Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 9, 2011

D. Cheng
Huawei Technologies

May 8, 2011

Radius Extensions for CGN Configurations
draft-cheng-behave-cgn-cfg-radius-ext-00

Abstract

This document proposes three new RADIUS attributes that can be used by a CGN device to communicate with a RADIUS server using RADIUS protocol [RFC2865] to configure or report TCP/UDP ports and ICMP identifiers mapping behavior for specific Internet subscribers.

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].

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 9, 2011.

Copyright Notice

Table of Contents

1. Introduction

In a broadband network, customer information is usually stored on a RADIUS server and at the time when a user initiates an Internet connection request, the RADIUS server will populate the user's configuration information to the NAS, which is usually co-located with the BNG, after the connection request is granted. In many cases, the CGN function is also implemented on the BNG, and therefore CGN TCP/UDP port (or ICMP identifier) mapping behavior can be configured on the RADIUS server as part of the user profile, and populated to the NAS in the same manner. In addition, during the operation, the CGN can also convey port/identifier mapping behavior specific to a user to the RADIUS server, as part of the normal RADIUS accounting process.

The CGN device that communicates with a RADIUS server using RADIUS extensions proposed in this document may perform NAT44 [I-D.ietf-behave-lsn-requirements], NAT64 [RFC6146], or Dual-Stack Lite [I-D.ietf-softwire-dual-stack-lite] function.

When IP packets traverse a CGN device, it would perform TCP/UDP source port mapping or ICMP identifier mapping as required. A TCP/UDP source port or ICMP identifier, along with source IP address, destination IP address, destination port and protocol identifier if applicable, uniquely identify an IP connection or session. Since the number space of TCP/UDP ports and ICMP identifiers in CGN's external realm is shared by multiple users sharing the same IPv4 address, the total number of a user's live IP sessions is usually limited.

In order to support the communication between a BNG-based CGN and RADIUS server as described above, this document proposes three new RADIUS attributes as RADIUS protocol's extensions, and they are used for separate purposes as follows:

- A session limit is configured on a RADIUS server based on service agreement with a specific user, and this parameter imposes the limit of total number of TCP/UDP ports plus ICMP identifiers that the user can use behind the CGN. Alternately, a separate session limit may be configured to limit the number of TCP ports, UDP ports, or the sum of the two, and ICMP identifiers, respectively, that the user can use. The session limit is carried by a new RADIUS attribute CGN-Session-Limit, which is included in a RADIUS Access-Accept message sent by the RADIUS server to the BNG based CGN. This new RADIUS attribute can also be included in a RADIUS CoA message sent by the RADIUS server to the BNG based CGN in order to change the session limit previously configured.

- A CGN may allocate or de-allocate some consecutive TCP/UDP ports or ICMP identifiers for a specific subscriber. When it does so, the associated session range along with the shared IPv4 address can be conveyed to the RADIUS server as part of the accounting process. These parameters are carried by a new RADIUS attribute CGN-Session-Range, which is included in a RADIUS Accounting-Request message sent by the BNG based CGN to the RADIUS server.

- A user may require the CGN device to perform port forwarding function, i.e., a port mapping is pre-configured on the CGN so that inbound IP packets sent by some applications from CGN external realm can pass through the CGN and reach the user. The port mapping information includes the CGN internal port, external port, and may also include the associated internal IPv4 or IPv6 address, and is carried by a new RADIUS attribute CGN-Port-Forwarding-Map, which is included in a RADIUS Access-Accept message sent by the RADIUS server to the BNG based CGN. This new RADIUS attribute can also be included in a RADIUS CoA message sent by the RADIUS server to the BNG based CGN in order to change the forwarding port mapping previously configured.

Note - This document is a merged version of draft-cheng-behave-nat44-pre-allocated-ports-02.txt and draft-cheng-behave-nat-fwd-port-radius-ext-00.txt with some updates.

2. Terminologies

Some terminologies that are used in this document are listed as follows:

- Session Limit - This is the maximum number of TCP ports, or UDP ports, or the total of the two, or ICMP identifiers, or the total of the three, that a CGN can use when performing mapping on TCP/UDP ports or ICMP identifiers for a specific user.

- Session Range - This specifies a consecutive TCP/UDP port numbers or ICMP identifiers, indicated by the port/identifier with the smallest numerical number and the port/identifier with the largest numerical number, inclusively.

- Internal IP Address - The IP address that is used as a source IP address in an outbound IP packet sent toward a CGN device in the internal realm. In IPv4 case, it is typically a private address [RFC1918].

- External IP Address - The IP address that is used as a source IP address in an outbound IP packet after traversing a CGN device in

the external realm. In IPv4 case, it is typically a global and routable IP address.

- Internal Port - The internal port is a UDP or TCP port, or an ICMP identifier, which is allocated by a host or application behind a CGN device for an outbound IP packet in the internal realm.

- External Port - The external port is a UDP or TCP port, or an ICMP identifier, which is allocated by a CGN device upon receiving an outbound IP packet in the internal realm, and is used to replace the internal port that is allocated by a user or application.

- External realm - The networking segment where IPv4 public addresses are used in respective of CGN.

- Internal realm - The networking segment that is behind a CGN and where IPv4 private addresses are used.

- Mapping - This term in this document associates with CGN for a relationship between an internal IP address, internal port and the protocol, and an external IP address, external port, and the protocol.

Note the terms "internal IP address", "internal port", "internal realm", "external IP address", "external port", "external realm", and "mapping" and their semantics are the same as in [I-D.ietf-pcp-base], and [I-D.ietf-behave-lsn-requirements].

3. Managing CGN Port Behavior using RADIUS

In a broadband network, customer information is usually stored on a RADIUS server, and the BNG hosts the NAS. The communication between the NAS and the RADIUS server is triggered by a subscriber when the user signs on to the Internet service, where either PPP or DHCP/ DHCPv6 is used. When a user signs on, the NAS sends a RADIUS Access-Request message to the server, which validates the request, and if succeeds, it in turn sends back a RADIUS Access-Accept message, which carries configuration information specific to that user, back to the NAS, where some of the information would pass on to the requesting user via PPP or DHCP/DHCPv6.

A CGN function in a broadband network would most likely reside on a BNG, and in that case, parameters for CGN port/identifier mapping behavior for users can be configured on the RADIUS server, and when a user signs on to the Internet service, the associated parameters can be conveyed to the NAS, and proper configuration is accomplished on the CGN device for that user.

Also, CGN operation status such as CGN port/identifier allocation and de-allocation for a specific user on the BNG can also be transmitted back to the RADIUS server for accounting purpose using RADIUS protocol.

Using RADIUS protocol that has already been deployed in broadband networks to manage BNG-based CGN introduces little overhead to the existing network operation.

In the following sub-sections, we describe how to manage CGN behavior using RADIUS protocol, with required RADIUS extensions proposed in Section 4.

3.1. Configure CGN Session Limit

In the face of IPv4 address shortage, there are currently proposals to multiplex multiple subscribers' connections over a smaller number of shared IPv4 addresses, such as Carrier Grade NAT [I-D.ietf-behave-lsn-requirements], Dual-Stack Lite [I-D.ietf-softwire-dual-stack-lite], NAT64 [RFC6146], etc. As a result, a single IPv4 public address may be shared by several 10s, 100s, etc. of subscribers. As indicated in [I-D.ietf-intarea-shared-addressing-issues], it is therefore necessary to impose limits on the total number of ports available to an individual subscriber to ensure that the shared resource, i.e., the IPv4 address remains available in some capacity to all the subscribers using it, and port limiting is also documented in [I-D.ietf-behave-lsn-requirements] as a requirement.

There are two practical granularities to impose such a limit. One is to define a session limit that is imposed to the total number of TCP and UDP ports, plus the number of ICMP identifiers, for a specific subscriber. Alternatively, a session limit can be specified for the sum of TCP ports and UDP ports, or a separate session limit for TCP ports and UDP ports, respectively, and another session limit for ICMP identifiers.

The per-subscriber based session limit(s) is configured on a RADIUS server, along with other user information such as credentials. The value of these session limit(s) is based on service agreement and its specification is out of the scope of this document.

When a subscriber signs on to the Internet service successfully, the session limit(s) for the subscriber is passed to the BNG based NAS, where CGN also locates, using a new RADIUS attribute called CGN-Session-Limit (defined in Section 4.1), along with other configuration parameters. While some parameters are passed to the subscriber, the session limit(s) is recorded on the CGN device for

imposing the usage of TCP/UDP ports and ICMP identifiers for that subscriber.

Figure-1 illustrates how RADIUS protocol is used to configure the maximum number of TCP/UDP ports for a given subscriber on a NAT44 device.

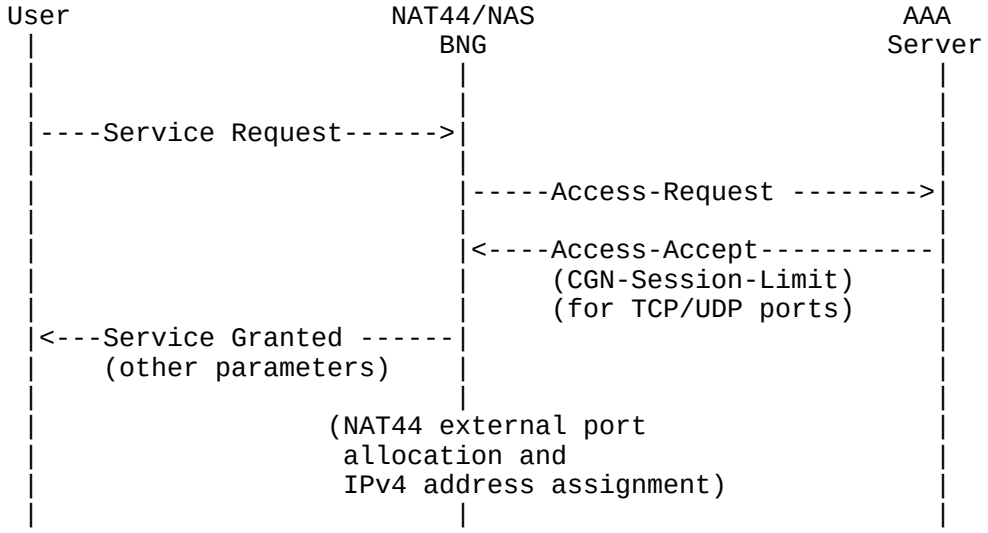


Figure 1: RADIUS Message Flow for Configuring NAT44 Port Limit

The session limit(s) created on a CGN device for a specific user using RADIUS extension as described above may be changed using RADIUS CoA message [RFC5176] that carries the same RADIUS attribute. The CoA message may be sent from the RADIUS server, or from a WEB Portal, directly to the NAS, which once accepts and sends back a RADIUS CoA AcK message, the new session limit replaces the previous one.

