammar similar to that given in Section 9, with some modifications for usability. For example, blank lines are allowed, lines beginning with a '#' character are interpreted as comments, numbers (RADIUS Types, etc.) are checked for minimum / maximum values, and RADIUS Attribute lengths are enforced.

The program is included here for demonstration purposes only, and does not define a standard of any kind.

```
-----
/*
 * Copyright (c) 2010 IETF Trust and the persons identified as
 * authors of the code. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * Neither the name of Internet Society, IETF or IETF Trust, nor the
 * names of specific contributors, may be used to endorse or promote
 * products derived from this software without specific prior written
 * permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
```

```
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* Author: Alan DeKok <aland@networkradius.com>
*/
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>

static int encode_tlv(char *buffer, uint8_t *output, size_t outlen);

static const char *hextab = "0123456789abcdef";

static int encode_data_string(char *buffer,
                              uint8_t *output, size_t outlen)
{
    int length = 0;
    char *p;

    p = buffer + 1;

    while (*p && (outlen > 0)) {
        if (*p == '"') {
            return length;
        }

        if (*p != '\\') {
            *(output++) = *(p++);
            outlen--;
            length++;
            continue;
        }

        switch (p[1]) {
        default:
            *(output++) = p[1];
            break;

        case 'n':
            *(output++) = '\n';
            break;

        case 'r':
            *(output++) = '\r';
```

```
        break;

    case 't':
        *(output++) = '\t';
        break;
    }

    outlen--;
    length++;
}

fprintf(stderr, "String is not terminated\n");
return 0;
}

static int encode_data_tlv(char *buffer, char **endptr,
                           uint8_t *output, size_t outlen)
{
    int depth = 0;
    int length;
    char *p;

    for (p = buffer; *p != '\0'; p++) {
        if (*p == '{') depth++;
        if (*p == '}') {
            depth--;
            if (depth == 0) break;
        }
    }

    if (*p != '}') {
        fprintf(stderr, "No trailing '}' in string starting "
                    "with \"%s\"\n",
                    buffer);
        return 0;
    }

    *endptr = p + 1;
    *p = '\0';

    p = buffer + 1;
    while (isspace((int) *p)) p++;

    length = encode_tlv(p, output, outlen);
    if (length == 0) return 0;

    return length;
}
```

```
static int encode_data(char *p, uint8_t *output, size_t outlen)
{
    int length;

    if (!isspace((int) *p)) {
        fprintf(stderr, "Invalid character following attribute "
            "definition\n");
        return 0;
    }

    while (isspace((int) *p)) p++;

    if (*p == '{') {
        int sublen;
        char *q;

        length = 0;

        do {
            while (isspace((int) *p)) p++;
            if (!*p) {
                if (length == 0) {
                    fprintf(stderr, "No data\n");
                    return 0;
                }

                break;
            }

            sublen = encode_data_tlv(p, &q, output, outlen);
            if (sublen == 0) return 0;

            length += sublen;
            output += sublen;
            outlen -= sublen;
            p = q;
        } while (*q);

        return length;
    }

    if (*p == '"') {
        length = encode_data_string(p, output, outlen);
        return length;
    }

    length = 0;
    while (*p) {
```

```
    char *c1, *c2;

    while (isspace((int) *p)) p++;

    if (!*p) break;

    if(!(c1 = memchr(hextab, tolower((int) p[0]), 16)) ||
        !(c2 = memchr(hextab, tolower((int) p[1]), 16))) {
        fprintf(stderr, "Invalid data starting at "
            "\"%s\"\n", p);
        return 0;
    }

    *output = ((c1 - hextab) << 4) + (c2 - hextab);
    output++;
    length++;
    p += 2;

    outlen--;
    if (outlen == 0) {
        fprintf(stderr, "Too much data\n");
        return 0;
    }
}

if (length == 0) {
    fprintf(stderr, "Empty string\n");
    return 0;
}

return length;
}

static int decode_attr(char *buffer, char **endptr)
{
    long attr;

    attr = strtol(buffer, endptr, 10);
    if (*endptr == buffer) {
        fprintf(stderr, "No valid number found in string "
            "starting with \"%s\"\n", buffer);
        return 0;
    }

    if (**endptr) {
        fprintf(stderr, "Nothing follows attribute number\n");
        return 0;
    }
}
```

```
    if ((attr <= 0) || (attr > 256)) {
        fprintf(stderr, "Attribute number is out of valid "
            "range\n");
        return 0;
    }

    return (int) attr;
}

static int decode_vendor(char *buffer, char **endptr)
{
    long vendor;

    if (*buffer != '.') {
        fprintf(stderr, "Invalid separator before vendor id\n");
        return 0;
    }

    vendor = strtol(buffer + 1, endptr, 10);
    if (*endptr == (buffer + 1)) {
        fprintf(stderr, "No valid vendor number found\n");
        return 0;
    }

    if (!**endptr) {
        fprintf(stderr, "Nothing follows vendor number\n");
        return 0;
    }

    if ((vendor <= 0) || (vendor > (1 << 24))) {
        fprintf(stderr, "Vendor number is out of valid range\n");
        return 0;
    }

    if (**endptr != '.') {
        fprintf(stderr, "Invalid data following vendor number\n");
        return 0;
    }
    (*endptr)++;

    return (int) vendor;
}

static int encode_tlv(char *buffer, uint8_t *output, size_t outlen)
{
    int attr;
    int length;
    char *p;
```

```
    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    output[0] = attr;
    output[1] = 2;

    if (*p == '.') {
        p++;
        length = encode_tlv(p, output + 2, outlen - 2);
    } else {
        length = encode_data(p, output + 2, outlen - 2);
    }

    if (length == 0) return 0;
    if (length > (255 - 2)) {
        fprintf(stderr, "TLV data is too long\n");
        return 0;
    }

    output[1] += length;

    return length + 2;
}

static int encode_vsa(char *buffer, uint8_t *output, size_t outlen)
{
    int vendor;
    int attr;
    int length;
    char *p;

    vendor = decode_vendor(buffer, &p);
    if (vendor == 0) return 0;

    output[0] = 0;
    output[1] = (vendor >> 16) & 0xff;
    output[2] = (vendor >> 8) & 0xff;
    output[3] = vendor & 0xff;

    length = encode_tlv(p, output + 4, outlen - 4);
    if (length == 0) return 0;
    if (length > (255 - 6)) {
        fprintf(stderr, "VSA data is too long\n");
        return 0;
    }
}
```

```
        return length + 4;
    }

static int encode_evs(char *buffer, uint8_t *output, size_t outlen)
{
    int vendor;
    int attr;
    int length;
    char *p;

    vendor = decode_vendor(buffer, &p);
    if (vendor == 0) return 0;

    attr = decode_attr(p, &p);
    if (attr == 0) return 0;

    output[0] = 0;
    output[1] = (vendor >> 16) & 0xff;
    output[2] = (vendor >> 8) & 0xff;
    output[3] = vendor & 0xff;
    output[4] = attr;

    length = encode_data(p, output + 5, outlen - 5);
    if (length == 0) return 0;

    return length + 5;
}

static int encode_extended(char *buffer,
                           uint8_t *output, size_t outlen)
{
    int attr;
    int length;
    char *p;

    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    output[0] = attr;

    if (attr == 26) {
        length = encode_evs(p, output + 1, outlen - 1);
    } else {
        length = encode_data(p, output + 1, outlen - 1);
    }
    if (length == 0) return 0;
    if (length > (255 - 3)) {
        fprintf(stderr, "Extended Attr data is too long\n");
    }
}
```



```
        return 0;
    }

    return length + 1;
}

static int encode_extended_flags(char *buffer,
                                uint8_t *output, size_t outlen)
{
    int attr;
    int length, total;
    char *p;

    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    /* output[0] is the extended attribute */
    output[1] = 4;
    output[2] = attr;
    output[3] = 0;

    if (attr == 26) {
        length = encode_evs(p, output + 4, outlen - 4);
        if (length == 0) return 0;

        output[1] += 5;
        length -= 5;
    } else {
        length = encode_data(p, output + 4, outlen - 4);
    }
    if (length == 0) return 0;

    total = 0;
    while (1) {
        int sublen = 255 - output[1];

        if (length <= sublen) {
            output[1] += length;
            total += output[1];
            break;
        }

        length -= sublen;

        memmove(output + 255 + 4, output + 255, length);
        memcpy(output + 255, output, 4);

        output[1] = 255;
    }
}
```

```
        output[3] |= 0x80;

        output += 255;
        output[1] = 4;
        total += 255;
    }

    return total;
}

static int encode_rfc(char *buffer, uint8_t *output, size_t outlen)
{
    int attr;
    int length, sublen;
    char *p;

    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    length = 2;
    output[0] = attr;
    output[1] = 2;

    if (attr == 26) {
        sublen = encode_vsa(p, output + 2, outlen - 2);
    } else if ((*p == ' ') || ((attr < 241) || (attr > 246))) {
        sublen = encode_data(p, output + 2, outlen - 2);
    } else {
        if (*p != '.') {
            fprintf(stderr, "Invalid data following "
                          "attribute number\n");
            return 0;
        }

        if (attr < 245) {
            sublen = encode_extended(p + 1,
                                     output + 2, outlen - 2);
        } else {
            /*
             *   Not like the others!
             */
            return encode_extended_flags(p + 1, output, outlen);
        }
    }
    if (sublen == 0) return 0;
}
```

```
        if (sublen > (255 - 2)) {
            fprintf(stderr, "RFC Data is too long\n");
            return 0;
        }

        output[1] += sublen;
        return length + sublen;
    }

int main(int argc, char *argv[])
{
    int lineno;
    size_t i, outlen;
    FILE *fp;
    char input[8192], buffer[8192];
    uint8_t output[4096];

    if ((argc < 2) || (strcmp(argv[1], "-") == 0)) {
        fp = stdin;
    } else {
        fp = fopen(argv[1], "r");
        if (!fp) {
            fprintf(stderr, "Error opening %s: %s\n",
                    argv[1], strerror(errno));
            exit(1);
        }
    }

    lineno = 0;
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        char *p = strchr(buffer, '\n');

        lineno++;

        if (!p) {
            if (!feof(fp)) {
                fprintf(stderr, "Line %d too long in %s\n",
                        lineno, argv[1]);
                exit(1);
            }
        } else {
            *p = '\0';
        }

        p = strchr(buffer, '#');
        if (p) *p = '\0';

        p = buffer;
```

```
while (isspace((int) *p)) p++;
if (!*p) continue;

strcpy(input, p);
outlen = encode_rfc(input, output, sizeof(output));
if (outlen == 0) {
    fprintf(stderr, "Parse error in line %d of %s\n",
        lineno, input);
    exit(1);
}

printf("%s -> ", buffer);
for (i = 0; i < outlen; i++) {
    printf("%02x ", output[i]);
}
printf("\n");
}

if (fp != stdin) fclose(fp);

return 0;
}
```

Author's Address

Alan DeKok
Network RADIUS SARL
57bis blvd des Alpes
38240 Meylan
France

Email: aland@networkradius.com
URI: <http://networkradius.com>

Avi Lior
Email: avi.ietf@lior.org

RADIUS EXTensions Working Group
Internet-Draft
Intended status: Experimental
Expires: January 3, 2014

A. Perez-Mendez
R. Marin-Lopez
F. Pereniguez-Garcia
G. Lopez-Millan
University of Murcia
D. Lopez
Telefonica I+D
A. DeKok
Network RADIUS
July 2, 2013

Support of fragmentation of RADIUS packets
draft-perez-radext-radius-fragmentation-06

Abstract

The Remote Authentication Dial-In User Service (RADIUS) protocol is limited to a total packet size of 4096 octets. Provisions exist for fragmenting large amounts of authentication data across multiple packets, via Access-Challenge. No similar provisions exist for fragmenting large amounts of authorization data. This document specifies how existing RADIUS mechanisms can be leveraged to provide that functionality. These mechanisms are largely compatible with existing implementations, and are designed to be invisible to proxies, and "fail-safe" to legacy clients and servers.

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 <http://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 January 3, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Language	4
2. Scope of this document	4
3. Overview	5
4. Fragmentation of packets	7
4.1. Pre-authorization	8
4.2. Post-authorization	12
5. Chunk size	14
6. Allowed large packet size	15
7. Handling special attributes	16
7.1. Proxy-State attribute	16
7.2. State attribute	17
7.3. Service-Type attribute	17
7.4. Rebuilding the original large packet	17
8. New attribute definition	18
8.1. Frag-Status attribute	18
8.2. Proxy-State-Len attribute	19
8.3. Table of attributes	20
9. Operation with proxies	20
9.1. Legacy proxies	20
9.2. Updated proxies	21
10. Security Considerations	22
11. IANA Considerations	23
12. References	23
12.1. Normative References	23
12.2. Informative References	24
Authors' Addresses	24

1. Introduction

The RADIUS [RFC2865] protocol carries authentication, authorization, and accounting information between a Network Access Server (NAS) and an Authentication Server (AS). Information is exchanged between the NAS and the AS through RADIUS packets. Each RADIUS packet is composed of a header, and zero or more attributes, up to a maximum packet size of 4096 octets. The protocol is a request/response protocol, as described in the operational model ([RFC6158], Section 3.1).

The above packet size limitation mean that peers desiring to send large amounts of data must fragment it across multiple packets. For example, RADIUS-EAP [RFC3579] defines how an EAP exchange occurs across multiple Access-Request / Access-Challenge sequences. No such exchange is possible for accounting or authorization data. [RFC6158] Section 3.1 suggests that exchanging large amounts authorization data is unnecessary in RADIUS. Instead, the data should be referenced by name. This requirement allows large policies to be pre-provisioned, and then referenced in an Access-Accept. In some cases, however, the authorization data sent by the server is large and highly dynamic. In other cases, the NAS needs to send large amounts of authorization data to the server. Both of these cases are un-met by the requirements in [RFC6158]. As noted in that document, the practical limit on RADIUS packet sizes is governed by the Path MTU (PMTU), which may be significantly smaller than 4096 octets. The combination of the two limitations means that there is a pressing need for a method to send large amounts of authorization data between NAS and AS, with no accompanying solution.

[RFC6158] recommends three approaches for the transmission of large amount of data within RADIUS. However, they are not applicable to the problem statement of this document for the following reasons:

- o The first approach does not talk about large amounts of data sent from the NAS to a server. Leveraging EAP (request/challenge) to send the data is not feasible, as EAP already fills packet to PMTU, and not all authentications use EAP. Moreover, as noted for NAS-Filter-Rule ([RFC4849]), this approach does entirely solve the problem of sending large amounts of data from a server to a NAS.
- o The second approach is not usable either, as using names rather than values is difficult when the nature of the data to be sent is highly dynamic (e.g. SAML sentences or NAS-Filter-Rule attributes). URLs could be used as a pointer to the location of the actual data, but their use would require them to be (a) dynamically created and modified, (b) securely accessed and (c) accessible from remote systems. Satisfying these constraints

would require the modification of several networking systems (e.g. firewalls and web servers). Furthermore, the set up of an additional trust infrastructure (e.g. PKI) would be required to allow secure retrieving of the information from the web server.

- o PMTU discovery does not solve the problem, as it does not allow to send data larger than the minimum of (PMTU or 4096) octets.

This document provides a mechanism to allow RADIUS peers to exchange large amounts of authorization data exceeding the 4096 octet limit, by fragmenting it across several client/server exchanges. The proposed solution does not impose any additional requirements to the RADIUS system administrators (e.g. need to modify firewall rules, set up web servers, configure routers, or modify any application server). It maintains compatibility with intra-packet fragmentation mechanisms (like those defined in [RFC3579] or in [I-D.ietf-radext-radius-extensions]). It is also transparent to existing RADIUS proxies, which do not implement this specification. The only systems needing to implement the draft are the ones which either generate, or consume the fragmented data being transmitted. Intermediate proxies just pass the packets without changes. Nevertheless, if a proxy supports this specification, it MAY re-assemble the data in order to either examine and/or modify it.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Scope of this document

This specification describes how a RADIUS client and a RADIUS server can exchange large amounts of data exceeding the 4096 octet limit. Specifically, its scope is limited to the exchange of authorization data, as other exchanges do not require of such a mechanism. In particular, authentication exchanges have already been defined to overcome this limitation (e.g. RADIUS-EAP). Moreover, as they represent the most critical part of a RADIUS conversation, its preferable to not introduce any modification to their operation that may affect existing equipment.

There is no need to fragment accounting packets either. While the accounting process can send large amounts of data, that data is typically composed of many small updates. That is, there is no demonstrated need to send indivisible blocks of more than 4K of data. The need to send large amounts of data per user session often

originates from the need for flow-based accounting. In this use-case, the client may send accounting data for many thousands of flows, where all those flows are tied to one user session. The existing Acct-Multi-Session-Id attribute defined in [RFC2866] Section 5.11 has been proven to work here.

Similarly, there is no need to fragment CoA packets. Instead, the CoA client MUST send a CoA-Request packet containing session identification attributes, along with Service-Type = Additional-Authorization, and a State attribute. Implementations not supporting fragmentation will respond with a CoA-NAK, and an Error-Cause of Unsupported-Service.

Implementations supporting this specification may not be able to change authorization data for a particular session. In that case, they MUST respond with a CoA-NAK, as above. Otherwise, the implementation MUST start fragmentation via Access-Request, using the methods defined here.

The above requirement solves a number of issues. It clearly separates session identification from authorization. Without this separation, it is difficult to both identify a session, and change its authorization using the same attribute. It also ensures that the authorization process is the same for initial authentication, and for CoA.

When a sessions authorization is changed, the CoA server MUST continue the existing service until the new authorization parameters are applied. The change of service SHOULD be done atomically. If the CoA server is unable to apply the new authorization, it MUST terminate the user session.

3. Overview

Authorization exchanges can occur either before or after end user authentication has been completed. An authorization exchange before authentication allows a RADIUS client to provide the RADIUS server with information that MAY modify how the authentication process will be performed (e.g. it MAY affect the selection of the EAP method). An authorization exchange after authentication allows the RADIUS server to provide the RADIUS client with information about the end user, the results of the authentication process and/or obligations to be enforced. In this specification we refer to the "pre-authentication" as the exchange of authorization information before the end user authentication has started, while the term "post-authentication" is used to refer to an authorization exchange happening after this authentication process.

In this specification we refer to the "size limit" as the practical limit on RADIUS packet sizes. This limit is the minimum of 4096 octets, and the current PMTU. We define below a method which uses Access-Request and Access-Accept in order to exchange fragmented data. The NAS and server exchange a series of Access-Request / Access-Accept packets, until such time as all of the fragmented data has been transported. Each packet contains a Frag-Status attribute which lets the other party know if fragmentation is desired, ongoing, or finished. Each packet may also contain the fragmented data, or instead be an "ACK" to a previous fragment from the other party. Each Access-Request contains a User-Name attribute, allowing it to be proxied if necessary. Each Access-Request may also contain a State attribute, which serves to tie it to a previous Access-Accept. Each Access-Accept contains a State attribute, for use by the NAS in a later Access-Request. Each Access-Accept contains a Service-Type indicating that the service being provided is fragmentation, and that the Access-Accept should not be interpreted as providing network access to the end user.

When a RADIUS client or server need to send data that exceeds the size limit, the mechanism proposed in this document is used. Instead of encoding one large RADIUS packet, a series of smaller RADIUS packets of the same type are encoded. Each smaller packet is called a "chunk" in this specification, in order to distinguish it from traditional RADIUS packets. The encoding process is a simple linear walk over the attributes to be encoded. This walk preserves the order of the attributes, as required by [RFC2865]. The number of attributes encoded in a particular chunk depends on the size limit, the size of each attribute, the number of proxies between client and server, and the overhead for fragmentation signalling attributes. Specific details are given in Section 5. A new attribute called Frag-Status (Section 8.1) signals the fragmentation status.

After the first chunk is encoded, it is sent to the other party. The packet is identified as a chunk via the Frag-Status attribute. The other party then requests additional chunks, again using the Frag-Status attribute. This process is repeated until all the attributes have been sent from one party to the other. When all the chunks have been received, the original list of attributes is reconstructed and processed as if it had been received in one packet.

When multiple chunks are sent, a special situation may occur for Extended Type attributes as defined in [I-D.ietf-radext-radius-extensions]. The fragmentation process may split a fragmented attribute across two or more chunks, which is not permitted by that specification. We address this issue by defining a new field in the Reserved field of the "Long Extended Type" attribute format. This field is one bit in size, and is called "T" for

Truncation. It indicates that the attribute is intentionally truncated in this chunk, and is to be continued in the next chunk of the sequence. The combination of the flags "M" and "T" indicates that the attribute is fragmented (flag M), but that all the fragments are not available in this chunk (flag T).

This last situation is expected to be the most common occurrence in chunks. Typically, packet fragmentation will occur as a consequence of a desire to send one or more large (and therefore fragmented) attributes. The large attribute will likely be split into two or more pieces. Where chunking does not split a fragmented attribute, no special treatment is necessary.

The setting of the "T" flag is the only case where the chunking process affects the content of an attribute. Even then, the "Value" fields of all attributes remain unchanged. Any per-packet security attributes such as Message-Authenticator are calculated for each chunk independently. There are neither integrity nor security checks performed on the "original" packet.

Each RADIUS packet sent or received as part of the chunking process MUST be a valid packet, subject to all format and security requirements. This requirement ensures that a "transparent" proxy not implementing this specification can receive and send compliant packets. That is, a proxy which simply forwards packets without detailed examination or any modification will be able to proxy "chunks".

4. Fragmentation of packets

When the NAS or the AS desires to send a packet that exceeds the size limit, it is split into chunks and sent via multiple client/server exchanges. The exchange is indicated via the Frag-Status attribute, which has value More-Data-Pending for all but the last chunk of the series. The chunks are tied together via the State attribute.

The following sections describe how to perform fragmentation for packets from the NAS to the server, followed by packets from the server to the NAS. We give the packet type, along with a RADIUS Identifier, to indicate that requests and responses are connected. We then give a list of attributes. We do not give values for most attributes, as we wish to concentrate on the fragmentation behaviour, rather than packet contents. Attribute values are given for attributes relevant to the fragmentation process. Where "long extended" attributes are used, we indicate the M (More) and T (Truncation) flags as optional square brackets after the attribute name. As no "long extended" attributes have yet been defined, we use

example attributes, named as "Example-Long-1", etc. The maximum chunk size is established in term of number of attributes (11), for sake of simplicity.

4.1. Pre-authorization

When the client needs to send a large amount of data to the server, the data to be sent is split into chunks and sent to the server via multiple Access-Request / Access-Accept exchanges. The example below shows this exchange.

The following is an Access-Request which the NAS intends to send to a server. However, due to a combination of issues (PMTU, large attributes, etc.), the content does not fit into one Access-Request packet.

```
Access-Request
  User-Name
  User-Password
  Calling-Station-Id
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
```

Figure 1: Desired Access-Request

The NAS therefore must send the attributes listed above in a series of chunks. The first chunk contains eight (8) attributes from the original Access-Request, and a Frag-Status attribute. Since last attribute is "Example-Long-1" with the "M" flag set, the chunking process also sets the "T" flag in that attribute. The Access-Request is sent with a RADIUS Identifier field having value 23. The Frag-Status attribute has value More-Data-Pending, to indicate that the NAS wishes to send more data in a subsequent Access-Request. The NAS also adds a Service-Type attribute, which indicates that it is part of the chunking process. The packet is signed with the Message-Authenticator attribute, completing the maximum number of attributes (11).

```
Access-Request (ID = 23)
  User-Name
  User-Password
  Calling-Station-Id
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [MT]
  Frag-Status = More-Data-Pending
  Service-Type = Additional-Authorization
  Message-Authenticator
```

Figure 2: Access-Request (chunk 1)

Compliant servers receiving this packet will see the Frag-Status attribute, and suspend all authorization and authentication handling until all of the chunks have been received. Non-compliant servers should also see the Service-Type requesting provisioning for an unknown service, and return Access-Reject. Other non-compliant servers may return an Access-Reject, Access-Challenge, or an Access-Accept with a particular Service-Type. Compliant NAS implementations MUST treat these responses as if they had received Access-Reject instead.

Compliant servers who wish to receive all of the chunks will respond with the following packet. The value of the State here is arbitrary, and serves only as a unique token for example purposes. We only note that it MUST be globally and temporally unique.

```
Access-Accept (ID = 23)
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xabc00001
  Message-Authenticator
```

Figure 3: Access-Accept (chunk 1)

The NAS will see this response, and use the RADIUS Identifier field to associate it with an ongoing chunking session. Compliant NASes will then continue the chunking process. Non-compliant NASes will never see a response such as this, as they will never send a Frag-Status attribute. The Service-Type attribute is included in the Access-Accept in order to signal that the response is part of the chunking process. This packet therefore does not provision any network service for the end user.

The NAS continues the process by sending the next chunk, which

includes an additional six (6) attributes from the original packet. It again includes the User-Name attribute, so that non-compliant proxies can process the packet. It sets the Frag-Status attribute to More-Data-Pending, as more data is pending. It includes a Service-Type for reasons described above. It includes the State attribute from the previous Access-accept. It signs the packet with Message-Authenticator, as there are no authentication attributes in the packet. It uses a new RADIUS Identifier field.

```
Access-Request (ID = 181)
  User-Name
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [MT]
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xabc000001
  Message-Authenticator
```

Figure 4: Access-Request (chunk 2)

Compliant servers receiving this packet will see the Frag-Status attribute, and look for a State attribute. Since one exists and it matches a State sent in an Access-Accept, this packet is part of a chunking process. The server will associate the attributes with the previous chunk. Since the Frag-Status attribute has value More-Data-Request, the server will respond with an Access-Accept as before. It MUST include a State attribute, with a value different from the previous Access-Accept. This State MUST again be globally and temporally unique.

```
Access-Accept (ID = 181)
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xdef00002
  Message-Authenticator
```

Figure 5: Access-Accept (chunk 2)

The NAS will see this response, and use the RADIUS Identifier field to associate it with an ongoing chunking session. The NAS continues the chunking process by sending the next chunk, with the final attribute(s) from the original packet, and again includes the original User-Name attribute. The Frag-Status attribute is not included in the next Access-Request, as no more chunks are available

for sending. The NAS includes the State attribute from the previous Access-accept. It signs the packet with Message-Authenticator, as there are no authentication attributes in the packet. It again uses a new RADIUS Identifier field.

```
Access-Request (ID = 241)
  User-Name
  Example-Long-2
  State = 0xdef00002
  Message-Authenticator
```

Figure 6: Access-Request (chunk 3)

On reception of this last chunk, the server matches it with an ongoing session via the State attribute, and sees that there is no Frag-Status attribute present. It then process the received attributes as if they had been sent in one RADIUS packet. See Section 7.4 for further details of this process. It generates the appropriate response, which can be either Access-Accept or Access-Reject. In this example, we show an Access-Accept. The server MUST send a State attribute, which permits link the received data with the authentication process.

```
Access-Accept (ID = 241)
  State = 0x98700003
  Message-Authenticator
```

Figure 7: Access-Accept (chunk 3)

The above example shows in practice how the chunking process works. We re-iterate the implementation and security requirements here.

Each chunk is a valid RADIUS packet, and all RADIUS format and security requirements MUST be followed before any chunking process is applied.

Every chunk except for the last one from a NAS MUST include a Frag-Status attribute, with value More-Data-Pending. The last chunk MUST NOT contain a Frag-Status attribute. Each chunk except for the last from a NAS MUST include a Service-Type attribute, with value Additional-Authorization. Each chunk MUST include a User-Name attribute, which MUST be identical in all chunks. Each chunk except for the first one from a NAS MUST include a State attribute, which MUST be copied from a previous Access-Accept.