Figure-2 illustrates how RADIUS protocol is used to increase the TCP/UDP port limit from 1024 to 2048 on a NAT44 device for a specific user.

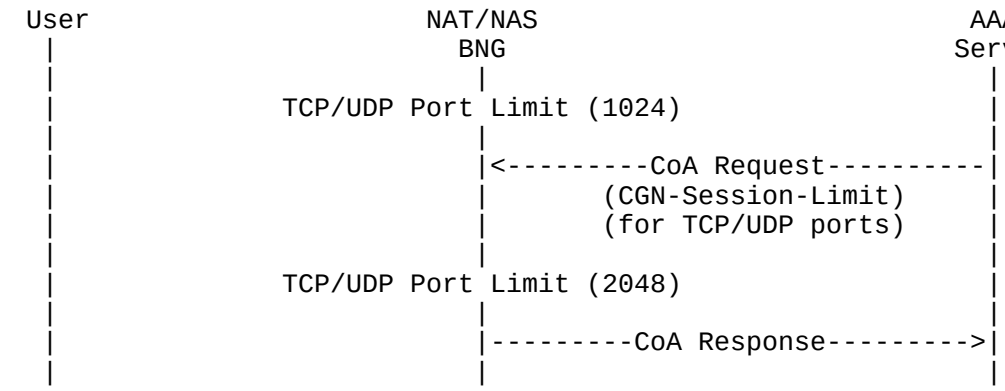


Figure 2: RADIUS Message Flow for changing a user's NAT44 port limit

3.2. Report CGN Session Allocation or De-allocation

Upon obtaining the session limit(s) for a subscriber, the CGN device needs to allocate a TCP/UDP port or an ICMP identifiers for the subscriber when receiving a new IP flow sent from that subscriber.

As one practice, a CGN may allocate a bulk of TCP/UDP ports or ICMP identifiers once at a time for a specific user, instead of one port/identifier at a time, and within each session bulk, the ports/identifiers may be randomly distributed or in consecutive fashion. When a CGN device allocates bulk of TCP/UDP ports and ICMP identifiers in consecutive order, the information can be easily conveyed to the RADIUS server by a new RADIUS attribute called the CGN-Session-Range (defined in Section 4.2). The CGN device may allocate one or more TCP/UDP port ranges or ICMP identifier ranges, or generally called session ranges, where each range contains some consecutive numbers representing TCP/UDP ports or ICMP identifiers, and the total number of sessions must be less or equal to the associated session limit defined for that subscriber. A CGN device may choose to allocate a small session range, and allocate more at a later time as needed; such practice is good because its randomization in nature.

At the same time, the CGN device also needs to decide the shared IPv4 address for that subscriber. The shared IPv4 address and the pre-allocated session range are both passed to the RADIUS server.

When a subscriber initiates an IP flow, the CGN device randomly selects a TCP/UDP port or ICMP identifier from the associated and pre-allocated session range for that subscriber to replace the original source TCP/UDP port or ICMP identifier, along with the replacement of the source IP address by the shared IPv4 address.

At anytime, a CGN device may decide to "free" some consecutive TCP/ UDP ports or ICMP identifiers that have been allocated for a specific subscriber but not currently in use, and with that, the CGN device must send the information of the de-allocated session range along with the shared IPv4 address to the RADIUS server.

Figure-3 illustrates how RADIUS protocol is used to report consecutive ports allocated and de-allocated, respectively, by a NAT44 device for a specific user to the RADIUS server.

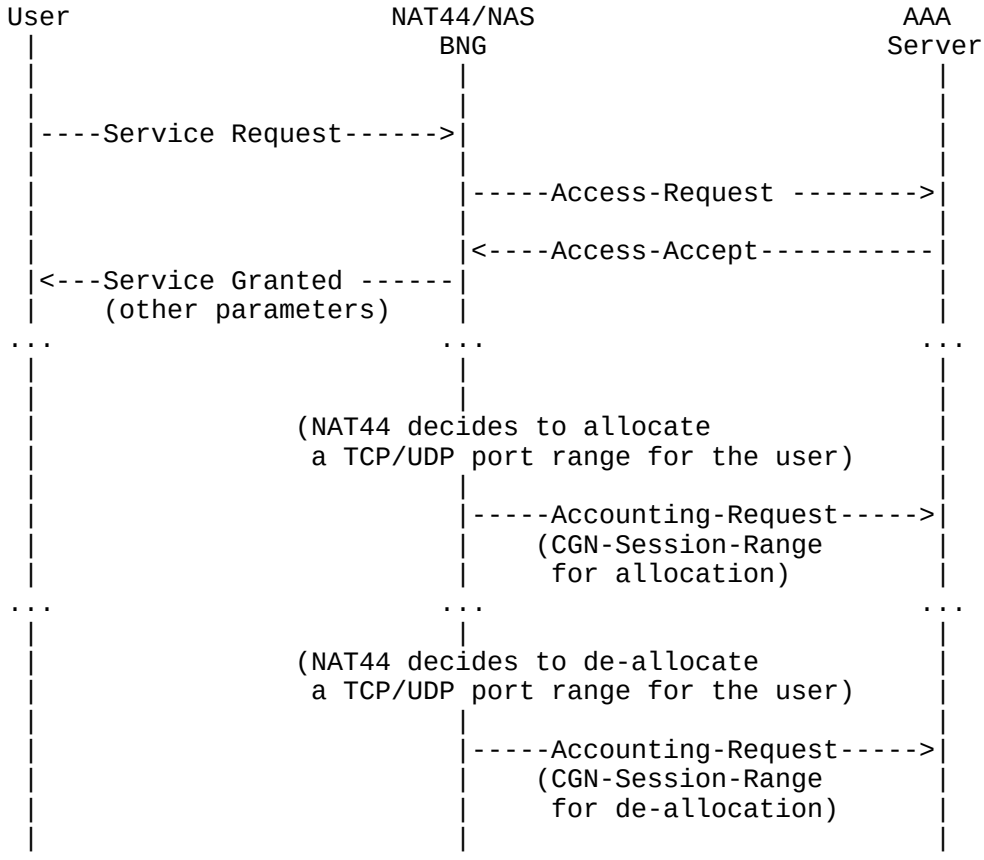


Figure 3: RADIUS Message Flow for reporting NAT44 allocation/ de-allocation of consecutive ports

Note using RADIUS extension as proposed in this document to report back to the RADIUS server about allocation/de-allocation of TCP/UDP

ports and ICMP identifiers requires the ports/identifiers in the session range are consecutive. However, a CGN might choose bulk session allocation but the ports/identifiers are not consecutive, or the ports/identifiers are allocated on individual basis; efficient mechanism to report session allocation/de-allocation to the RADIUS server using these methods is under discussion and is currently outside of the scope of this document. Also, the impacts on utilization of TCP/UDP ports and ICMP identifiers, CGN logging, security etc. with different allocation schemes are referred to Section 5 of [I-D.ietf-behave-lsn-requirements].

3.3. Configure CGN Forwarding Port Mapping

In most scenarios, the port mapping on a NAT device is dynamically created when the IP packets of an IP connection initiated by a user arrives. For some applications, the port mapping needs to be pre-defined allowing IP packets of applications from outside a CGN device to pass through and "port forwarded" to the correct user located behind the CGN device.

Port Control Protocol or PCP [I-D.ietf-pcp-base], provides a mechanism to create a mapping from an external IP address and port to an internal IP address and port on a CGN device just to achieve the "port forwarding" purpose. PCP is a server-client protocol capable of creating or deleting a mapping along with a rich set of features on a CGN device in dynamic fashion. In some deployment, all users need is a few, typically just one pre-configured port mapping for applications such as web cam at home, and the lifetime of such a port mapping remains valid throughout the duration of the customer’s Internet service connection time. In such an environment, it is possible to statically configure a port mapping on the RADIUS server for a user and let the RADIUS protocol to propagate the information to the associated CGN device.

Figure-4 illustrates how RADIUS protocol is used to configure a forwarding port mapping on a NAT44 device by using RADIUS protocol.

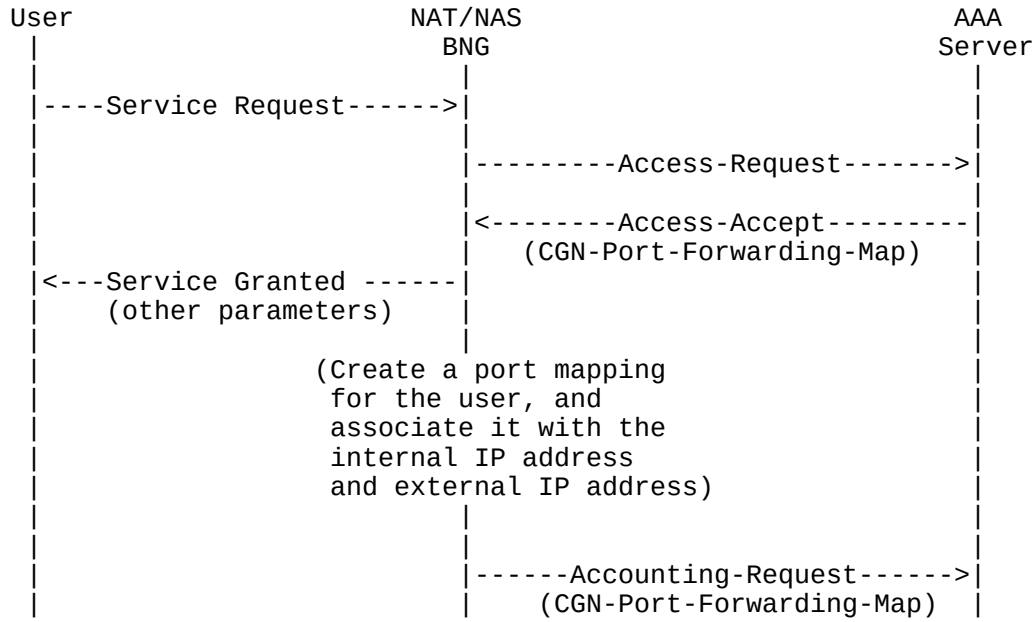


Figure 4: RADIUS Message Flow for configuring a forwarding port mapping

A port forwarding mapping that is created on a CGN device using RADIUS extension as described above may also be changed using RADIUS CoA message [RFC5176] that carries the same RADIUS associate. The CoA message may be sent from the RADIUS server, or from a WEB Portal, directly to the NAS, which once accepts and sends back a RADIUS CoA AcK message, the new port forwarding mapping then replaces the previous one.

Figure-5 illustrates how RADIUS protocol is used to change an existing port mapping from (a:X) to (a:Y), where "a" is an internal port, and "X" and "Y" are external ports, respectively, for a specific user with a specific IP address

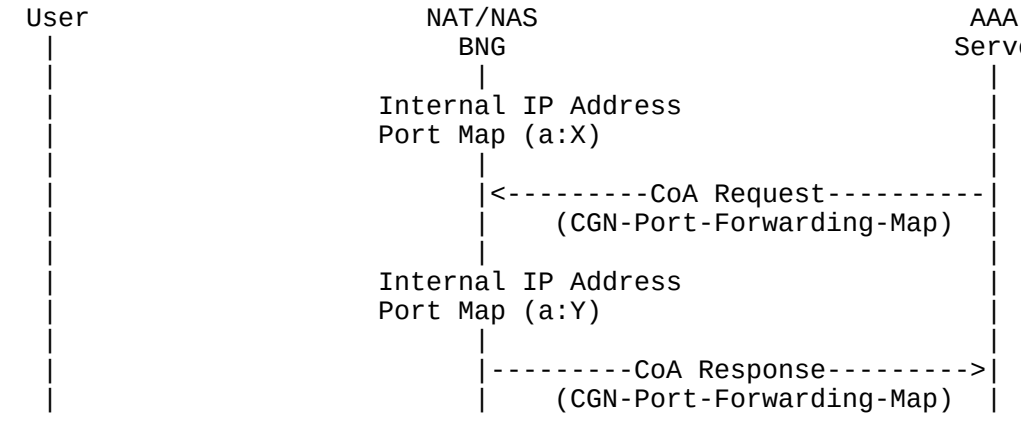


Figure 5: RADIUS Message Flow for changing a user's forwarding port mapping

3.4. An Example

An ISP assigns TCP/UDP 500 ports for the subscriber Joe based on a service agreement. This number is the limit that can be used for TCP/UDP ports on a NAT44 device for Joe, and is configured on a RADIUS server. Also, Joe asks for a pre-defined port forwarding mapping on the NAT44 device for his web cam applications (external port 5000 maps to internal port 80).

When Joe successfully connects to the Internet service, the RADIUS server conveys the TCP/UDP port limit (1000) and the forwarding port mapping (external port 5000 to internal port 80) to the NAT44 device, using CGN-Session-Limit attribute and CGN-Forwarding-Port-Map attribute, respectively, carried by an Access-Accept message to the BNG where NAS and CGN co-located.

Upon receiving the first outbound IP packet sent from Joe's laptop, the NAT44 device decides to allocate a small port pool that contains 40 consecutive ports, from 3500 to 3540, inclusively, and also assign a shared IPv4 address 192.0.2.15, for Joe. The NAT44 device also randomly selects one port from the allocated range (say 3519) and use that port to replace the original source port in outbound IP packets.

For accounting purpose, the NAT44 device passes this port range (3500-3540) and the shared IPv4 address 192.0.2.15 together to the RADIUS server using CGN-Session-Range attribute carried by an Accounting-Request message.

When Joe works on more applications with more outbound IP sessions and the port pool (3500-3540) is close to exhaust, the NAT44 device

allocates a second port pool (8500-8800) in a similar fashion, and also passes the new port range (8500-8800) and IPv4 address 192.0.2.15 together to the RADIUS server using CGN-Session-Range attribute carried by an Accounting-Request message. Note when the CGN allocates more ports, it needs to assure that the total number of ports allocated for Joe is within the limit.

Joe decides to upgrade his service agreement with more TCP/UDP ports allowed (up to 1000 ports). The ISP updates the information in Joe's profile on the RADIUS server, which then sends a CoA-Request message that carries the CGN-Session-Limit attribute with 1000 ports to the NAT44 device; the NAT44 device in turn sends back a CoA-Ack message. With that, Joe enjoys more available TCP/UDP ports for his applications.

When Joe travels, most of the IP sessions are closed with their associated TCP/UDP ports released on the NAT44 device, which then sends the relevant information back to the RADIUS server using CGN-Session-Range attribute carried by Accounting-Request message.

Throughout Joe's connection with his ISP Internet service, applications can communicate with his web cam at home from external realm directly traversing the pre-configured mapping on the CGN device.

When Joe disconnects from his Internet service, the CGN device will de-allocate all TCP/UDP ports as well as the port-forwarding mapping, and send the relevant information to the RADIUS server.

4. RADIUS Attributes

4.1. CGN-Session-Limit Attribute

Description

This attribute specifies the limit of TCP ports, or UDP ports, or the sum of the two, or ICMP identifiers, or the sum of the three, which is configured on a CGN device corresponding to a specific subscriber for CGN operation.

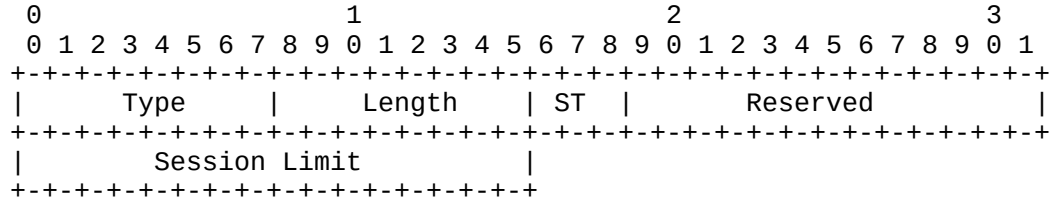
The CGN-Session-Limit MAY appear in an Access-Accept packet, it MAY also appear in an Access-Request packet as a hint by the CGN device, which is co-allocated with the NAS, to the RADIUS server as a preference, although the server is not required to honor such a hint.

The CGN-Session-Limit MAY appear in an CoA-Request packet.

The CGN-Session-Limit MAY appear in an Accounting-Request packet.

The CGN-Session-Limit MUST NOT appear in any other RADIUS packets.

The format of the CGN-Session-Limit RADIUS attribute format is shown below. The fields are transmitted from left to right.



Type:

TBA1 for CGN-Port-Limit.

Length:

6 Octets. This field indicates the total length in octets of this attribute including the Type and the Length field.

ST (Session Type):

This 3-bit field contains a value that indicates the applicability of the Session Limit as follows:

0:

The limit as specified is applied to the sum of TCP ports, UDP ports, and ICMP Identifiers as a whole.

1:

The limit as specified is applied to the sum of TCP ports and UDP ports.

2:

The limit as specified is applied to TCP ports.

3:

The limit as specified is applied to UDP ports.

4:

The limit as specified is applied to ICMP Identifiers.

5-7:

These values are not used.

Reserved:

This field is set to zero by the sender and ignored by the receiver.

Session Limit:

This field contains the maximum number that is imposed to the total number of TCP ports, or UDP ports, or the sum of the two, or ICMP Identifiers, or the sum of the three, depending on the value in the Session Type field, that the specific user can use during CGN operation.

4.2. CGN-Session-Range Attribute

Description

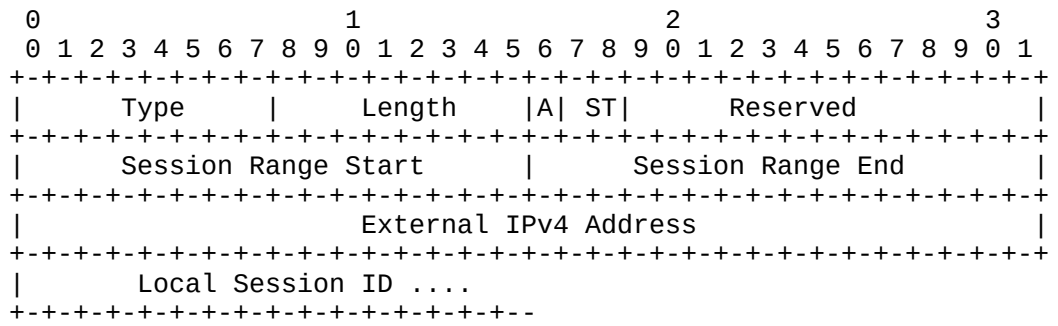
This attribute contains a range of consecutive numbers for TCP ports or UDP ports, or both, or for ICMP Identifiers, which has been allocated or de-allocated by a CGN device for a given subscriber, along with an external IPv4 address that is associated with any TCP/UDP port or ICMP identifier in the range.

In some CGN deployment scenarios as described such as L2NAT [I-D.miles-behave-l2nat] and DS-Extra-Lite [I-D.arkko-dual-stack-extra-lite], parameters at a customer premise such as MAC address, interface ID, VLAN ID, PPP session ID, VRF ID, etc., may also be required to pass to the RADIUS server as part of the accounting record.

The CGN-Session-Range MAY appear in an Accounting-Request packet.

The CGN-Session-Range MUST NOT appear in any other RADIUS packets.

The format of the CGN-Session-Range RADIUS attribute format is shown below. The fields are transmitted from left to right.



Type:

TBA2 for CGN-Alloc-Port-Range.

Length:

12 Octets. This field indicates the total length in octets of this attribute including the Type and the Length field.

A-bit Flag:

This field is set to 0 or 1, indicates that the session range has been allocated or de-allocated, respectively, by the CGN for a specific user.

ST (Session Type):

This 2-bit field contains a value that indicates the semantics of the session range as follows:

0:

The session range is applied to TCP ports.

1:

The session range is applied to UDP ports.

2:

The session range is applied to the both TCP ports and UDP ports.

3:

The session range is applied to the ICMP Identifiers.

Reserved:

This field is set to zero by the sender and ignored by the receiver.

External Session Range Start:

This field contains the smallest TCP/UDP Port number or the smallest ICMP identifier number in the session range, which contains consecutive TCP/UDP ports or ICMP identifiers, depending on the value of Session Type.

External Session Range End:

This field contains the largest TCP/UDP port number or the largest ICMP identifier number in the session range, which contains consecutive TCP/UDP ports or ICMP identifiers, depending on the value of Session Type.

External IPv4 Address:

This field contains the IPv4 address assigned to the associated subscriber to be used in the external realm.

Local Session ID:

This is an optional field and if presents, it contains a local session identifier at the customer premise, such as MAC address, interface ID, VLAN ID, PPP sessions ID, VRF ID,etc.The length of this field equals to the total attribute length minus 12 octets.