Each Access-Accept MUST include a State attribute. The value for this attribute MUST change in every new Access-Accept, and MUST be globally and temporally unique.

4.2. Post-authorization

When the AS wants to send a large amount of authorization data to the NAS after authentication, the operation is very similar to the pre-authorization one. The presence of Service-Type = Additional-Authorization attribute ensures that a NAS not supporting this specification will treat that unrecognized Service-Type as though an Access-Reject had been received instead ([RFC2865] Section 5.6). If the original large Access-Accept packet contained a Service-Type attribute, it will be included with its original value in the last transmitted chunk, to avoid confusion with the one used for fragmentation signalling.

Client supporting this specification MUST include a Frag-Status = Fragmentation-Supported attribute in the first Access-Request sent to the server, in order to indicate they would accept fragmented data from the sever. This is not required if pre-authorization process was carried out, as it is implicit.

The following is an Access-Accept which the AS intends to send to a client. However, due to a combination of issues (PMTU, large attributes, etc.), the content does not fit into one Access-Accept packet.

```
Access-Accept
  User-Name
  EAP-Message
  Service-Type(Login)
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
```

Figure 8: Desired Access-Accept

The AS therefore must send the attributes listed above in a series of chunks. The first chunk contains eight (7) attributes from the original Access-Accept, and a Frag-Status attribute. Since last attribute is "Example-Long-1" with the "M" flag set, the chunking process also sets the "T" flag in that attribute. The Access-Accept

is sent with a RADIUS Identifier field having value 30 corresponding to a previous Access-Request not depicted. The Frag-Status attribute has value More-Data-Pending, to indicate that the AS wishes to send more data in a subsequent Access-Accept. The AS also adds a Service-Type attribute with value Additional-Authorization, which indicates that it is part of the chunking process. Note that the original Service-Type is not included in this chunk. Finally, a State attribute is included to allow matching subsequent requests with this conversation, and the packet is signed with the Message-Authenticator attribute, completing the maximum number of attributes of 11.

```
Access-Accept (ID = 30)
  User-Name
  EAP-Message
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [MT]
  Frag-Status = More-Data-Pending
  Service-Type = Additional-Authorization
  State = 0xcba00004
  Message-Authenticator
```

Figure 9: Access-Accept (chunk 1)

Compliant clients receiving this packet will see the Frag-Status attribute, and suspend all authorization and authentication handling until all of the chunks have been received. Non-compliant clients should also see the Service-Type indicating the provisioning for an unknown service, and will treat it as an Access-Reject.

Clients who wish to receive all of the chunks will respond with the following packet, where the value of the State attribute is taken from the received Access-Accept. They also include the User-Name attribute so that non-compliant proxies can process the packet.

```
Access-Request (ID = 131)
  User-Name
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xcba00004
  Message-Authenticator
```

Figure 10: Access-Request (chunk 1)

The AS receives this request, and uses the State attribute to associate it with an ongoing chunking session. Compliant ASes will

then continue the chunking process. Non-compliant ASes will never see a response such as this, as they will never send a Frag-Status attribute.

The AS continues the chunking process by sending the next chunk, with the final attribute(s) from the original packet. The value of the Identifier field is taken from the received Access-Request. A Frag-Status attribute is not included in the next Access-Accept, as no more chunks are available for sending. The AS includes an State attribute to allow the client to send additional authorization data. The original Service-Type attribute is included in this final chunk.

```
Access-Accept (ID = 131)
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
  Service-Type = Login
  State = 0xfda000005
  Message-Authenticator
```

Figure 11: Access-Accept (chunk 2)

On reception of this last chunk, the client matches it with an ongoing session via the Identifier field, and sees that there is no Frag-Status attribute present. It then processes the received attributes as if they had been sent in one RADIUS packet. See Section 7.4 for further details of this process.

5. Chunk size

In an ideal scenario, each intermediate chunk would be exactly the size limit in length. In this way, the number of round trips required to send a large packet would be optimal. However, this is not possible for several reasons.

1. RADIUS attributes have a variable length, and must be included completely in a chunk. Thus, it is possible that, even if there is some free space in the chunk, it is not enough to include the next attribute. This can generate up to 254 octets of spare space on every chunk.
2. RADIUS fragmentation requires the introduction of some extra attributes for signalling. Specifically, a Frag-Status attribute

(7 octets) is included on every chunk of a packet, except the last one. A RADIUS State attribute (from 3 to 255 octets) is also included in most chunks, to allow the server to bind an Access-Request with a previous Access-Challenge. User-Name attributes (from 3 to 255 octets) are introduced on every chunk the client sends as they are required by the proxies to route the packet to its destination. Together, these attributes can generate from up to 13 to 517 octets of signalling data, reducing the amount of payload information that can be sent on each chunk.

3. RADIUS packets SHOULD be adjusted to avoid exceeding the network MTU. Otherwise, IP fragmentation may occur, having undesirable consequences. Hence, maximum chunk size would be decreased from 4096 to the actual MTU of the network.
4. The inclusion of Proxy-State attributes by intermediary proxies can decrease the availability of usable space into the chunk. This is described with further detail in Section 7.1.

6. Allowed large packet size

There are no provisions for signalling how much data is to be sent via the fragmentation process as a whole. It is difficult to define what is meant by the "length" of any fragmented data. That data can be multiple attributes, which includes RADIUS attribute header fields. Or it can be one or more "large" attributes (more than 256 octets in length). Proxies can also filter these attributes, to modify, add, or delete them and their contents. These proxies act on a "packet by packet" basis, and cannot know what kind of filtering actions they take on future packets. As a result, it is impossible to signal any meaningful value for the total amount of additional data.

Unauthenticated clients are permitted to trigger the exchange of large amounts of fragmented data between the NAS and the AS, having the potential to allow Denial of Service (DoS) attacks. An attacker could initiate a large number of connections, each of which requests the server to store a large amount of data. This data could cause memory exhaustion on the server, and result in authentic users being denied access. It is worth noting that authentication mechanisms are already designed to avoid exceeding the size limit.

Hence, implementations of this specification MUST limit the total amount of data they send and/or receive via this specification. It is RECOMMENDED that the limits be set to a few tens of kilooctets. Any more than this may turn RADIUS into a generic transport protocol, which is undesired. It is RECOMMENDED that this limit be exposed to

administrators, so that it can be changed if necessary.

Implementations of this specification MUST limit the total number of round trips used during the fragmentation process. It is RECOMMENDED that the number of round trips be limited to twenty (20). Any more than this may indicate an implementation error, misconfiguration, or a denial of service (DoS) attack. It is RECOMMENDED that this limit be exposed to administrators, so that it can be changed if necessary.

7. Handling special attributes

7.1. Proxy-State attribute

RADIUS proxies may introduce Proxy-State attributes into any Access-Request packet they forward. Should they cannot add this information to the packet, they may silently discard forwarding it to its destination, leading to DoS situations. Moreover, any Proxy-State attribute received by a RADIUS server in an Access-Request packet MUST be copied into the reply packet to it. For these reasons, Proxy-State attributes require a special treatment within the packet fragmentation mechanism.

When the RADIUS server replies to an Access-Request packet as part of a conversation involving a fragmentation (either a chunk or a request for chunks), it MUST include every Proxy-State attribute received into the reply packet. This means that the server MUST take into account the size of these Proxy-State attributes in order to calculate the size of the next chunk to be sent.

However, while a RADIUS server will always know how many space MUST be left on each reply packet for Proxy-State attributes (as they are directly included by the RADIUS server), a RADIUS client cannot know this information, as Proxy-State attributes are removed from the reply packet by their respective proxies before forwarding them back. Hence, clients need a mechanism to discover the amount of space required by proxies to introduce their Proxy-State attributes. In the following we describe a new mechanism to perform such a discovery:

1. When a RADIUS client does not know how many space will be required by intermediate proxies for including their Proxy-State attributes, it SHOULD start using a conservative value (e.g. 1024 octets) as the chunk size.
2. When the RADIUS server receives a chunk from the client, it can calculate the total size of the Proxy-State attributes that have been introduced by intermediary proxies along the path. This

information MUST be returned to the client in the next reply packet, encoded into a new attribute called Proxy-State-Len.

3. The RADIUS client reacts upon the reception of this attribute by adjusting the maximum size for the next chunk accordingly.

7.2. State attribute

This RADIUS fragmentation mechanism makes use of the State attribute to link all the chunks belonging to the same fragmented packet. However, some considerations are required when the RADIUS server is fragmenting a packet that already contains a State attribute for other purposes not related with the fragmentation. If the procedure described in Section 4 is followed, two different State attributes could be included into a single chunk, incurring into two problems. First, [RFC2865] explicitly forbids that more than one State attribute appears into a single packet.

A straightforward solution consists on making the RADIUS server to send the original State attribute into the last chunk of the sequence (attributes can be re-ordered as specified in [RFC2865]). As the last chunk (when generated by the RADIUS server) does not contain any State attribute due to the fragmentation mechanism, both situations described above are avoided.

Something similar happens when the RADIUS client has to send a fragmented packet that contains a State attribute on it. The client MUST assure that this original State is included into the first chunk sent to the server (as this one never contains any State attribute due to fragmentation).

7.3. Service-Type attribute

This RADIUS fragmentation mechanism makes use of the Service-Type attribute to indicate an Access-Accept packet is not granting access to the service yet, since additional authorization exchange needs to be performed. Similarly to the State attribute, the RADIUS server has to send the original Service-Type attribute into the last Access-Accept of the RADIUS conversation to avoid ambiguity.

7.4. Rebuilding the original large packet

The RADIUS client stores the RADIUS attributes received on each chunk in order to be able to rebuild the original large packet after receiving the last chunk. However, some of these received attributes MUST NOT be stored in this list, as they have been introduced as part of the fragmentation signalling and hence, they are not part of the original packet.

- o State (except the one in the last chunk, if present)
- o Service-Type = Additional-Authorization
- o Frag-Status
- o Proxy-State-Len

Similarly, the RADIUS server MUST NOT store the following attributes as part of the original large packet:

- o State (except the one in the first chunk, if present)
- o Frag-Status
- o Proxy-State (except the ones in the last chunk)
- o User-Name (except the one in the first chunk)

8. New attribute definition

This document proposes the definition of two new extended type attributes, called Frag-Status and Proxy-State-Len. The format of these attributes follows the indications for an Extended Type attribute defined in [I-D.ietf-radext-radius-extensions].

8.1. Frag-Status attribute

This attribute is used for fragmentation signalling, and its meaning depends on the code value transported within it. The following figure represents the format of the Frag-Status attribute.

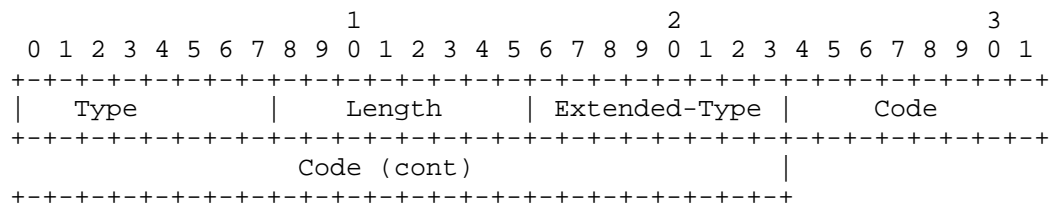


Figure 12: Frag-Status format

Type

To be assigned (TBA)

Length

7

Extended-Type

To be assigned (TBA).

Code

4 byte. Integer indicating the code. The values defined in this specifications are:

- 0 - Reserved
- 1 - Fragmentation-Supported
- 2 - More-Data-Pending
- 3 - More-Data-Request

This attribute MAY be present in Access-Request, Access-Challenge and Access-Accept packets. It MUST not be included in Access-Reject packets.

8.2. Proxy-State-Len attribute

This attribute indicates to the RADIUS client the length of the Proxy-State attributes received by the RADIUS server. This information is useful to adjust the length of the chunks sent by the RADIUS client. The format of this Proxy-State-Len attribute is the following:

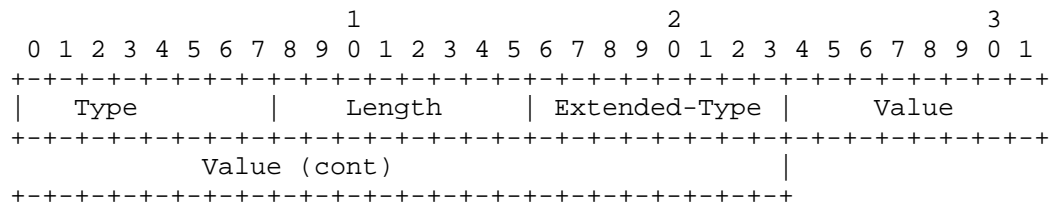


Figure 13: Proxy-State-Len format

Type

To be assigned (TBA)

Length

7

Extended-Type

To be assigned (TBA).

Value

4 octets. Total length (in octets) of received Proxy-State attributes (including headers).

This attribute MAY be present in Access-Challenge and Access-Accept packets. It MUST not be included in Access-Request or Access-Reject packets.

8.3. Table of attributes

The following table shows the different attributes defined in this document related with the kind of RADIUS packets where they can be present.

Attribute Name	Kind of packet			
	Req	Acc	Rej	Cha
Frag-Status	0-1	0-1	0	0-1
Proxy-State-Len	0	0-1	0	0-1

Figure 14

9. Operation with proxies

The fragmentation mechanism defined above is designed to be transparent to legacy proxies, as long as they do not want to modify any fragmented attribute. Nevertheless, updated proxies supporting this specification can even modify fragmented attributes.

9.1. Legacy proxies

As every chunk is indeed a RADIUS packet, legacy proxies treat them as the rest of packets, routing them to their destination. Proxies can introduce Proxy-State attributes to Access-Request packets, even if they are indeed chunks. This will not affect how fragmentation is managed. The server will include all the received Proxy-State attributes into the generated response, as described in [RFC2865].