4.3. CGN-Forwarding-Port-Map Attribute

Description

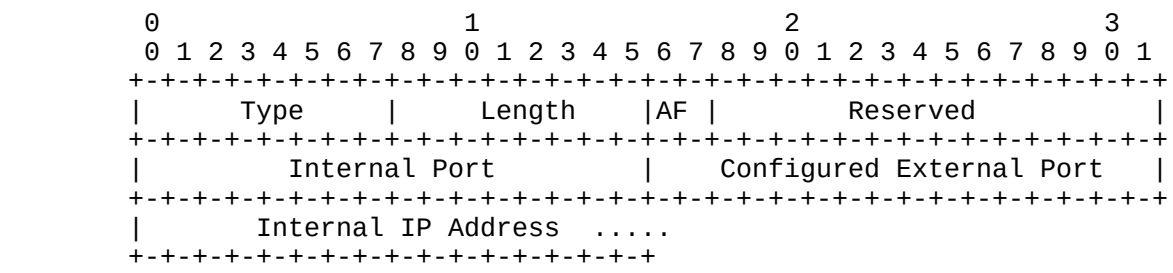
This attribute contains a 16-bit Internal Port that identifies the source TCP/UDP port number of an IP packet sent by the user, or the destination port number of an IP packet destined to the user, and in both cases, the IP packet travels behind the NAT device. Also they contain a 16-bit Configured External Port that identifies the source TCP/UDP port number of an IP packet sent by the user, or the destination port number of an IP packet destined to the user, and in both cases, the IP packet travels outside of the NAT device. In addition, the attribute may contain a 32-bit IPv4 address or a 128-

bit IPv6 address, respectively, as their respective NAT mappings internal IP address. Together, the port pair and IP address determine the port mapping rule for a specific IP flow that traverses a NAT device.

The attribute MAY appear in an Access-Accept packet, and may also appear in an Accounting-Request packet. In either case, the attribute MUST NOT appear more than once in a single packet.

The attribute MUST NOT appear in any other RADIUS packets.

The format of the CGN-Forwarding-Port-Map RADIUS attribute format is shown below. The fields are transmitted from left to right.



Type:

TBA3 for CGN-Forwarding-Port-Map.

Length:

This field indicates the total length in octets of this attribute including the Type and the Length field. Depending on the value of the AF field, the length could be 8, 12 or 24 octets.

AF (Address Family):

This 2-bit field contains a value that indicates address family of the internal IP address at the mapping as follows:

0:

There is no internal address attached.

1:

The internal address is an IPv4 address.

2:

The internal address is an IPv6 address.

3:

Unused.

Reserved:

This field is set to zero by the sender and ignored by the receiver.

Internal Port:

This field contains the internal port for the CGN mapping.

Configured External Port:

This field contains the external port for the CGN mapping.

Internal IP Address:

This field may or may not present, and when it does, contains the internal IPv4 or IPv6 address for the CGN mapping.

5. Table of Attributes

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

Request	Accept	Reject	Challenge	Accounting Request	#	Attribute
0-1	0-1	0	0	0-1	TBA1	CGN-Session-Limit
0-1	0-1	0	0	0-1	TBA2	CGN-Session-Range
0-1	0-1	0	0	0-1	TBA3	CGN-Forwarding-Port-Map

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

6. Security Considerations

This document does not introduce any security issue than what has been identified in [RFC2865].

7. IANA Considerations

This document requires new code point assignment for the three new RADIUS attributes as follows: :

- CGN-Session-Limit
- CGN-Session-Range
- CGN-Forwarding-Port-Map

8. Acknowledgements

Many thanks to Dan Wing, Roberta Maglione, Daniel Derksen, and David Thaler for their useful comments and suggestions.

9. References

9.1. Normative References

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[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.

9.2. Informative References

[I-D.arkko-dual-stack-extra-lite]
Arkko, J., Eggert, L., and M. Townsley, "Scalable Operation of Address Translators with Per-Interface Bindings", draft-arkko-dual-stack-extra-lite-05 (work in progress), February 2011.

[I-D.ietf-behave-lsn-requirements]
Perreault, S., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common requirements for IP address sharing schemes", draft-ietf-behave-lsn-requirements-01 (work in progress), March 2011.

[I-D.ietf-intarea-shared-addressing-issues]
Ford, M., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", draft-ietf-intarea-shared-addressing-issues-05 (work in progress), March 2011.

[I-D.ietf-pcp-base]
Wing, D., Cheshire, S., Boucadair, M., and R. Penno, "Port Control Protocol (PCP)", draft-ietf-pcp-base-10 (work in progress), April 2011.

[I-D.ietf-softwire-dual-stack-lite]
Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", draft-ietf-softwire-dual-stack-lite-08 (work in progress), May 2011.

[I-D.miles-behave-l2nat]
Miles, D. and M. Townsley, "Layer2-Aware NAT", draft-miles-behave-l2nat-00 (work in progress), March 2009.

[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.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, April 2011.

Author's Address

Dean Cheng
Huawei Technologies
2330 Central Expressway
95050
USA

Email: dean.cheng@huawei.com

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/1id-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

Table of Contents

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:

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- 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:

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- 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 Taffic
: 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 the 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:

- Assignee: change "Mike McCauley" to "IESG".
- Contact: change "Mike McCauley" to "IETF Chair"
- Reference: Add this document as a reference
- 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 acceptible only when DTLS is used with an 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 searchs 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, howrver, 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

Table of Contents

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.

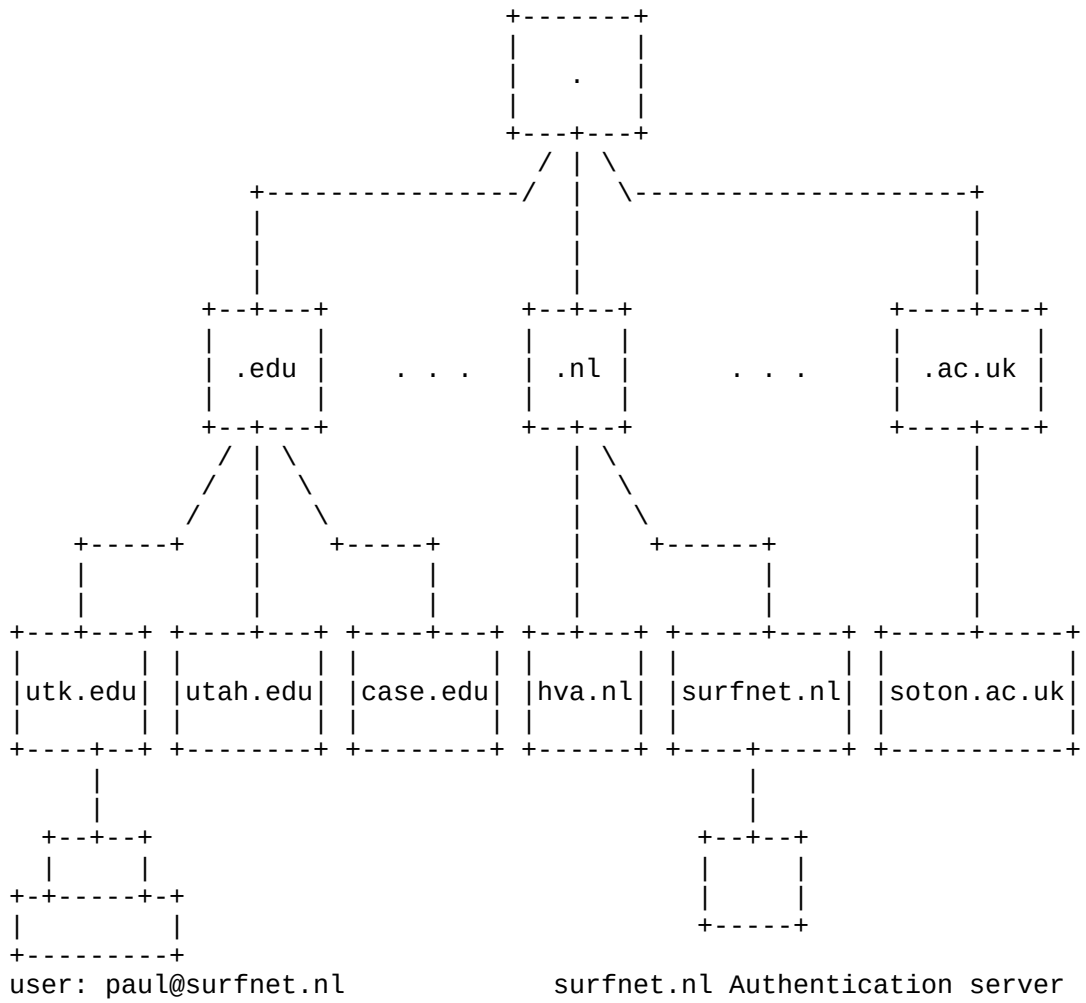


Figure 1: RADIUS hierarchy based on Top-Level Domain partitioning

However, such partitioning is not always possible. As an example, in one real-life deployment, the administrative boundaries and RADIUS forwarding servers are 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:

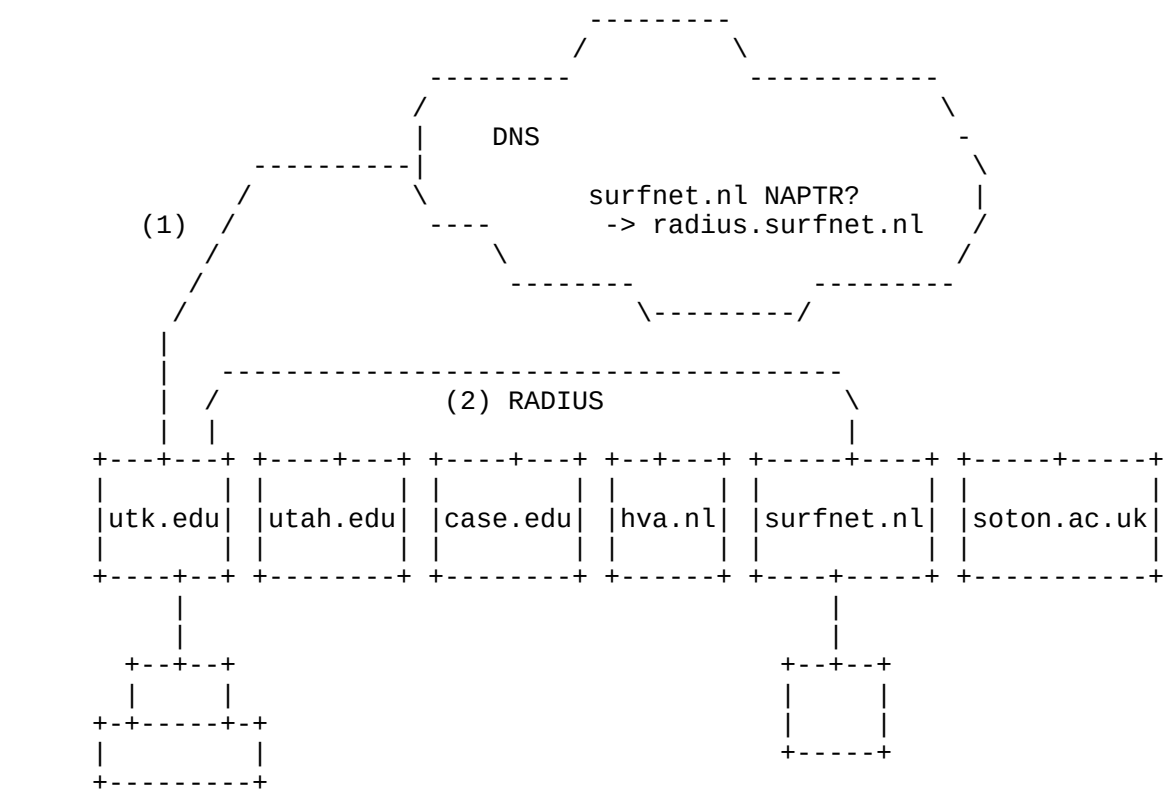


Figure 2: RADIUS hierarchy based on Top-Level Domain partitioning

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

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

2.1.3. 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" "" roamserv.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, preceeded 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’ = (set of {hostname; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this hostname) } for all terminal lookup results).

12. Proceed with step 18.

13. Generate R’ = (prefix R with "_radiustls._tcp." 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 = { empty set }, O-2 = BACKOFF_TIME and terminate.

16. If the result is a negative DNS response, O-1 = { empty set }, O-2 = min { O-2, Effective TTL (TTL value of the SOA record) } and terminate.

17. O’ = (set of {hostname; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this result) } for all hostnames).

18. Generate O-1 by resolving hostnames in O’ into corresponding A and/or AAAA addresses: O-1 = (set of {IP address; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this result) } 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 {IP address; port}. If yes, O-1 = { empty set }, O-2 = BACKOFF_TIME, log error, Terminate (see next section for a rationale). If no, O is the result of dynamic discovery. Terminate.

20. O-1 = { empty set }, O-2 = 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 priorty, 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 possiblity that a subsequent hop has a statically configured next-hop which leads to an earlier host in the loop. Another reason for occuring 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,

- 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.

- 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.

- 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.

- 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,

- 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, ...).

- 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.

- 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.

- 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:

- 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 wide-spread 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 occurences 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 16, 2013

W. Dec, Ed.
Cisco Systems, Inc.
B. Sarikaya
Huawei USA
G. Zorn
Network Zen
D. Miles
Google
B. Lourdelet
Juniper Networks

February 12, 2013

RADIUS attributes for IPv6 Access Networks
draft-ietf-radext-ipv6-access-16

Abstract

This document specifies additional IPv6 RADIUS attributes useful in residential broadband network deployments. The attributes, which are used for authorization and accounting, enable assignment of a host IPv6 address and IPv6 DNS server address via DHCPv6; assignment of an IPv6 route announced via router advertisement; assignment of a named IPv6 delegated prefix pool; and assignment of a named IPv6 pool for host DHCPv6 addressing.

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].

Status of this Memo

Table of Contents

1. Introduction

This document specifies additional RADIUS attributes used to support configuration of DHCPv6 and/or ICMPv6 Router Advertisement (RA) parameters on a per-user basis. The attributes, which complement those defined in [RFC3162] and [RFC4818], support the following:

- o Assignment of specific IPv6 addresses to hosts via DHCPv6.
- o Assignment of an IPv6 DNS server address, via DHCPv6 or Router Advertisement [RFC6106].
- o Configuration of more specific routes to be announced to the user via the Route Information Option defined in [RFC4191] Section 2.3.
- o The assignment of a named delegated prefix pool for use with "IPv6 Prefix Options for DHCPv6" [RFC3633].
- o The assignment of a named stateful address pool for use with DHCPv6 stateful address assignment [RFC3315].

2. Deployment Scenarios

The extensions in this draft are intended to be applicable across a wide variety of network access scenarios where RADIUS is involved. One such typical network scenario is illustrated in Figure 1. It is composed of a IP Routing Residential Gateway (RG) or host, a Layer 2 Access-Node (AN) e.g. a Digital Subscriber Line Access Multiplexer - DSLAM, an IP Network Access Servers (NASes), and an Authentication Authorization & Accounting (AAA) server.

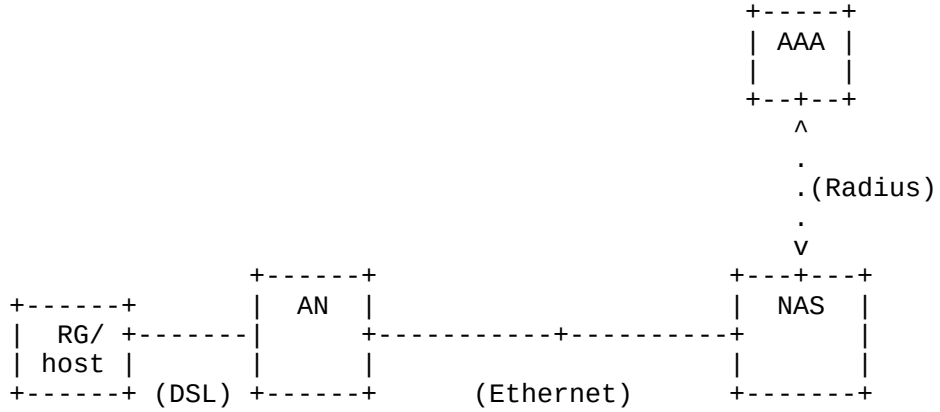


Figure 1

In the depicted scenario the NAS may utilize an IP address configuration protocol (e.g. a DHCPv6 server) to handle address assignment to RGs/hosts. The RADIUS server authenticates each RG/host and returns to the attributes used for authorization and accounting. These attributes can include a host's IPv6 address, a DNS server address and a set of IPv6 routes to be advertised via any suitable protocol, eg ICMPv6 (Neighbour Discovery). The name of a prefix pool to be used for DHCPv6 Prefix Delegation, or the name of an address pool to be used for DHCPv6 address assignment can also be attributes provided to the NAS by the RADIUS AAA server.

The following sub-sections discuss how these attributes are used in more detail.

2.1. IPv6 Address Assignment

DHCPv6 [RFC3315] provides a mechanism to assign one or more non-temporary IPv6 addresses to hosts. To provide a DHCPv6 server residing on a NAS with one or more IPv6 addresses to be assigned, this document specifies the Framed-IPv6-Address Attribute.

While [RFC3162] permits an IPv6 address to be specified via the combination of the Framed-Interface-Id and Framed-IPv6-Prefix attributes, this separation is more natural for use with PPP's IPv6 Control Protocol than it is for use with DHCPv6, and the use of a single IPv6 address attribute makes for easier processing of accounting records.

Since DHCPv6 can be deployed on the same network as ICMPv6 stateless (SLAAC) [RFC4862], it is possible that the NAS will require both stateful and stateless configuration information. Therefore it is possible for the Framed-IPv6-Address, Framed-IPv6-Prefix and Framed-Interface-Id attributes [RFC3162] to be included within the same packet. To avoid ambiguity in this case, the Framed-IPv6-Address attribute is intended for authorization and accounting of DHCPv6-assigned addresses and the Framed-IPv6-Prefix and Framed-Interface-Id attributes used for authorization and accounting of addresses assigned via SLAAC.

2.2. DNS Servers

DHCPv6 provides an option for configuring a host with the IPv6 address of a DNS server. The IPv6 address of a DNS server can also be conveyed to the host using ICMPv6 with Router Advertisements, via the [RFC6106] option. To provide the NAS with the IPv6 address of a DNS server, this document specifies the DNS-Server-IPv6-Address Attribute.

2.3. IPv6 Route Information

An IPv6 Route Information option, defined in [RFC4191] is intended to be used to inform a host connected to the NAS that a specific route is reachable via any given NAS.

This document specifies the RADIUS attribute that allows the AAA server to provision the announcement by the NAS of a specific Route Information Option to an accessing host. The NAS may advertise this route using the method defined in [RFC4191], or using other equivalent methods. Any other information, such as preference or life-time values, that is to be present in the actual announcement using a given method is assumed to be determined by the NAS using means not scoped by this document (e.g. Local configuration on the NAS).

While the Framed-IPv6-Prefix attribute defined in [RFC3162] Section 2.3 causes the route to be advertised in an RA, it cannot be used to configure more specific routes. While the Framed-IPv6-Route attribute defined in [RFC3162] Section 2.5 causes the route to be configured on the NAS, and potentially announced via an IP routing protocol, depending on the value of Framed-Routing, it does not result in the route being announced in an RA.

2.4. Delegated IPv6 Prefix Pool

DHCPv6 Prefix Delegation (DHCPv6-PD) [RFC3633] involves a delegating router selecting a prefix and delegating it on a temporary basis to a requesting router. The delegating router may implement a number of strategies as to how it chooses what prefix is to be delegated to a requesting router, one of them being the use of a local named prefix pool. The Delegated-IPv6-Prefix-Pool attribute allows the RADIUS server to convey a prefix pool name to a NAS hosting a DHCPv6-PD server and acting as a delegating router.

Since DHCPv6 Prefix Delegation can be used with SLAAC on the same network, it is possible for the Delegated-IPv6-Prefix-Pool and Framed-IPv6-Pool attributes to be included within the same packet. To avoid ambiguity in this scenario, use of the Delegated-IPv6-Prefix-Pool attribute should be restricted to authorization and accounting of prefix pools used in DHCPv6 Prefix Delegation and the Framed-IPv6-Pool attribute should be used for authorization and accounting of prefix pools used in SLAAC.

2.5. Stateful IPv6 address pool

DHCPv6 [RFC3315] provides a mechanism to assign one or more non-temporary IPv6 addresses to hosts. Section 3.1 introduces the

Framed-IPv6-Address attribute to be used for providing a DHCPv6 server residing on a NAS with one or more IPv6 addresses to be assigned to the clients. An alternative way to achieve a similar result is for the NAS to select the IPv6 address to be assigned from an address pool configured for this purpose on the NAS. This document specifies the Stateful-IPv6-Address-Pool attribute to allow the RADIUS server to convey a pool name to be used for such stateful DHCPv6 based addressing, and any subsequent accounting.

3. Attributes

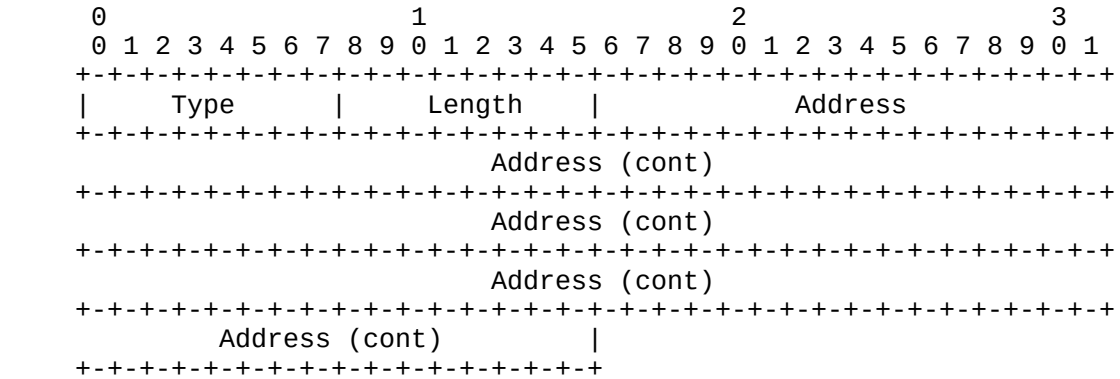
The fields shown in the diagrams below are transmitted from left to right.