Hence, proxies do not distinguish between a regular RADIUS packet and a chunk.

9.2. Updated proxies

Updated proxies can interact with clients and servers in order to obtain the complete large packet before start forwarding it. In this way, proxies can manipulate (modify and/or remove) any attribute of the packet, or introduce new attributes, without worrying about crossing the boundaries of the chunk size. Once the manipulated packet is ready, it is sent to the original destination using the fragmentation mechanism (if required). The following example shows how an updated proxy interacts with the NAS to obtain a large Access-Request packet, modify an attribute resulting into a even more large packet, and interacts with the AS to complete the transmission of the modified packet.

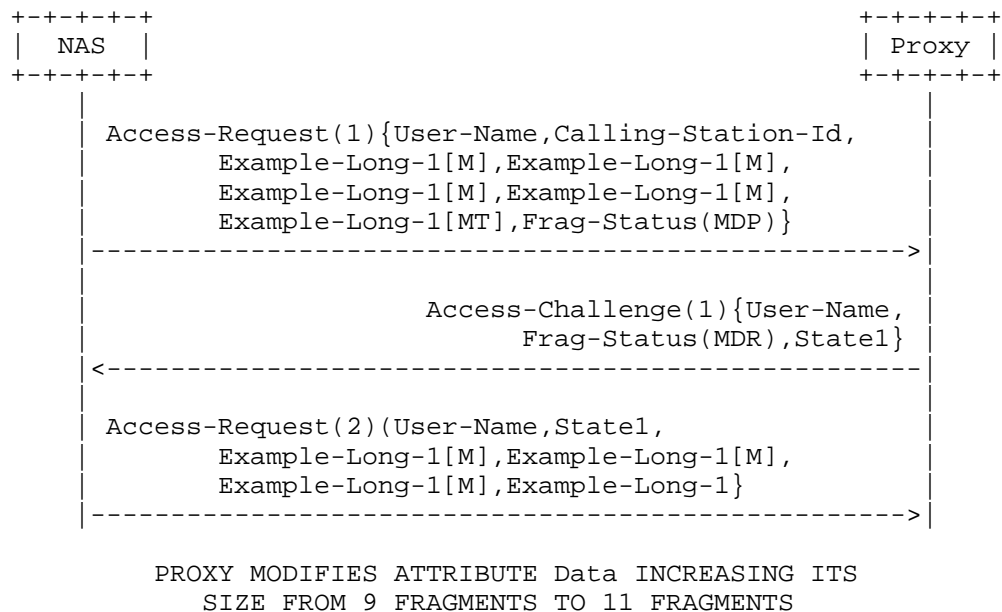


Figure 15: Updated proxy interacts with NAS

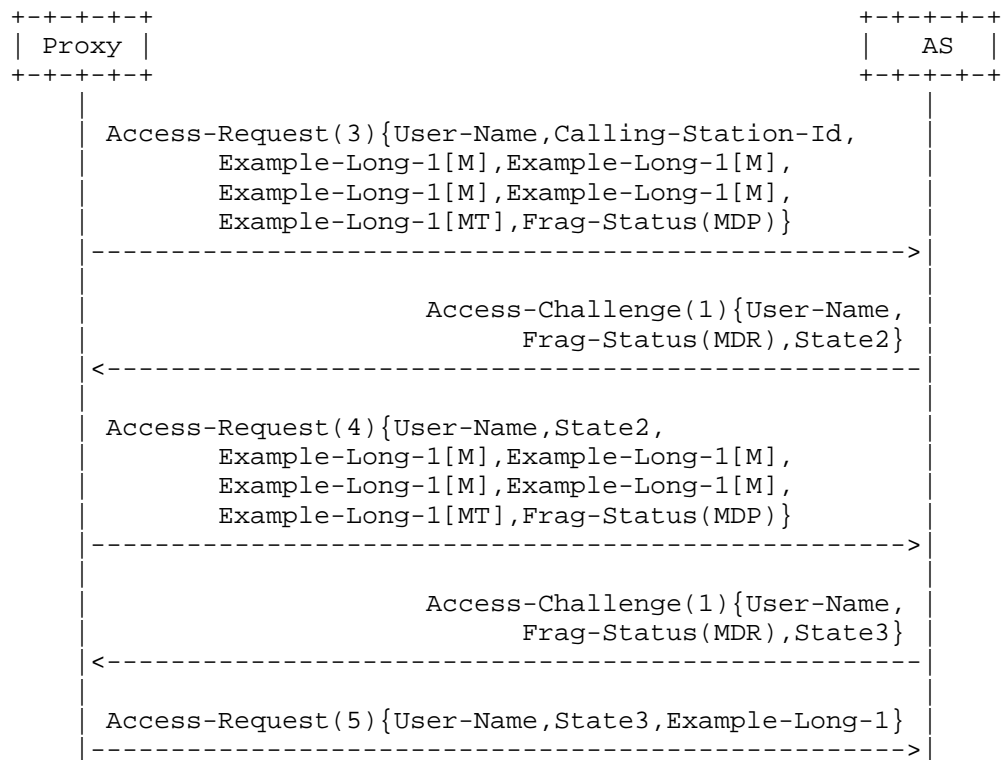


Figure 16: Updated proxy interacts with AS

10. Security Considerations

As noted in many earlier specifications ([RFC5080], [RFC6158], etc.) RADIUS security is problematic. This specification changes nothing related to the security of the RADIUS protocol. It requires that all Access-Request packets associated with fragmentation are signed using the existing Message-Authenticator attribute. This signature prevents forging and replay, to the limits of the existing security.

The ability to send bulk data from one party to another creates new security considerations. Clients and servers may have to store large amounts of data per session. The amount of this data can be significant, leading to the potential for resource exhaustion. We therefore suggest that implementations limit the amount of bulk data stored per session. The exact method for this limitation is implementation-specific. Section 6 gives some indications on what could be a reasonable limits.

The bulk data can often be pushed off to storage methods other than the memory of the RADIUS implementation. For example, it can be stored in an external database, or in files. This approach mitigates the resource exhaustion issue, as servers today already store large amounts of accounting data.

11. IANA Considerations

The authors request that Attribute Types and Attribute Values defined in this document be registered by the Internet Assigned Numbers Authority (IANA) from the RADIUS namespaces as described in the "IANA Considerations" section of [RFC3575], in accordance with BCP 26 [RFC5226]. For RADIUS packets, attributes and registries created by this document IANA is requested to place them at <http://www.iana.org/assignments/radius-types>.

This document defines the following RADIUS messages:

- o Frag-Status
- o Proxy-State-Len

Additionally, allocation of a new Service-Type value for "Additional-Authorization" is requested.

12. References

12.1. Normative References

- [I-D.ietf-radext-radius-extensions]
DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions",
draft-ietf-radext-radius-extensions-13 (work in progress),
February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575,

July 2003.

- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.
- [RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6158] DeKok, A. and G. Weber, "RADIUS Design Guidelines", BCP 158, RFC 6158, March 2011.

12.2. Informative References

- [RFC4849] Congdon, P., Sanchez, M., and B. Aboba, "RADIUS Filter Rule Attribute", RFC 4849, April 2007.

Authors' Addresses

Alejandro Perez-Mendez (Ed.)
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 46 44
Email: alex@um.es

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 85 01
Email: rafa@um.es

Fernando Pereniguez-Garcia
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 78 82
Email: pereniguez@um.es

Gabriel Lopez-Millan
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 85 04
Email: gabilm@um.es

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 84
Madrid, 28006
Spain

Phone: +34 913 129 041
Email: diego@tid.es

Alan DeKok
Network RADIUS
15 av du Granier
Meylan, 38240
France

Phone: +34 913 129 041
Email: aland@networkradius.com
URI: <http://networkradius.com>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

K. Wierenga
Cisco Systems
S. Winter
RESTENA
T. Wolniewicz
Nicolaus Copernicus University
March 9, 2015

The eduroam architecture for network roaming
draft-wierenga-ietf-eduroam-05.txt

Abstract

This document describes the architecture of the eduroam service for federated (wireless) network access in academia. The combination of IEEE 802.1X, EAP and RADIUS that is used in eduroam provides a secure, scalable and deployable service for roaming network access. The successful deployment of eduroam over the last decade in the educational sector may serve as an example for other sectors, hence this document. In particular the initial architectural and standards choices are described, along with the changes that were prompted by operational experience.

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 <http://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 September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Notational Conventions	4
1.3. Design Goals	4
1.4. Solutions that were considered	5
2. Classic Architecture	5
2.1. Authentication	6
2.1.1. IEEE 802.1X	6
2.1.2. EAP	7
2.2. Federation Trust Fabric	9
2.2.1. RADIUS	9
3. Issues with initial Trust Fabric	11
3.1. Server Failure Handling	12
3.2. No error condition signalling	13
3.3. Routing table complexity	14
3.4. UDP Issues	15
3.5. Insufficient payload encryption and EAP server validation	16
4. New Trust Fabric	17
4.1. RADIUS with TLS	18
4.2. Dynamic Discovery	19
4.2.1. Discovery of responsible server	19
4.2.2. Verifying server authorisation	20
4.2.3. Operational Experience	21
4.2.4. Possible Alternatives	21
5. Abuse prevention and incident handling	21
5.1. Incident Handling	22
5.1.1. Blocking users on the SP side	23
5.1.2. Blocking users on the IdP side	24
5.1.3. Communicating account blocking to the end user	25
5.2. Operator Name	25
5.3. Chargeable User Identity	26
6. Privacy Considerations	27
6.1. Collusion of Service Providers	27
6.2. Exposing user credentials	28
6.3. Track location of users	28
7. Security Considerations	28
7.1. Man in the middle and Tunneling Attacks	28
7.1.1. Verification of Server Name not supported	29

7.1.2.	Neither Specification of CA nor Server Name checks during bootstrap	29
7.1.3.	User does not configure CA or Server Name checks	29
7.1.4.	Tunneling authentication traffic to obfuscate user origin	30
7.2.	Denial of Service Attacks	30
7.2.1.	Intentional DoS by malign individuals	31
7.2.2.	DoS as a side-effect of expired credentials	31
8.	IANA Considerations	32
9.	References	32
9.1.	Normative References	32
9.2.	Informative References	34
Appendix A.	Acknowledgments	37
Appendix B.	Changes	37
Authors' Addresses	37

1. Introduction

In 2002 the European Research and Education community set out to create a network roaming service for students and employees in academia [eduroam-start]. Now over 10 years later this service has grown to more than 10,000 service locations, serving millions of users on all continents with the exception of Antarctica.

This memo serves to explain the considerations for the design of eduroam as well as to document operational experience and resulting changes that led to IETF standardization effort such as RADIUS over TCP [RFC6613] and RADIUS with TLS [RFC6614] and that promoted alternative uses of RADIUS like in ABFAB [I-D.ietf-abfab-arch]. Whereas the eduroam service is limited to academia, the eduroam architecture can easily be reused in other environments.

First this memo describes the original architecture of eduroam. Then a number of operational problems are presented that surfaced when eduroam gained wide-scale deployment. Lastly, enhancements to the eduroam architecture that mitigate the aforementioned issues are discussed.

1.1. Terminology

This document uses identity management and privacy terminology from [RFC6973]. In particular, this document uses the terms Identity Provider, Service Provider and identity management.

1.2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Note: Also the policy that eduroam participants subscribe to, expresses the requirements for participation in RFC 2119 language.

1.3. Design Goals

The guiding design considerations for eduroam were as follows:

- Unique identification of users at the edge of the network

The access Service Provider (SP) needs to be able to determine whether a user is authorized to use the network resources. Furthermore, in case of abuse of the resources, there is a requirement to be able to identify the user uniquely (with the cooperation of the user's Identity Provider (IdP) operator).

- Enable (trusted) guest use

In order to enable roaming it should be possible for users of participating institutions to get seamless access to the networks of other institutions.

Note: traffic separation between guest users and normal users is possible (for example through the use of VLANs), and indeed widely used in eduroam.

- Scalable

The infrastructure that is created should scale to a large number of users and organizations without requiring a lot of coordination and other administrative procedures (possibly with the exception of an initial set up). Specifically, it should not be necessary for a user that visits another organization to go through an administrative process.

- Easy to install and use

It should be easy for both organizations and users to participate in the roaming infrastructure as that may otherwise inhibit wide scale adoption. In particular, there should be no or easy client installation and only one-time configuration.

- Secure

An important design criterion has been that there needs to be a security association between the end-user and their Identity Provider, eliminating the possibility of credentials theft. The minimal requirements for security are specified in the eduroam policy and subject to change over time. As an additional protection against user errors and negligence, it should be possible for participating Identity Providers add their own requirements for the quality of authentication of their own users without the need for the infrastructure as a whole to implement the same standard.

- Privacy preserving

The design of the system should provide for user anonymization, i.e. a possibility to hide the user's identity from any third parties, including Service Providers.

- Standards based

In an infrastructure in which many thousands of organizations participate it is obvious that it should be possible to use equipment from different vendors, therefore it is important to build the infrastructure using open standards.

1.4. Solutions that were considered

Three architectures were trialed: one based on the use of VPN-technology (deemed secure but not-scalable), one Web captive-portal based (scalable but not secure) and IEEE 802.1X-based, the latter being the basis of what is now the eduroam architecture. An overview of the candidate architectures and their relative merits can be found in [nrenroaming-select].

The chosen architecture is based on:

- o IEEE 802.1X ([dot1X-standard]) as port based authentication framework using
- o EAP ([RFC3748]) for integrity and confidentially protected transport of credentials and a
- o RADIUS ([RFC2865]) hierarchy as trust fabric.