3.1. Framed-IPv6-Address

This attribute indicates an IPv6 address that is assigned to the NAS-facing interface of the RG/host. It MAY be used in Access-Accept packets, and MAY appear multiple times. It MAY be used in an Access-Request packet as a hint by the NAS to the RADIUS server that it would prefer these IPv6 address(es), but the RADIUS server is not required to honor the hint. Since it is assumed that the NAS will add a route corresponding to the address, it is not necessary for the RADIUS server to also send a host Framed-IPv6-Route attribute for the same address.

This attribute can be used by a DHCPv6 process on the NAS to assign a unique IPv6 address to the RG/host.

A summary of the Framed-IPv6-Address attribute format is shown below. The format of the address is as per [RFC3162].



Type

TBA1 for Framed-IPv6-Address

Length

18

Address

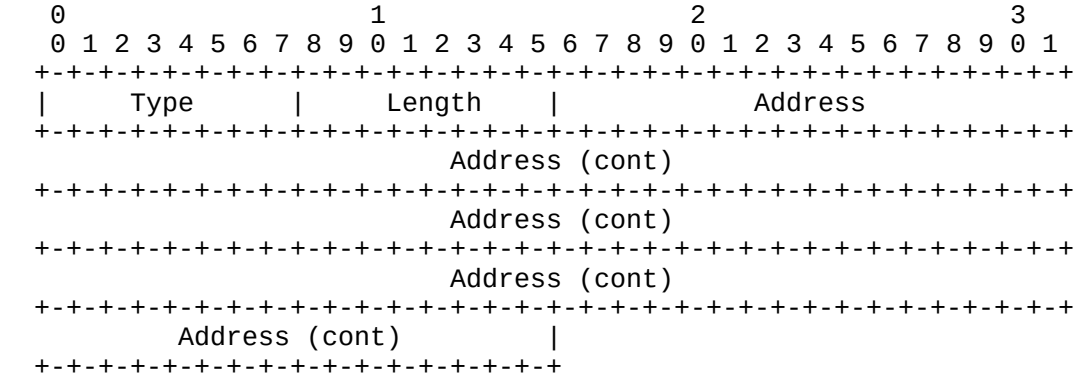
The IPv6 address field contains a 128-bit IPv6 address.

3.2. DNS-Server-IPv6-Address

The DNS-Server-IPv6-Address attribute contains the IPv6 address of a DNS server. This attribute MAY be included multiple times in Access-Accept packets, when the intention is for a NAS to announce more than one DNS server addresses to a RG/host. The same order of the attributes is expected to be followed in the announcements to the RADIUS client. The attribute MAY be used in an Access-Request packet as a hint by the NAS to the RADIUS server regarding the DNS IPv6 address, but the RADIUS server is not required to honor the hint.

The content of this attribute can be inserted in a DHCPv6 option as specified in [RFC3646] or in an IPv6 Router Advertisment as per [RFC6106].

A summary of the DNS-Server-IPv6-Address attribute format is given below. The format of the address is as per [RFC3162].



Type

TBA2 for DNS-Server-IPv6-Address

Length

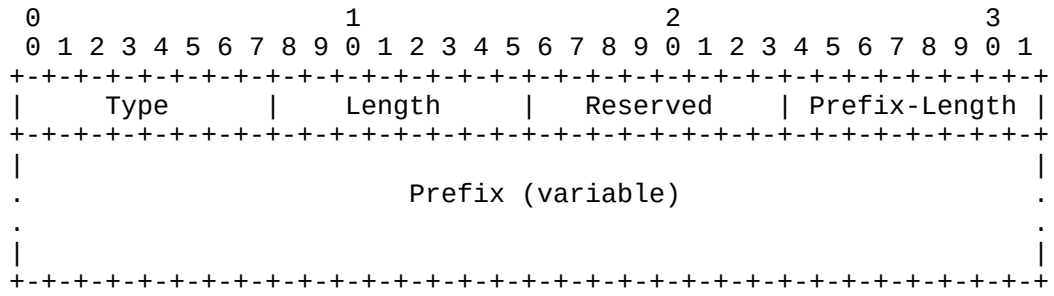
18

Address

The 128-bit IPv6 address of a DNS server.

3.3. Route-IPv6-Information

This attribute specifies a prefix (and corresponding route) for the user on the NAS, which is to be announced using the Route Information Option defined in "Default Router Preferences and More Specific Routes" [RFC4191] Section 2.3. It is used in the Access-Accept packet and can appear multiple times. It MAY be used in an Access-Request packet as a hint by the NAS to the RADIUS server, but the RADIUS server is not required to honor the hint. The Route-IPv6-Information attribute format is depicted below. The format of the prefix is as per [RFC3162].



Type

TBA3 for Route-IPv6-Information

Length

Length in bytes. At least 4 and no larger than 20; typically 12 or less.

Prefix Length

8-bit unsigned integer. The number of leading bits in the prefix that are valid. The value ranges from 0 to 128. The prefix field is 0, 8 or 16 octets depending on Length.

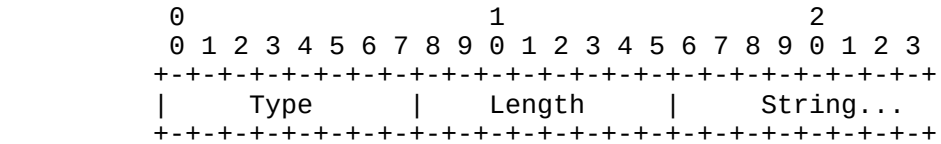
Prefix

Variable-length field containing an IP prefix. The prefix length field contains the number of valid leading bits in the prefix. The bits in the prefix after the prefix length (if any) are reserved and MUST be initialized to zero.

3.4. Delegated-IPv6-Prefix-Pool

This attribute contains the name of an assigned pool that SHOULD be used to select an IPv6 delegated prefix for the user on the NAS. If a NAS does not support prefix pools, the NAS MUST ignore this attribute. It MAY be used in an Access-Request packet as a hint by the NAS to the RADIUS server regarding the pool, but the RADIUS server is not required to honor the hint.

A summary of the Delegated-IPv6-Prefix-Pool attribute format is shown below.



Type

TBA4 for Delegated-IPv6-Prefix-Pool

Length

Length in bytes. At least 3.

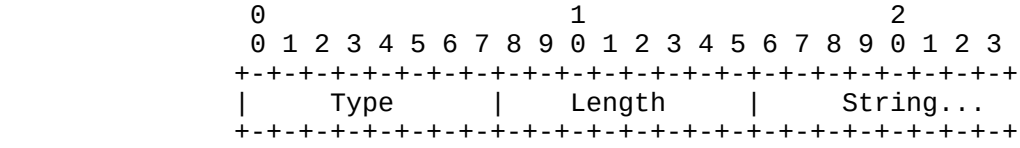
String

The string field contains the name of an assigned IPv6 prefix pool configured on the NAS. The field is not NULL (hexadecimal 00) terminated.

Note: The string data type is as documented in [RFC6158], and carries binary data that is external to the Radius protocol, eg the name of a pool of prefixes configured on the NAS.

3.5. Stateful-IPv6-Address-Pool

This attribute contains the name of an assigned pool that SHOULD be used to select an IPv6 address for the user on the NAS. If a NAS does not support address pools, the NAS MUST ignore this attribute. A summary of the Stateful-IPv6-Address-Pool attribute format is shown below. It MAY be used in an Access-Request packet as a hint by the NAS to the RADIUS server regarding the pool, but the RADIUS server is not required to honor the hint.



Type

TBA5 for Stateful-IPv6-Address-Pool

Length

Length in bytes. At least 3.

String

The string field contains the name of an assigned IPv6 stateful address pool configured on the NAS. The field is not NULL (hexadecimal 00) terminated.

Note: The string data type is as documented in [RFC6158], and carries binary data that is external to the Radius protocol, eg the name of a pool of addresses configured on the NAS.

3.6. Table of attributes

The following table provides a guide to which attributes may be found in which kinds of packets, and in what quantity. The optional inclusion of the options in Access Request messages is intended to allow for a network access server (NAS) to provide the RADIUS server with a hint of the attributes in advance of user authentication, which may be useful in cases where a user re-connects or has a static address. The server is under no obligation to honor such hints.

Request	Accept	Reject	Challenge	Accounting Request	#	Attribute
0+	0+	0	0	0+	TBA1	Framed-IPv6-Address
0+	0+	0	0	0+	TBA2	DNS-Server-IPv6-Address
0+	0+	0	0	0+	TBA3	Route-IPv6-Information
0+	0+	0	0	0+	TBA4	Delegated-IPv6-Prefix-Pool
0+	0+	0	0	0+	TBA5	Stateful-IPv6-Address-Pool

4. Diameter Considerations

Given that the attributes defined in this document are allocated from the standard RADIUS type space (see Section 6), no special handling is required by Diameter entities.

5. Security Considerations

This document specifies additional IPv6 RADIUS attributes useful in residential broadband network deployments. In such networks, the RADIUS protocol may run either over IPv4 or over IPv6 and known security vulnerabilities of the RADIUS protocol, e.g. [SECI], apply to the attributes defined in this document. A trust relationship between a NAS and RADIUS server is expected to be in place, with communication optionally secured by IPSec or TLS [RFC6614] .

6. IANA Considerations

This document requires the assignment of five new RADIUS attribute types in the "Radius Types" registry (currently located at http://www.iana.org/assignments/radius-types for the following attributes:

- Framed-IPv6-Address
- DNS-Server-IPv6-Address
- Route-IPv6-Information
- Delegated-IPv6-Prefix-Pool
- Stateful-IPv6-Address-Pool

7. Acknowledgements

The authors would like to thank Bernard Aboba, Benoit Claise, Peter Deacon, Alan DeKok, Alfred Hines, Jouni Korhonen, Roberta Maglione, Leaf Yeh, Mark Smith, Pete Resnik, Ralph Droms, Stephen Farrell, Brian Haberman, for their help and comments in reviewing this document.

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.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

8.2. Informative References

[RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.

[RFC3315] Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, December 2003.

[RFC3646] Droms, R., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, December 2003.

[RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, November 2005.

[RFC4818] Salowey, J. and R. Droms, "RADIUS Delegated-IPv6-Prefix Attribute", RFC 4818, April 2007.

[RFC6106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 6106, November 2010.

[RFC6158] DeKok, A. and G. Weber, "RADIUS Design Guidelines", BCP 158, RFC 6158, March 2011.

[RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.

[SECI] -, "http://regul.uni-mb.si/~meolic/ptk-seminarske/radius.pdf", November 2001.

Authors' Addresses

Wojciech Dec (editor)
Cisco Systems, Inc.
Haarlerbergweg 13-19
Amsterdam , NOORD-HOLLAND 1101 CH
Netherlands

Email: wdec@cisco.com

Behcet Sarikaya
Huawei USA
1700 Alma Dr. Suite 500
Plano, TX
US

Phone: +1 972-509-5599
Email: sarikaya@ieee.org

Glen Zorn
Network Zen
1310 East Thomas Street
Seattle, WA
US

Email: gwz@net-zen.net

David Miles
Google

Phone:
Fax:
Email: david.miles@google.com
URI:

Benoit Lourdelet
Juniper Networks
France

Email: blourdel@juniper.net

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/1id-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

Table of Contents

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.

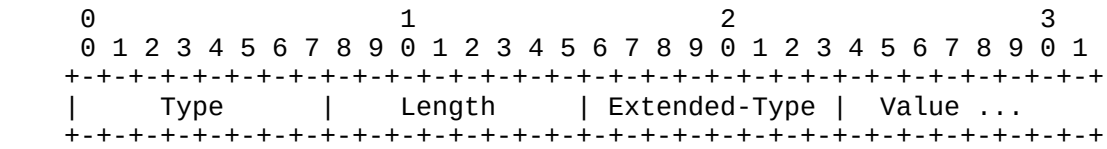
The changes to the attribute format come about by "stealing" one or more octets from the Value field. This change has the effect that the Value field of [RFC2865] Section 5 contains both the new octets given here, and any attribute-specific Value. The result is that Values in this specification are limited 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.



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.

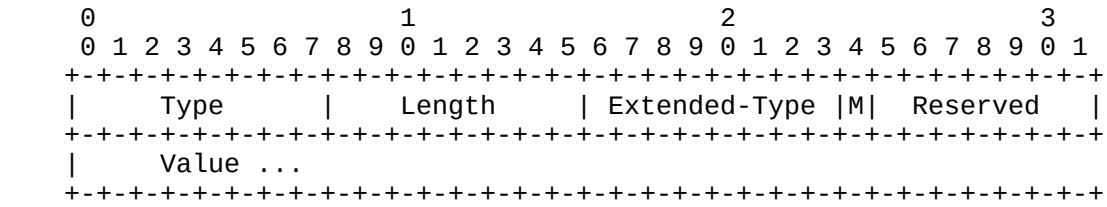
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

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.



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:

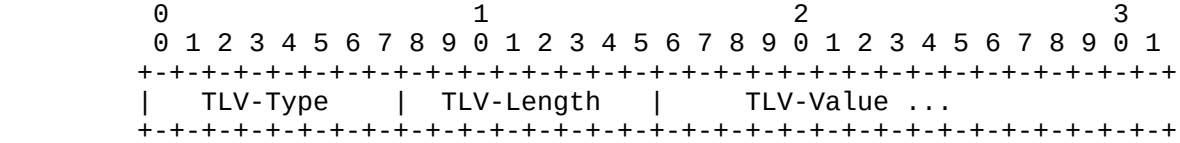
If multiple Attributes with the same Type are present, the order of Attributes with the same Type MUST be preserved by any proxies. The order of Attributes of different Types is not required to be preserved. A RADIUS server or client MUST NOT have any dependencies on the order of attributes of different 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:



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.

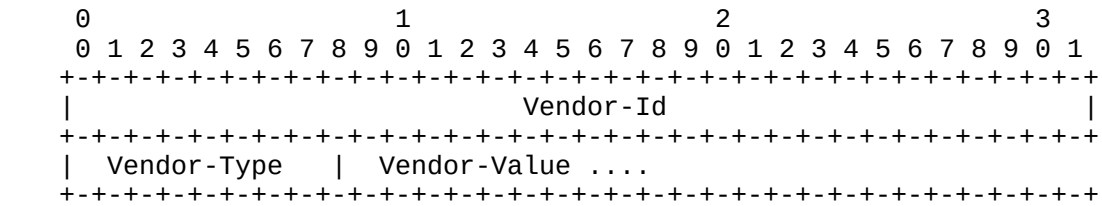
This data type is intended use in attributes which carry Vendor-Specific information, as is done with the Vendor-Specific Attribute (26). It is RECOMMENDED that this data type be used by a vendor only when the Vendor-Specific Attribute Type space has been fully allocated.

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.



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.

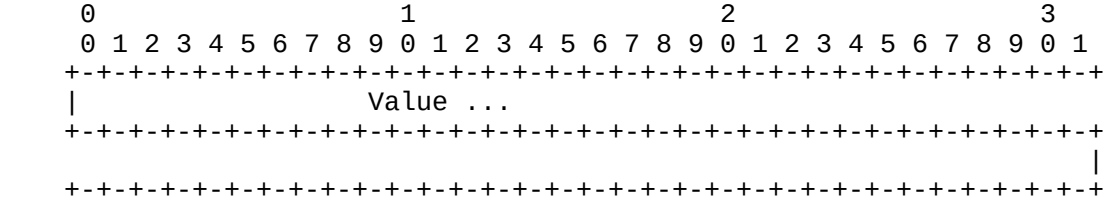
2.5. Integer64 Data Type

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.



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

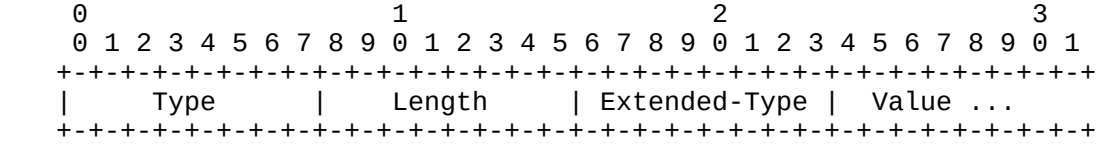
The rest of this section gives a detailed definition for each Attribute based on the above summary.

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.



Type

241 for Extended-Type-1.

Length

>= 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.

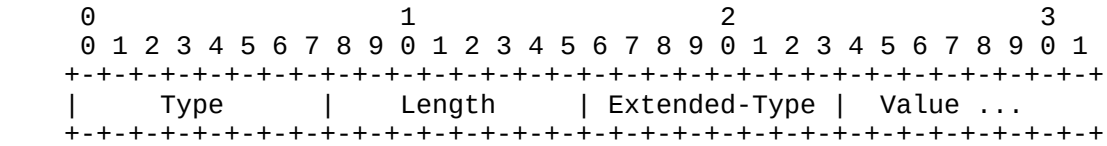
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

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.



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.

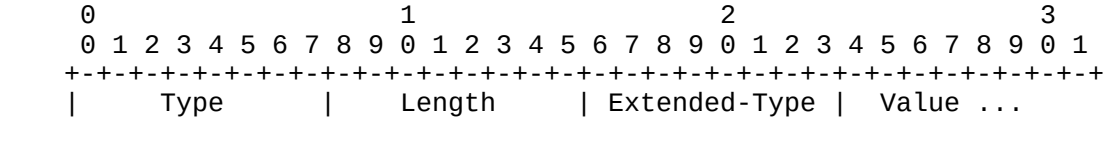
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field

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.



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

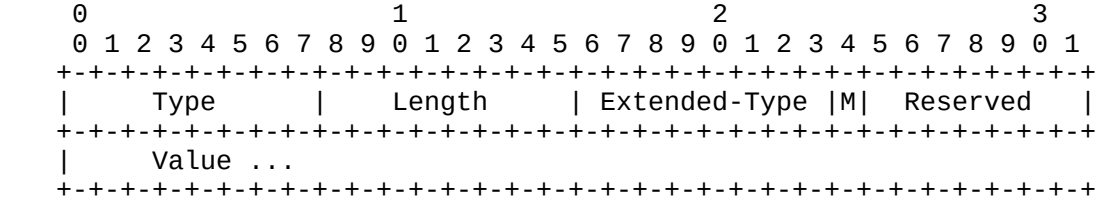
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value Field.

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.



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.

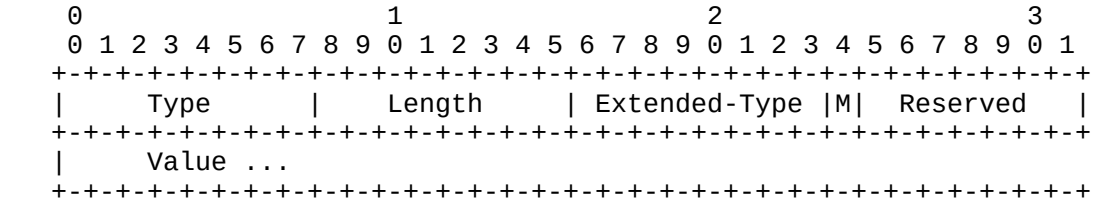
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the Value field.

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.



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

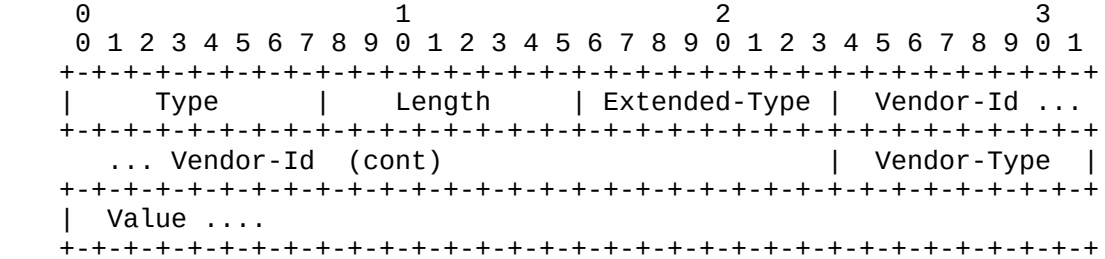
The rest of this section gives a detailed definition for each Attribute based on the above summary.

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.



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.