2. Classic Architecture

Federations, like eduroam, implement essentially two types of direct trust relations (and one indirect). The trust relation between an end-user and the IdP (operated by the home organization of the user) and between the IdP and the SP (in eduroam the operator of the

network at the visited location). In eduroam the trust relation between user and IdP is through mutual authentication. IdPs and SP establish trust through the use of a RADIUS hierarchy.

These two forms of trust relations in turn provide the transitive trust relation that makes the SP trust the user to use its network resources.

2.1. Authentication

Authentication in eduroam is achieved by using a combination of IEEE 802.1X [dot1X-standard] and EAP [RFC4372] (the latter carried over RADIUS for guest access, see below).

2.1.1. IEEE 802.1X

By using the IEEE 802.1X [dot1X-standard] framework for port-based network authentication, organizations that offer network access (SPs) for visiting (and local) eduroam users can make sure that only authorized users get access. The user (or rather the user's supplicant) sends an access request to the authenticator (wireless access point or switch) at the SP, the authenticator forwards the access request to the authentication server of the SP which in turn proxies the request through the RADIUS hierarchy to the authentication server of the user's home organization (the IdP, see below).

Note: The security of the connections between local wireless infrastructure and local RADIUS servers is a part of the local network of each SP, therefore it is out of scope for this document. For completeness it should be stated that security between access points and their controllers is vendor specific, security between controllers (or standalone access points) and local RADIUS servers is based on the typical RADIUS shared secret mechanism.

In order for users to be aware of the availability of the eduroam service, an SP that offers wireless network access MUST broadcast the SSID 'eduroam', unless that conflicts with the SSID of another eduroam SP, in which case an SSID starting with "eduroam-" MAY be used. The downside of the latter is that clients will not automatically connect to that SSID, thus losing the seamless connection experience.

Note: A direct implication of the common eduroam SSID is that the users cannot distinguish between a connection to the home network and a guest network at another eduroam institution (IEEE 802.11-2012 does have the so-called "Interworking" extensions to make that distinction, but these are not widely implemented yet). Furthermore,

without proper server verification users may even be tricked into joining a rogue eduroam network. Therefore, users should be made aware that they should not assume data confidentiality in the eduroam infrastructure.

To protect over-the-air user data confidentiality, IEEE 802.11 wireless networks of eduroam SP's MUST deploy WPA2+AES, and MAY additionally support WPA/TKIP as a courtesy to users of legacy hardware.

2.1.2. EAP

The use of the Extensible Authentication Protocol (EAP) [RFC4372] serves 2 purposes. In the first place a properly chosen EAP-method allows for integrity and confidentiality protected transport of the user credentials to the home organization. Secondly, by having all RADIUS servers transparently proxy access requests regardless of the EAP-method inside the RADIUS packet, the choice of EAP-method is between the 'home' organization of the user and the user, in other words, in principle every authentication form that can be carried inside EAP can be used in eduroam, as long as they adhere to minimal requirements as set forth in the eduroam Service Definition [eduroam-service-definition].

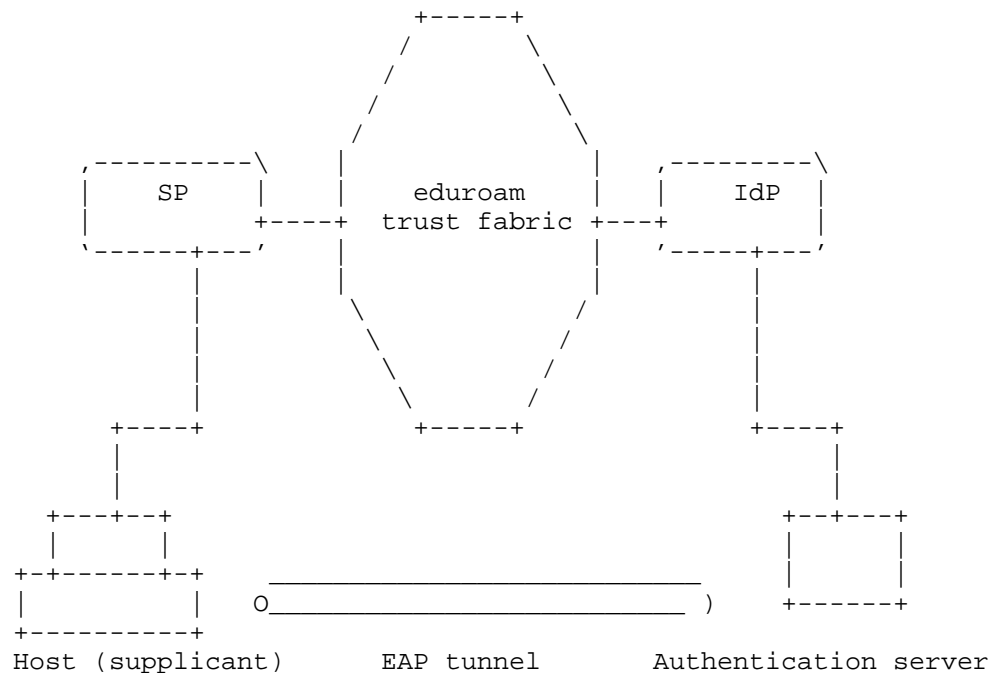


Figure 1: Tunnelled EAP

Proxying of access requests is based on the outer identity in the EAP-message. Those outer identities MUST be a valid user identifier with a mandatory realm as per [I-D.ietf-radext-nail], i.e. be of the form something@realm or just @realm, where the realm part is the domain name of the institution that the IdP belongs to. In order to preserve credentials protection, participating organizations MUST deploy EAP-methods that provide mutual authentication. For EAP methods that support outer identity, anonymous outer identities are recommended. Most commonly used in eduroam are the so-called tunneled EAP-methods, that first create a server authenticated TLS tunnel through which the user credentials are transmitted. As depicted in Figure 1, the use of a tunneled EAP-method creates a direct logical connection between the supplicant and the authentication server, even though the actual traffic flows through the RADIUS-hierarchy.

2.2. Federation Trust Fabric

The eduroam federation trust fabric is based on RADIUS. RADIUS trust is based on shared secrets between RADIUS peers. In eduroam any RADIUS message originating from a trusted peer is implicitly assumed to originate from a member of the roaming consortium.

Note: See also the security considerations for a discussion on RADIUS security that motivated the work on RADIUS with TLS (RFC6614 [RFC6614])

2.2.1. RADIUS

The eduroam trust fabric consists of a proxy hierarchy of RADIUS servers (organizational, national, global), loosely based on the DNS hierarchy. That is, typically an organizational RADIUS server agrees on a shared secret with a national server and the national server in turn agrees on a shared secret with the root server. Access requests are routed through a chain of RADIUS proxies towards the Identity Provider of the user, and the access accept (or reject) follows the same path back.

Note: In some circumstances there are more levels of RADIUS servers, like for example regional or continental servers, but that doesn't change the general model. Also, the packet exchange that is described below requires in reality several round-trips.

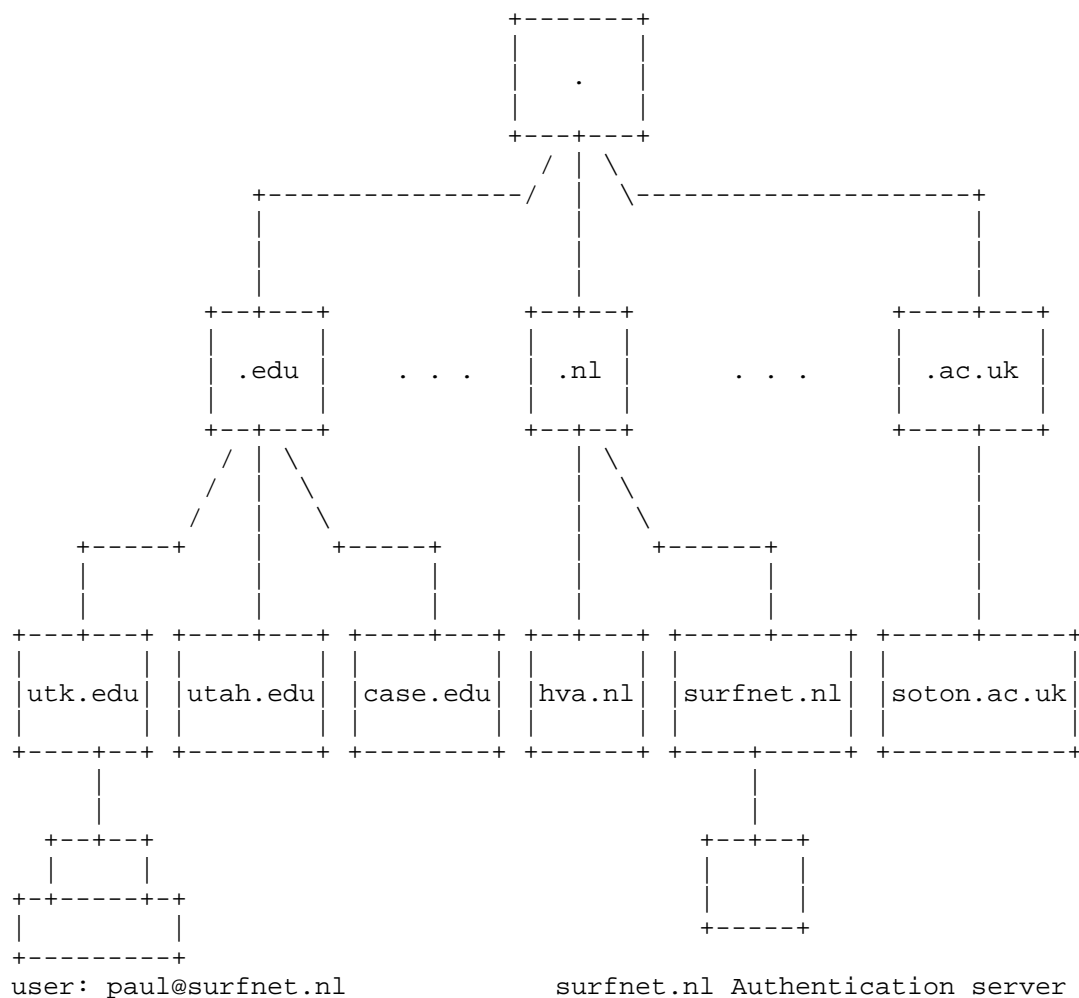


Figure 2: eduroam RADIUS hierarchy

Routing of access requests to the home IdP is done based on the realm part of the outer identity. For example (see: Figure 2), when user paul@surfnet.nl of SURFnet (surfnet.nl) tries to gain wireless network access at the University of Tennessee at Knoxville (utk.edu) the following happens:

- o Paul's supplicant transmits an EAP access request to the Access Point (Authenticator) at UTK with outer identity say anonymous@surfnet.nl

- o The Access Point forwards the EAP message to its Authentication Server (the UTK RADIUS server)
- o The UTK RADIUS server checks the realm to see if it is a local realm, since it isn't the request is proxied to the .edu RADIUS server
- o The .edu RADIUS server verifies the realm, and since it is not a in a .edu subdomain it proxies the request to the root server
- o The root RADIUS server proxies the request to the .nl RADIUS server, since the ".nl" domain is known to the root server.
- o The .nl RADIUS server proxies the request to the surfnet.nl server, since it knows the SURFnet server.
- o The surfnet.nl RADIUS server decapsulates the EAP message and verifies the user credentials, since the user is known to SURFnet.
- o The surfnet.nl RADIUS server informs the utk.edu server of the outcome of the authentication request (Access-Accept or Access-Reject) by proxying the outcome through the RADIUS hierarchy in reverse order.
- o The UTK RADIUS server instructs the UTK Access Point to either accept or reject access based on the outcome of the authentication.

Note: The depiction of the root RADIUS server is a simplification. In reality the root server is distributed over 3 continents and each maintains a list of the top level realms that a specific root server is responsible for. This also means that, for intercontinental roaming, there is an extra proxy step from one root server to the other. Also, the physical distribution of nodes doesn't need to mirror the logical distribution of nodes. This helps with stability and scalability.

3. Issues with initial Trust Fabric

While the hierarchical RADIUS architecture described in the previous section has served as the basis for eduroam operations for an entire decade, the exponential growth of authentications is expected to lead to, and has in fact in some cases already led to, performance and operations bottlenecks on the aggregation proxies. The following sections describe some of the shortcomings, and the resulting remedies.

3.1. Server Failure Handling

In eduroam, authentication requests for roaming users are statically routed through pre-configured proxies. The number of proxies varies: in a national roaming case, the number of proxies is typically 1 or 2 (some countries deploy regional proxies, which are in turn aggregated by a national proxy); in international roaming, 3 or 4 proxy servers are typically involved (the number may be higher along some routes).

RFC2865 [RFC2865] does not define a failover algorithm. In particular, the failure of a server needs to be deduced from the absence of a reply. Operational experience has shown that this has detrimental effects on the infrastructure and end user experience:

1. Authentication failure: the first user whose authentication path is along a newly-failed server will experience a long delay and possibly timeout
2. Wrongly deduced states: since the proxy chain is longer than 1 hop, a failure further along in the authentication path is indistinguishable from a failure in the next hop.
3. Inability to determine recovery of a server: only a "live" authentication request sent to a server which is believed inoperable can lead to the discovery that the server is in working order again. This issue has been resolved with RFC5997 [RFC5997].

The second point can have significant impact on the operational state of the system in a worst-case scenario: Imagine one realm's home server being inoperable. A user from that realm is trying to roam internationally and tries to authenticate. The RADIUS server on the hotspot location may assume its own national proxy is down, because it does not reply. That national server, being perfectly alive, in turn will assume that the international aggregation proxy is down; which in turn will believe the home country proxy national server is down. None of these assumptions are true. Worse yet: should any of these servers trigger a failover to a redundant backup RADIUS server, it will still not receive a reply, because the request will still be routed to the same defunct home server. Within a short time, all redundant aggregation proxies might be considered defunct by their preceding hop.

In the absence of proper next-hop state derivation, some interesting concepts have been introduced by eduroam participants; the most noteworthy being a failover logic which considers up/down states not per next-hop RADIUS peer, but instead per realm (See [dead-realm] for details). As of recent, RFC5997 [RFC5997] implementations and

cautious failover parameters make false "downs" unlikely to happen, as long as every hop implements RFC5997. Dead realm detection in that case serves mainly to prevent proxying of large numbers of requests to known dead realms.

3.2. No error condition signalling

The RADIUS protocol lacks signalling of error conditions, and the IEEE 802.1X protocol does not allow to convey extended failure reasons to the end-user's device. For eduroam, this creates issues in a twofold way:

- o The home server may have an operational problem, for example its authentication decisions may depend on an external data source such as ActiveDirectory or an SQL server, and the external data source is unavailable. If the RADIUS interface is still functional, there are two options how to reply to an Access-Request which can't be serviced due to such error conditions:
 - 1. Do Not Reply: the inability to reach a conclusion can be treated by not replying to the request. The upside of this approach is that the end-user's software doesn't come to wrong conclusions and won't give unhelpful hints such as "maybe your password is wrong". The downside is that intermediate proxies may come to wrong conclusions because their downstream RADIUS server isn't responding.
 - 2. Reply with Reject: in this option, the inability to reach a conclusion is treated like an authentication failure. The upside of this approach is that intermediate proxies maintain a correct view on the reachability state of their RADIUS peer. The downside is that EAP supplicants on end-user devices often react with either false advice ("your password is wrong") or even trigger permanent configuration changes (e.g. the Windows built-in supplicant will delete the credential set from its registry, prompting the user for their password on the next connection attempt). The latter case of Windows is a source of significant helpdesk activity; users may have forgotten their password after initially storing it, but are suddenly prompted again.

There have been epic discussions in the eduroam community as well as in the IETF RADEXT Working Group as to which of the two approaches is more appropriate; but they were not conclusive.

- o Similar considerations as above apply when an intermediate proxy does not receive a reply from a downstream RADIUS server. The proxy may either choose not to reply to the original request,

leading to retries and its upstream peers coming to wrong conclusions about its own availability; or it may decide to reply with Access-Reject to indicate its own liveness, but again with implications for the end user.

The ability to send Status-Server watchdog requests is only of use after the fact, in case a downstream server doesn't reply (or hasn't been contacted in a long while, so that it's previous working state is stale). The active link-state monitoring of the TCP connection with e.g. RADIUS/TLS (see below) gives a clearer indication whether there is an alive RADIUS peer, but does not solve the defunct backend problem. An explicit ability to send Error-Replies, on the RADIUS (for other RADIUS peer information) and EAP level (for end-user supplicant information), would alleviate these problems but is currently not available.

3.3. Routing table complexity

The aggregation of RADIUS requests based on the structure of the user's realm implies that realms ending with the same top-level domain are routed to the same server; i.e. to a common administrative domain. While this is true for country code Top Level Domains (ccTLDs), which map into national eduroam federations, it is not true for realms residing in generic Top Level Domains (gTLDs). Realms in gTLDs were historically discouraged because the automatic mapping "realm ending" -> "eduroam federation's server" could not be applied. However, with growing demand from eduroam realm administrators, it became necessary to create exception entries in the forwarding rules; such realms need to be mapped on a realm-by-realm basis to their eduroam federations. Example: "kit.edu" (Karlsruher Institut fuer Technologie) needs to be routed to the German federation server, whereas "iu.edu" (Indiana University) needs to be routed to the USA federation server.

While the ccTLDs occupy only approx. 50 routing entries in total (and have a upper bound of approx. 200), the potential size of the routing table becomes virtually unlimited if it needs to accomodate all individual entries in .edu, .org, etc.

In addition to that, all these routes need to be synchronised between three international root servers, and the updates need to be applied manually to RADIUS server configuration files. The frequency of the required updates makes this approach fragile and error-prone as the number of entries grows.

3.4. UDP Issues

RADIUS is based on UDP, which was a reasonable choice when its main use was with simple PAP requests which required only exactly one packet exchange in each direction.

When transporting EAP over RADIUS, the EAP conversations requires multiple round-trips; depending on the total payload size, 8-10 round-trips are not uncommon. The loss of a single UDP packet will lead to user-visible delays and might result in servers being marked as dead due to the absence of a reply. The proxy path in eduroam consists of several proxies, all of which introduce a very small packet loss probability; i.e. the more proxies are needed, the higher the failure rate is going to be.

For some EAP types, depending on the exact payload size they carry, RADIUS servers and/or supplicants may choose to fill as much EAP data into a single RADIUS packet as the supplicant's layer 2 medium allows for, typically 1500 Bytes. In that case, the RADIUS encapsulation around the EAP-Message will add more bytes to the overall RADIUS payload size and in the end exceed the 1500 Byte limit, leading to fragmentation of the UDP datagram on the IP layer. While this is not a problem in theory, practice has shown evidence of misbehaving firewalls which erroneously discard non-first UDP fragments, which ultimately leads to a denial of service for users with such EAP types and that specific configuration.

One EAP type proved to be particularly problematic: EAP-TLS. While it is possible to configure the EAP server to send smaller chunks of EAP payload to the supplicant (e.g. 1200 Bytes, to allow for another 300 Bytes of RADIUS overhead without fragmentation), very often the supplicants which send the client certificate do not expose such a configuration detail to the user. Consequently, when the client certificate is beyond 1500 Bytes in size, the EAP-Message will always make use of the maximum possible layer-2 chunk size, which introduces the fragmentation on the path from EAP peer to EAP server.

Both of the previously mentioned sources of errors (packet loss, fragment discard) lead to significant frustration for the affected users. Operational experience of eduroam shows that such cases are hard to debug since they require coordinated cooperation of all eduroam administrators on the authentication path. For that reason the eduroam community is developing monitoring tools that help to locate fragmentation problems.

Note: For more detailed discussion of these issues please refer to section 1.1 of [RFC6613].

3.5. Insufficient payload encryption and EAP server validation

The RADIUS protocol's design foresaw only the encryption of select RADIUS attributes, most notably User-Password. With EAP methods conforming to the requirements of [RFC4017], the user's credential is not transmitted using the User-Password attribute, and stronger encryption than the one for RADIUS' User-Password is in use (typically TLS).

Still, the use of EAP does not encrypt all personally identifiable details of the user session as some are carried inside clear-text RADIUS attributes. In particular, the user's device can be identified by inspecting the Calling-Station-ID attribute; and the user's location may be derived from observing NAS-IP-Address, NAS-Identifier or Operator-Name attributes. Since these attributes are not encrypted, even IP-layer third parties can harvest the corresponding data. In a worst-case scenario, this enables the creation of mobility profiles. Pervasive passive surveillance using this connection metadata such as the recently uncovered NSA/GCHQ incidents becomes possible by tapping RADIUS traffic from an IP hop near a RADIUS aggregation proxy. While this is possible, the authors are not aware whether this has actually been done.

These profiles are not necessarily linkable to an actual user because EAP allows for the use of anonymous outer identities and protected credential exchanges. However, practical experience has shown that many users neglect to configure their supplicants in a privacy-preserving way or their supplicant doesn't support that. In particular, for EAP-TLS users, the use of EAP-TLS identity protection is not usually implemented and cannot be used. In eduroam, concerned individuals and IdPs which use EAP-TLS are using pseudonymous client certificates to provide for better privacy.

One way out, at least for EAP types involving a username, is to pursue the creation and deployment of pre-configured supplicant configurations which makes all the required settings in user devices prior to their first connection attempt; this depends heavily on the remote configuration possibilities of the supplicants though.

A further threat involves the verification of the EAP server's identity. Even though the cryptographic foundation, TLS tunnels, is sound, there is a weakness in the supplicant configuration: many users do not understand or are willing to invest time into the inspection of server certificates or the installation of a trusted CA. As a result, users may easily be tricked into connecting to an unauthorized EAP server, ultimately leading to a leak of their credentials to that unauthorized third party.

Again, one way out of this particular threat is to pursue the creation and deployment of pre-configured supplicant configurations which makes all the required settings in user devices prior to their first connection attempt.

Note: there are many different and vendor-proprietary ways to pre-configure a device with the necessary EAP parameters (examples include Apple, Inc's "mobileconfig" and Microsoft's "EAPHost" XML schema). Some manufacturers even completely lack any means to distribute EAP configuration data. We believe there is value in defining a common EAP configuration metadata format which could be used across manufacturers, ideally leading to a situation where IEEE 802.1X network end-users merely need to apply this configuration file to configure any of their devices securely with the required connection properties.

Another possible privacy threat involves transport of user-specific attributes in a Reply-Message. If, for example, a RADIUS server sends back a hypothetical RADIUS Vendor-Specific-Attribute "User-Role = Student of Computer Science" (e.g. for consumption of an SP RADIUS server and subsequent assignment into a "student" VLAN), this information would also be visible for third parties and could be added to the mobility profile.

The only way out to mitigate all information leakage to third parties is by protecting the entire RADIUS packet payload so that IP-layer third parties cannot extract privacy-relevant information. RFC2865 RADIUS does not offer this possibility though. This motivated [RFC6614], see below.

4. New Trust Fabric

The operational difficulties with an ever increasing number of participants, as documented in the previous section, have led to a number of changes to the eduroam architecture that in turn have, as mentioned in the introduction, led to standardization effort.

Note: The enhanced architecture components are fully backwards compatible with the existing installed base, and are in fact gradually replacing those parts of it where problems may arise.

Whereas the user authentication using IEEE 802.1X and EAP has remained unchanged (i.e. no need for end-users to change any configurations), the issues as reported above have resulted in a major overhaul of the way EAP messages are transported from the RADIUS server of the SP to that of the IdP and back. The two fundamental changes are the use of TCP instead of UDP and reliance on TLS instead of shared secrets between RADIUS peers.

4.1. RADIUS with TLS

The deficiencies of RADIUS over UDP as described in Section 3.4 warranted a search for a replacement of RFC2865 [RFC2865] for the transport of EAP. By the time this need was understood, the designated successor protocol to RADIUS, Diameter [RFC3588], was already specified by the IETF. However, within the operational constraints of eduroam:

- o reasonably cheap to deploy on many administrative domains
- o supporting NASREQ Application
- o supporting EAP Application
- o supporting Diameter Redirect
- o supporting validation of authentication requests of the most popular EAP types (EAP-TTLS, PEAP, and EAP-TLS)
- o possibility to retrieve these credentials from popular backends such as Microsoft ActiveDirectory, MySQL

no single combination of software could be found. In addition to that, no Wireless Access Points at the disposal of eduroam participants supported Diameter, nor did any of the manufacturers have a roadmap towards Diameter support (that is believed to still be true, more than 10 years later). This led to the open question of lossless translation from RADIUS to Diameter and vice versa; a question not satisfactorily answered by NASREQ.

After monitoring the Diameter implementation landscape for a while, it became clear that a solution with better compatibility and a plausible upgrade path from the existing RADIUS hierarchy was needed. The eduroam community actively engaged in the IETF towards the specification of several enhancements to RADIUS to overcome the limitations mentioned in Section 3. The outcome of this process was [RFC6614] and [I-D.ietf-radext-dynamic-discovery].

With its use of TCP instead of UDP, and with its full packet encryption, while maintaining full packet format compatibility with RADIUS/UDP, RADIUS/TLS [RFC6614] allows to upgrade any given RADIUS link in eduroam without the need of a "flag day".

In a first upgrade phase, the classic eduroam hierarchy (forwarding decision taken by inspecting the realm) remains intact. That way, RADIUS/TLS merely enhances the underlying transport of the RADIUS datagrams. But this already provides some key advantages:

- o explicit peer reachability detection using long-lived TCP sessions
- o protection of user credentials and all privacy-relevant RADIUS attributes

RADIUS/TLS connections for the static hierarchy could be realised with the TLS-PSK operation mode (which effectively provides a 1:1 replacement for RADIUS/UDP's "shared secrets"), but since this operation mode is not widely supported as of yet, all RADIUS/TLS links in eduroam are secured by TLS with X.509 certificates from a set of accredited CAs.

This first deployment phase does not yet solve the routing table complexity problem (see (Section 3.3); this aspect is covered by introducing dynamic discovery for RADIUS/TLS servers.

4.2. Dynamic Discovery

When introducing peer discovery, two separate issues had to be addressed:

1. How to find the network address of a responsible RADIUS server for a given realm?
2. How to verify that this realm is an authorized eduroam participant?

4.2.1. Discovery of responsible server

Issue 1 can relatively simply be addressed by putting eduroam-specific service discovery information into the global DNS tree. In eduroam this is done by using Naming Authority Pointer (NAPTR) records as per the S-NAPTR specification [RFC3958] with a private-use NAPTR service tag ("x-eduroam:radius.tls"). The usage profile of that NAPTR resource record is that exclusively "S" type delegations are allowed, and that no regular expressions are allowed.

A subsequent lookup of the resulting SRV records will eventually yield hostnames and IP addresses of the authoritative server(s) of a given realm.

Example (wrapped for readability):

```
> dig -t naptr education.example.

;; ANSWER SECTION:
education.example.      43200   IN      NAPTR   100 10 "s"
                        "x-eduroam:radius.tls" ""
                        _radsec._tcp.eduroam.example.

> dig -t srv _radsec._tcp.eduroam.example.

;; ANSWER SECTION:
_radsec._tcp.eduroam.example. 43200   IN      SRV      0 0 2083
                        tld1.eduroam.example.

> dig -t aaaa tld1.eduroam.example.

;; ANSWER SECTION:
tld1.eduroam.example.    21751   IN      AAAA     2001:db8:1::2
```

Figure 3: SRV record lookup

From the operational experience with this mode of operation, eduroam is pursuing standardisation of this approach for generic AAA use cases. The current radext working group document for this is [I-D.ietf-radext-dynamic-discovery].

Note: It is worth mentioning that this move to a more complex, flexible system may make the system as a whole more fragile, as compared to the static set up.

4.2.2. Verifying server authorisation

Any organisation can put "x-eduroam" NAPTR entries into their Domain Name Server, pretending to be eduroam Identity Provider for the corresponding realm. Since eduroam is a service for a heterogeneous, but closed, user group, additional sources of information need to be consulted to verify that a realm with its discovered server is actually an eduroam participant.

The eduroam consortium has chosen to deploy a separate PKI infrastructure which issues certificates only to authorised eduroam Identity Providers and eduroam Service Providers. Since certificates are needed for RADIUS/TLS anyway, it was a straightforward solution to reuse the PKI for that. The PKI fabric allows multiple CAs as trust roots (overseen by a Policy Management Authority), and requires that certificates which were issued to verified eduroam participants are marked with corresponding "X509v3 Policy OID" fields; eduroam

RADIUS servers and clients need to verify the existence of these OIDs in the incoming certificates.

The policies and OIDs can be retrieved from the "eduPKI Trust Profile for eduroam Certificates" ([edupki]).

4.2.3. Operational Experience

The discovery model as described above is currently deployed in approximately 10 countries that participate in eduroam, making more than 100 realms discoverable via their NAPTR records. Experience has shown that the model works and scales as expected; the only drawback being that the additional burden of operating a PKI which is not local to the national eduroam administrators creates significant administrative complexities. Also, the presence of multiple CAs and regular updates of Certificate Revocation Lists makes the operation of RADIUS servers more complex.

4.2.4. Possible Alternatives

There are two alternatives to the above approach which are monitored by the eduroam community:

1. DNSSEC + DANE TLSA records
2. ABFAB Trust Router

For DNSSEC+DANE TLSA, its biggest advantage is that the certificate data itself can be stored in the DNS - possibly obsoleting the PKI infrastructure *if* a new place for the server authorization checks can be found. Its most significant downside is that the DANE specifications only include client-to-server certificate checks, while RADIUS/TLS requires also server-to-client verification.

For the ABFAB Trust Router, the biggest advantage is that it would work without certificates altogether (by negotiating TLS-PSK keys ad-hoc). The downside is that it is currently not formally specified and not as thoroughly understood as any of the other solutions.

5. Abuse prevention and incident handling

Since the eduroam service is a confederation of autonomous networks, there is little justification for transferring accounting information from the Service Provider to any other in general, or in particular to the Identity Provider of the user. Accounting in eduroam is therefore considered to be a local matter of the Service Provider. The eduroam compliance statement ([eduroam-compliance]) in fact specifies that accounting traffic SHOULD NOT be forwarded.

The static routing infrastructure of eduroam acts as a filtering system blocking accounting traffic from misconfigured local RADIUS servers. Proxy servers are configured to terminate accounting request traffic by answering to Accounting-Requests with an Accounting-Response in order to prevent the retransmission of orphaned Accounting-Request messages. With dynamic discovery, Identity Providers which are discoverable via DNS will need to apply these filtering measures themselves. This is an increase in complexity of the Identity Provider RADIUS configuration.

Roaming creates accountability problems, as identified by [RFC4372] (Chargeable User Identity). Since the NAS can only see the (likely anonymous) outer identity of the user, it is impossible to correlate usage with a specific user (who may use multiple devices). A NAS that supports this can request the Chargeable-User-Identity and, if supplied by the authenticating RADIUS server in the Access-Accept message, add this value to corresponding Access-Request packets. While eduroam does not have any charging mechanisms, it may still be desirable to identify traffic originating from one particular user. One of the reasons is to prevent abuse of guest access by users living nearby university campuses. Chargeable User Identity (see below) supplies the perfect answer to this problem, however at the moment of writing, to our knowledge only one hardware vendor (Meru Networks) implements RFC4372 on their Access Points. For all other vendors, requesting the Chargeable-User-Identity attribute needs to happen on the RADIUS server to which the Access Point is connected to. FreeRADIUS supports this ability in the latest distribution, and Radiator can be retrofitted to do the same.

5.1. Incident Handling

10 years of experience with eduroam have not exposed any serious incident. This may be taken as evidence for proper security design as well as suggest that awareness of users that they are identifiable, acts as an effective deterrent. It could of course also mean that eduroam operations lack the proper tools or insight into the actual use and potential abuse of the service. In any case, many of the attack vectors that exist in open networks or networks where access control is based on shared secrets are not present, arguably leading to a much more secure system.

Below a discussion of countermeasures that are taken in eduroam.

The European eduroam policy Service Definition [eduroam-service-definition], as an example, describes incident scenarios and actions to be taken, in this document we present the relevant technical issues.

The initial implementation has been lacking reliable tools for an SP to make it's own decision or for an IdP to introduce a conditional rule applying only to a given SP. The introduction of support for Operator-Name and Chargeable-User-Identity (see below) to eduroam makes both of these scenarios possible.

5.1.1. Blocking users on the SP side

The first action in the case of an incident is to block the user's access to eduroam at the Service Provider. Since the roaming user's true identity is likely hidden behind an anonymous/fake outer identity, the Service Provider can only rely on the realm of the user and his MAC address; if the Identity Provider has already sent a Chargeable-User-Identity (see Section 5.3 for details), then this extra information can be used to identify the user more reliably.

A first attempt at the SP side may be to block based on the MAC address or outer identity. This blocking can be executed before the EAP authentication occurs - typically in the first datagram, acting on the RADIUS attributes EAP-Message/EAP-Response/Identity and Calling-Station-ID. The datagram can either be dropped (supplicant notices a time-out) or replied-to with a RADIUS Access-Reject containing an EAP-Failure. Since malicious users can change both their MAC addresses and the local part of their outer identity between connection attempts, this first attempt is not sufficient to lock out a determined user.

As a second measure, the SP can let the EAP authentication proceed as normal, and verify whether the final Access-Accept response from the RADIUS server contains a Chargeable-User-Identity (CUI). If so, the SP RADIUS server can be configured to turn all future Access- Accepts for this CUI into an Access-Reject/EAP-Failure. This measure is effective and efficient: it locks out exactly the one user which is supposed to be locked out, and has no side-effects on other users, even from the same realm.

If the EAP authentication does not reveal a CUI, the SP can not reliably determine the user in question. The only reliable information to act upon is then the realm portion of the outer identity of the user. The SP will need to resort to blocking the entire realm that the offending user belongs to. This can be done at the EAP-Message/EAP-Response/Identity stage as described above). This is effective, but not efficient: it locks out the user in question, but has a DoS side-effect on all other visiting users from the same realm.

In the absence of a CUI handle, SPs which are not willing to take the drastic step of blocking an entire realm will be forced to contact

the Identity Provider in question and demand that the user be blocked at the IdP side. This involves human interaction between SP and IdP is not possible in real-time.

5.1.2. Blocking users on the IdP side

The IdP has the power to disable a user account altogether, thus blocking this user from accessing eduroam in all sites. If blocking the user is done due a request of an SP (as per the previous section), there may be a more fine-grained possibility to block access to a specific SP - if the SP in question sends the Operator-Name attribute along with his Access-Requests (see Section 5.2 for details).

If the IdP decides to block the user globally, this is typically done by treating the login attempt as if the credentials were wrong: the entire EAP conversation needs to be executed to the point where the true inner identity is revealed (before that, the IdP does not know yet which user is attempting to authenticate). This typically coincides with the point in time where credentials are exchanged. Instead, or in addition to, checking the credential for validity, the Identity Provider also checks whether the user's account is (still) eligible for eduroam use and will return an Access- Reject/EAP-Failure if not.

There may well be cases where opinions between the SP desiring a user lockout and the IdP of the user differ. E.g. an SP might consider massive amounts of up-/downloads with file sharing protocols unacceptable as per local policy, and desire blocking of users that create too much traffic - but the IdP does not take offense on such actions and would not want to lock his user out of eduroam globally because of this one SP's opinion.

In the absence of the Operator-Name attribute, there is no way to apply a login restriction only for a given SP and not eduroam as a whole; eduroam eligibility is an all-or-nothing decision for the IdP.

If the Operator-Name attribute is present, the IdP can use this information to fail the authentication attempt only if the attempt originated from SPs which desire such blocking. Even though the Operator-Name attribute is available from the first RADIUS Access-Request datagram onwards, the EAP authentication needs to be carried out until the true inner identity is known just as in the global blocking case above. The Operator-Name is simply an additional piece of information which the IdP can use to make its decision.

5.1.3. Communicating account blocking to the end user

All the measures above alter the EAP conversation. They either create a premature rejection or timeout at the start of the conversation, or change the outcome of the authentication attempt at the very end of the conversation.

On the supplicant side, these alterations are undistinguishable from an infrastructure failure: a premature rejection or timeout could be due to a RADIUS server being unresponsive, and a rejection at the end of the conversation could be either user error (wrong password) or server error (credential lookup failed). For the supplicant, it is thus difficult to communicate a meaningful error to the user. The newly specified EAP type TEAP, "Tunnel Extensible Authentication Protocol" [RFC7170] has a means to transport fine-grained error reason codes to the supplicant; this has the potential to improve the situation in the future.

The EAP protocol foresees one mechanism to provide such user-interactive communication: the EAP state machine contains states which allow user-visible communication: an extra round of EAP-Request/Notification and the corresponding acknowledgement can be injected before the final EAP-Failure.

However, anecdotal evidence suggests that supplicants typically do not implement this part of the EAP state machine at all. One supplicant is reported to support it, but only logs the contents of the notification in a log file - which is not at all helpful for the end user.

The discovery of reasons and scope of account blocking are thus left as an out-of-band action. The eduroam user support process requires that users with authentication problems contact their Identity Provider as a first measure (via unspecified means, e.g. they could phone their IdP or send an email via a 3G backup link). If the Identity Provider is the one which blocked their access, the user will immediately be informed by them. If the reason for blocking is at the SP side, the Identity Provider will instead inform the user that the account is in working order and that the user needs to contact the SP IT support to get further insight. In that case, that SP-side IT support will notify the users about the reasons for blocking.

5.2. Operator Name

The Operator-Name attribute is defined in [RFC5580] as a means of unique identification of the access site.

The Proxy infrastructure of eduroam makes it impossible for home sites to tell where their users roam to. While this may be seen as a positive aspect enhancing user's privacy, it also makes user support, roaming statistics and blocking offending user's access to eduroam significantly harder.

Sites participating in eduroam are encouraged to add the Operator-Name attribute using the REALM namespace, i.e. sending a realm name under control of the given site.

The introduction of Operator-Name in eduroam has identified one operational problem - the identifier 126 assigned to this attribute has been previously used by some vendors for their specific purposes and has been included in attribute dictionaries of several RADIUS server distributions. Since the syntax of this hijacked attribute had been set to Integer, this introduces a syntax clash with the the RFC definition (Text). Operational tests in eduroam have shown that servers using the Integer syntax for attribute 126 may either truncate the value to 4 octets or even drop the entire RADIUS packet (thus making authentication impossible). The eduroam monitoring and eduroam test tools try to locate problematic sites. [RFC6929] clarifies in Section 2.8 the handling of these packets.

When a Service Provider sends its Operator-Name value, it creates a possibility for the home sites to set up conditional blocking rules, depriving certain users of access to selected sites. Such action will cause much less concern than blocking users from all of eduroam.

In eduroam the Operator Name is also used for the generation of Chargeable User Identity values.

The addition of Operator-Name is a straightforward configuration of the RADIUS server and may be easily introduced on a large scale.

5.3. Chargeable User Identity

The Chargeable-User-Identity (CUI) attribute is defined by RFC4372 [RFC4372] as an answer to accounting problems caused by the use of anonymous identity in some EAP methods. In eduroam the primary use of CUI is in incident handling, but it can also enhance local accounting.

The eduroam policy requires that a given user's CUI generated for requests originating from different sites should be different (to prevent collusion attacks). The eduroam policy thus mandates that a CUI request be accompanied by the Operator-Name attribute, which is used as one of the inputs for the CUI generation algorithm. The Operator-Name requirement is considered to be the "business

requirement" described in Section 2.1 of RFC4372 [RFC4372] and hence conforms to the RFC.

When eduroam started considering using CUI, there were no NAS implementations, therefore the only solution was moving all CUI support to the RADIUS server.

CUI request generation requires only the addition of NUL CUI attributes to outgoing Access-Requests, however the real strength of CUI comes with accounting. Implementation of CUI based accounting in the server requires that the authentication and accounting RADIUS servers used directly by the NAS are actually the same or at least have access to a common source of information. Upon processing of an Access-Accept the authenticating RADIUS server must store the received CUI value together with the device's Calling-Station-Id in a temporary database. Upon receipt of an Accounting-Request, the server needs to update the packet with the CUI value read from the database.

A wide introduction of CUI support in eduroam will significantly simplify incident handling at Service Providers. Introducing local, per-user access restriction will be possible. Visited sites will also be able to notify the home site about the introduction of such a restriction, pointing to the CUI value and thus making it possible for the home site to identify the user. When the user reports the problem at his home support, the reason will be already known.

6. Privacy Considerations

The eduroam architecture has been designed with protection of user credentials in mind as may be clear from the discussion above. However, operational experience has revealed some more subtle points with regards to privacy.

6.1. Collusion of Service Providers

If users use anonymous outer identities, SPs cannot easily collude by linking outer identities to users that are visiting their campus. This poses however problems with remediation of abuse or misconfiguration. It is impossible to find the user that exhibits unwanted behaviour or whose system has been compromised.

For that reason the Chargeable-User-Identity has been introduced in eduroam, constructed so that only the IdP of the user can uniquely identify the user. In order to prevent collusion attacks that CUI is required to be unique per user per Service Provider.

6.2. Exposing user credentials

Through the use of EAP, user credentials are not visible to anyone but the IdP of the user. That is, if a sufficiently secure EAP-method is chosen and EAP is not terminated prematurely.

There is one privacy sensitive user attribute that is necessarily exposed to third parties and that is the realm the user belongs to. Routing in eduroam is based on the realm part of the user identifier, so even though the outer identity in a tunneled EAP-method may be set to an anonymous identifier it MUST contain the realm of the user, and may thus lead to identifying the user if the realm in question contains few users. This is considered a reasonable trade-off between user privacy and usability.

6.3. Track location of users

Due to the fact that access requests (potentially) travel through a number of proxy RADIUS servers, the home IdP of the user typically cannot tell where a user roams to.

The introduction of Operator-Name and dynamic lookups (i.e. direct connections between IdP and SP) however, give the home IdP insight into the location of the user.

7. Security Considerations

This section addresses only security considerations associated with the use of eduroam. For considerations relating to IEEE 802.1X, RADIUS and EAP in general, the reader is referred to the respective specification and to other literature.

7.1. Man in the middle and Tunneling Attacks

The security of user credentials in eduroam ultimately lies within the EAP server verification during the EAP conversation. Therefore, the eduroam policy mandates that only EAP types capable of mutual authentication are allowed in the infrastructure, and requires that IdPs publish all information that is required to uniquely identify the server (i.e. usually the EAP server's CA certificate and its Common Name or subjectAltName:dNSName).

While this in principle makes Man-in-the-middle attacks impossible, practice has shown that several attack vectors exist nonetheless. Most of these deficiencies are due to implementation shortcomings in EAP supplicants. Examples:

7.1.1.1. Verification of Server Name not supported

Some supplicants only allow to specify which CA issues the EAP server certificate; it's name is not checked. As a result, any entity that is able to get a server certificate from the same CA can create its own EAP server and trick the end user to submit his credentials to that fake server.

As a mitigation to that problem, eduroam Operations suggests the use of a private CA which exclusively issues certificates to the organisation's EAP servers. In that case, no other entity will get a certificate from the CA and the above supplicant shortcoming does not present a security threat any more.

7.1.1.2. Neither Specification of CA nor Server Name checks during bootstrap

Some supplicants allow for insecure bootstrapping in that they allow to simply select a network and accept the incoming server certificate, identified by its fingerprint. The certificate is then saved as trusted for later re-connection attempts. If users are near a fake hotspot during initial provisioning, they may be tricked to submit their credentials to a fake server; and furthermore will be branded to trust only that fake server in the future.

eduroam Identity Providers are advised to provide their users with complete documentation for setup of their supplicants without the shortcut of insecure bootstrapping. In addition, eduroam Operations has created a tool which makes correct, complete and secure settings on many supplicants: eduroam CAT ([eduroam-cat]).

7.1.1.3. User does not configure CA or Server Name checks

Unless automatic provisioning tools such as eduroam CAT are used, it is cumbersome for users to initially configure an EAP supplicant securely. User Interfaces of supplicants often invite the users to take shortcuts ("Don't check server certificate") which are easier to setup or hide important security settings in badly accessible sub-menus. Such shortcuts or security parameter omissions make the user subject to man-in-the-middle attacks.

eduroam IdPs are advised to educate their users regarding the necessary steps towards a secure setup. eduroam Research and Development is in touch with supplicant developers to improve their User Interfaces.

7.1.4. Tunneling authentication traffic to obfuscate user origin

There is no link between the EAP outer ("anonymous") identity and the EAP inner ("real") identity. In particular, they can both contain a realm name, and the realms need not be identical. It is possible to craft packets with an outer identity of user@RealmB, and an inner identity of user@realmA. With the eduroam request routing, a Service Provider would assume that the user is from realmB and send the request there. The server at realm B inspects the inner user name, and if proxying is not explicitly disabled for tunneled request content, may decide to send the tunneled EAP payload to realmA, where the user authenticates. A CUI value would likely be generated by the server at realmB, even though this is not its user.

Users can craft such packets to make their identification harder; usually, the eduroam SP would assume the troublesome user to originate from realmB and demand there that the user be blocked. The operator of realmB however has no control over the user, and can only trace back the user to his real origin if logging of proxied requests is also enabled for EAP tunnel data.

eduroam Identity Providers are advised to explicitly disable proxying on the parts of their RADIUS server configuration which processes EAP tunnel data.

7.2. Denial of Service Attacks

Since eduroam's roaming infrastructure is based on IP and RADIUS, it suffers from the usual DoS attack vectors that apply to these protocols.

The eduroam hotspots are susceptible to typical attacks on consumer edge networks, such as rogue RA, rogue DHCP servers, and others. Notably, eduroam hotspots are more robust against malicious users' DHCP pool exhaustion than typical open or "captive portal" hotspots, because a DHCP address is only leased after a successful authentication, which reduces the pool of possible attackers to eduroam account holders (as opposed to the general public). Furthermore, attacks involving ARP spoofing or ARP flooding are also reduced to authenticated users, because an attacker needs to be in possession of a valid WPA2 session key to be able to send traffic on the network.

This section does not discuss standard threats to consumer edge networks and IP networks in general. The following sections describe attack vectors specific to eduroam.

7.2.1. Intentional DoS by malign individuals

The eduroam infrastructure is more robust against Distributed DoS attacks than typical services which are reachable on the internet because triggering authentication traffic can only be done when physically being in proximity of an eduroam hotspot (be it a wired IEEE 802.1X enabled socket or a Wi-Fi Access Point).

However, when being in the vicinity, it is easy to craft authentication attempts that traverse the entire international eduroam infrastructure; an attacker merely needs to choose a realm from another world region than his physical location to trigger Access-Requests which need to be processed by the SP, then SP-side national, then world region, then target world region, then target national, then target IdP server. So long as the realm actually exists, this will be followed by an entire EAP conversation on that path. Not having actual credentials, the request will ultimately be rejected; but it consumed processing power and bandwidth across the entire infrastructure, possibly affecting all international authentication traffic.

EAP is a lock-step protocol. A single attacker at an eduroam hotspot can only execute one EAP conversation after another, and is thus rate-limited by round-trip times of the RADIUS chain.

Currently eduroam processes several hundred thousands of successful international roaming authentications per day (and, incidentally, approximately 1.5 times as many Access-Rejects). With the requirement of physical proximity, and the rate-limiting induced by EAP's lock-step nature, it requires a significant amount of attackers and a time-coordinated attack to produce significant load. So far eduroam Operations has not yet observed critical load conditions which could reasonably be attributed to such an attack.

The introduction of dynamic discovery further eases this problem, as authentications will then not traverse all infrastructure servers, removing the world-region aggregation servers as obvious bottlenecks. Any attack would then be limited between an SP and IdP directly.

7.2.2. DoS as a side-effect of expired credentials

In eduroam Operations it is observed that a significant portion of (failed) eduroam authentications is due to user accounts which were once valid, but have in the meantime been de-provisioned (e.g. if a student has left the university after graduation). Configured eduroam accounts are often retained on the user devices, and when in the vicinity of an eduroam hotspot, the user device's operating system will attempt to connect to this network.

As operation of eduroam continues, the amount of devices with left-over configurations is growing, effectively creating a pool of devices which produce unwanted network traffic whenever they can.

Up until recently, this problem did not emerge with much prominence, because there is also a natural shrinking of that pool of devices due to users finally de-commissioning their old computing hardware.

As of recent, particularly smartphones are programmed to make use of cloud storage and online backup mechanisms which save most, or all, configuration details of the device with a third-party. When renewing their personal computing hardware, users can restore the old settings onto the new device. It has been observed that expired eduroam accounts can survive perpetually on user devices that way. If this trend continues, it can be pictured that an always-growing pool of devices will clog up eduroam infrastructure with doomed-to-fail authentication requests.

There is not currently a useful remedy to this problem, other than instructing users to manually delete their configuration in due time. Possible approaches to this problem are:

- o Creating a culture of device provisioning where the provisioning profile contains a "ValidUntil" property, after which the configuration needs to be re-validated or disabled. This requires a data format for provisioning as well as implementation support.
- o Improvements to supplicant software so that it maintains state over failed authentications. E.g. if a previously known-working configuration failed to authenticate consistently for 30 calendar days, it should be considered stale and be disabled.

8. IANA Considerations

There are no IANA Considerations

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4372] Adrangi, F., Lior, A., Korhonen, J., and J. Loughney, "Chargeable User Identity", RFC 4372, January 2006.
- [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, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5580] Tschofenig, H., Adrangi, F., Jones, M., Lior, A., and B. Aboba, "Carrying Location Objects in RADIUS and Diameter", RFC 5580, August 2009.
- [RFC5997] DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", RFC 5997, August 2010.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.
- [RFC6613] DeKok, A., "RADIUS over TCP", RFC 6613, May 2012.
- [RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.

- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, July 2013.

9.2. Informative References

- [I-D.ietf-abfab-arch]
Howlett, J., Hartman, S., Tschofenig, H., Lear, E., and J. Schaad, "Application Bridging for Federated Access Beyond Web (ABFAB) Architecture", draft-ietf-abfab-arch-13 (work in progress), July 2014.
- [I-D.ietf-radext-dtls]
DeKok, A., "DTLS as a Transport Layer for RADIUS", draft-ietf-radext-dtls-13 (work in progress), July 2014.
- [I-D.ietf-radext-dynamic-discovery]
Winter, S. and M. McCauley, "NAI-based Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS", draft-ietf-radext-dynamic-discovery-13 (work in progress), March 2015.
- [I-D.ietf-radext-nai]
DeKok, A., "The Network Access Identifier", draft-ietf-radext-nai-15 (work in progress), December 2014.
- [MD5-attacks]
Black, J., Cochran, M., and T. Highland, "A Study of the MD5 Attacks: Insights and Improvements", October 2006, <<http://www.springerlink.com/content/40867185727r7084/>>.
- [RFC3539] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, June 2003.
- [RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.
- [RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.
- [RFC4017] Stanley, D., Walker, J., and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs", RFC 4017, March 2005.
- [RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.

- [RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.
- [RFC6421] Nelson, D., "Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS)", RFC 6421, November 2011.
- [RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, April 2013.
- [RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, May 2014.
- [dead-realm] Tomasek, J., "Dead-realm marking feature for Radiator RADIUS servers", 2006, <<http://wiki.eduroam.cz/dead-realm/docs/dead-realm.html>>.
- [dot1X-standard] IEEE, "IEEE std 802.1X-2010", February 2010, <<http://standards.ieee.org/getieee802/download/802.1X-2010.pdf>>.
- [edupki] Delivery of Advanced Network Technology to Europe, "eduPKI", 2012, <<https://www.edupki.org/edupki-pma/edupki-trust-profiles/>>.
- [eduroam-cat] Delivery of Advanced Network Technology to Europe, "eduroam CAT", 2012, <<https://cat.eduroam.org>>.
- [eduroam-compliance] Trans-European Research and Education Networking Association, "eduroam compliance statement", 2011, <http://www.eduroam.org/downloads/docs/eduroam_Compliance_Statement_v1_0.pdf>.

- [eduroam-homepage]
Trans-European Research and Education Networking Association, "eduroam Homepage", 2007,
<<http://www.eduroam.org/>>.
- [eduroam-policy]
Delivery of Advanced Network Technology to Europe, "European Confederation eduroam policy", 2011,
<http://www.eduroam.org/downloads/docs/GN3-12-194_eduroam-policy-%20for-signing_ver2%204_18052012.pdf>.
- [eduroam-service-definition]
Delivery of Advanced Network Technology to Europe, "European eduroam policy Service Definition", 2011,
<https://www.eduroam.org/downloads/docs/GN3-12-192_eduroam-policy-service-definition_ver28_26072012.pdf>.
- [eduroam-start]
Wierenga, K., "Initial proposal for what is now called eduroam", 2002, <<http://www.terena.org/activities/tf-mobility/start-of-eduroam.pdf>>.
- [geant]
Geant Association, "Geant Association", 2008,
<<http://www.terena.org/>>.
- [geant2]
Delivery of Advanced Network Technology to Europe, "European Commission Information Society and Media: GEANT2", 2008, <<http://www.geant2.net/>>.
- [nrenroaming-select]
Trans-European Research and Education Networking Association, "Preliminary selection for inter-NREN roaming", 2003, <<http://www.terena.org/activities/tf-mobility/deliverables/delG/DelG-final.pdf>>.
- [radsec-whitepaper]
Open System Consultants, "RadSec - a secure, reliable RADIUS Protocol", May 2005,
<<http://www.open.com.au/radiator/radsec-whitepaper.pdf>>.
- [radsecproxy-impl]
Venaas, S., "radsecproxy Project Homepage", 2007,
<<http://software.uninett.no/radsecproxy/>>.

Appendix A. Acknowledgments

The authors would like to thank the participants in the Geant Association Task Force on Mobility and Network Middleware as well as the Geant project for their reviews and contributions. Special thanks go to Jim Schaad for doing an excellent review of the first version and to him and Alan de Kok for additional reviews.

The eduroam trademark is registered by TERENA.

Appendix B. Changes

This section to be removed prior to publication.

- o 00 Initial Revision
- o 01 Added Dynamic Discovery, addressed review comments
- o 02 Cosmetic changes
- o 03 Even More Cosmetic Changes
- o 04 Included review comments from Jim Schaad
- o 05 Included review comments Jim Schaad and Alan deKok

Authors' Addresses

Klaas Wierenga
Cisco Systems
Haarlerbergweg 13-19
Amsterdam 1101 CH
The Netherlands

Phone: +31 20 357 1752
Email: klaas@cisco.com

Stefan Winter
Fondation RESTENA
6, rue Richard Coudenhove-Kalergi
Luxembourg 1359
Luxembourg

Phone: +352 424409 1
Fax: +352 422473
Email: stefan.winter@restena.lu
URI: <http://www.restena.lu>

Tomasz Wolniewicz
Nicolaus Copernicus University
pl. Rapackiego 1
Torun
Poland

Phone: +48-56-611-2750
Fax: +48-56-622-1850
Email: twoln@umk.pl
URI: <http://www.home.umk.pl/~twoln/>