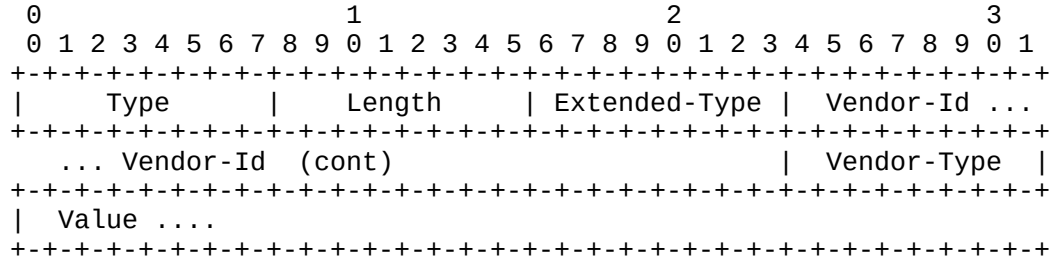
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

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.



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.

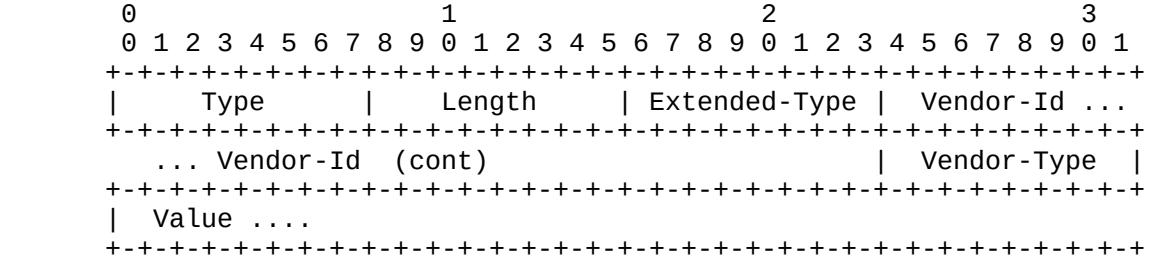
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

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.



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.

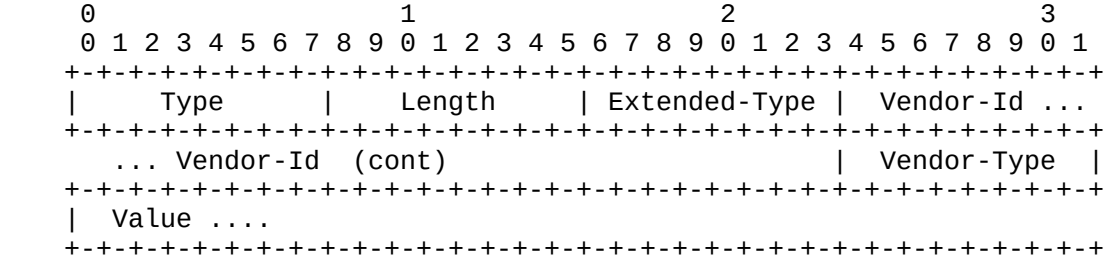
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

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.



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.

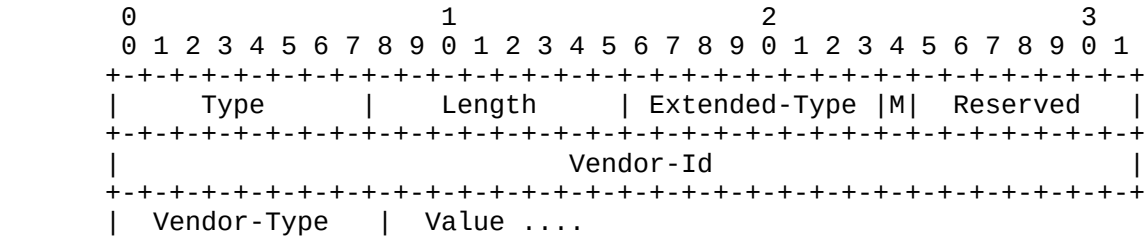
Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

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.



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

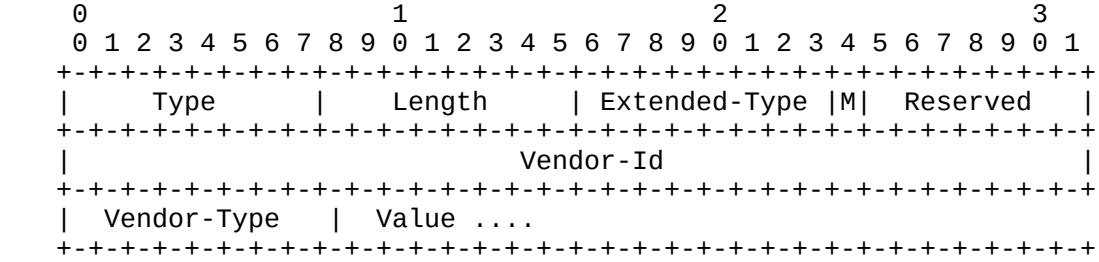
Identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the Value field.

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.



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.

```
  245.2 { 1 23 45 } { 2 67 89 }
    -> f5 0c 02 00 01 04 23 45 02 04 67 89
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating a TLV.

```
  245.2 { 1 23 45 } { 3 { 1 ab cd } }
    -> f5 0e 02 00 01 04 23 45 03 06 01 04 ab cd
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating two TLVs.

```
  245.2 { 1 23 45 } { 3 { 1 ab cd } { 2 "foo" } }
    -> f5 13 02 00 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.

```
  245.1 { 1 { 2 { 3 { 4 { 5 cd ef } } } } }
    -> f5 10 01 00 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.

```
  245.26.1.4 "test"
    -> f5 0d 1a 00 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.

```
  245.26.1.5 { 3 "test" }
    -> f5 0f 1a 00 00 00 00 01 05 03 06 74 65 73 74
```

Attribute encapsulating more than 251 octets of data. The "Data" portions are indented for readability.

```
  245.4 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbccccccccccccccccccc
```

```
              ccccccccccc"
          -> 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 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 f5 13 04 00 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 ad Vendor-Type fields.

The "Data" portions are indented for readability.

```
        245.26.1.6  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
              bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
              bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
              bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
              bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccccccccccc
              ccccccccccccccccc"
          -> 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 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 f5 18 1a 00 bb
bb bb bb bb cc 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: August 17, 2012

S. Winter
RESTENA
M. McCauley
OSC
S. Venaas
K. Wierenga
Cisco

February 14, 2012

Transport Layer Security (TLS) encryption for RADIUS
draft-ietf-radext-radsec-12

Abstract

This document specifies a transport profile for RADIUS using Transport Layer Security (TLS) over TCP as the transport protocol. This enables dynamic trust relationships between RADIUS 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 August 17, 2012.

Copyright Notice

Table of Contents

1. Introduction

The RADIUS protocol [RFC2865] is a widely deployed authentication and authorisation protocol. The supplementary RADIUS Accounting specification [RFC2866] also provides accounting mechanisms, thus delivering a full Authentication, Authorization, and Accounting (AAA) solution. However, RADIUS is experiencing several shortcomings, such as its dependency on the unreliable transport protocol UDP and the lack of security for large parts of its packet payload. RADIUS security is based on the MD5 algorithm, which has been proven to be insecure.

The main focus of RADIUS over TLS is to provide a means to secure the communication between RADIUS/TCP peers using TLS. The most important use of this specification lies in roaming environments where RADIUS packets need to be transferred through different administrative domains and untrusted, potentially hostile networks. An example for a world-wide roaming environment that uses RADIUS over TLS to secure communication is "eduroam", see [eduroam].

There are multiple known attacks on the MD5 algorithm which is used in RADIUS to provide integrity protection and a limited confidentiality protection (see [MD5-attacks]). RADIUS over TLS wraps the entire RADIUS packet payload into a TLS stream and thus mitigates the risk of attacks on MD5.

Because of the static trust establishment between RADIUS peers (IP address and shared secret) the only scalable way of creating a massive deployment of RADIUS-servers under control by different administrative entities is to introduce some form of a proxy chain to route the access requests to their home server. This creates a lot of overhead in terms of possible points of failure, longer transmission times as well as middleboxes through which authentication traffic flows. These middleboxes may learn privacy-relevant data while forwarding requests. The new features in RADIUS over TLS obsolete the use of IP addresses and shared MD5 secrets to identify other peers and thus allow the use of more contemporary trust models, e.g. checking a certificate by inspecting the issuer and other certificate properties.

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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

1.2. Terminology

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

RADIUS/TLS Client: a RADIUS over TLS instance which initiates a new connection.

RADIUS/TLS Server: a RADIUS over TLS instance which listens on a RADIUS over TLS port and accepts new connections

RADIUS/UDP: classic RADIUS transport over UDP as defined in [RFC2865]

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 (see also Section 4). The specification does not fulfill all recommendations on a AAA transport profile as per [RFC3539]; in particular, by being based on TCP as a transport layer, it does not prevent head-of-line blocking issues.

If this specification is indeed selected for advancement to standards track, certificate verification options (section 2.3.2) need to be refined.

Another experimental characteristic of this specification is the question of key management between RADIUS/TLS peers. RADIUS/UDP only allowed for manual key management, i.e. distribution of a shared secret between a client and a server. RADIUS/TLS allows manual distribution of long-term proofs of peer identity as well (by using TLS-PSK cipher suites, or identifying clients by a certificate fingerprint), but as a new feature enables use of X.509 certificates in a PKIX infrastructure. It remains to be seen if one of these methods prevail, or if both will find their place in real-life deployments. The authors can imagine pre-shared keys to be popular in small-scale deployments (SOHO or isolated enterprise deployments) where scalability is not an issue and the deployment of a CA is considered too much a hassle; but can also imagine large roaming consortia to make use of PKIX. Readers of this specification are encouraged to read the discussion of key management issus 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 at large scale is by observing the uptake, usability and operational behaviour of the protocol in large-scale, real-life deployments.

An example for a world-wide roaming environment that uses RADIUS over TLS to secure communication is "eduroam", see [eduroam].

2. Normative: Transport Layer Security for RADIUS/TCP

2.1. TCP port and packet types

The default destination port number for RADIUS over TLS is TCP/2083. There are no separate ports for authentication, accounting and dynamic authorisation changes. The source port is arbitrary. See section Section 3.4 for considerations regarding separation of authentication, accounting and dynamic authorization traffic.

2.2. TLS negotiation

RADIUS/TLS has no notion of negotiating TLS in an established connection. Servers and clients need to be preconfigured to use RADIUS/TLS for a given endpoint.

2.3. Connection Setup

RADIUS/TLS nodes

1. establish TCP connections as per [I-D.ietf-radext-tcp-transport]. Failure to connect leads to continuous retries, with exponentially growing intervals between every try. If multiple servers are defined, the node MAY attempt to establish a connection to these other servers in parallel, in order to implement quick failover.

2. after completing the TCP handshake, immediately negotiate TLS sessions according to [RFC5246] or its predecessor TLS 1.1. The following restrictions apply:

   * Support for TLS v1.1 [RFC4346] or later (e.g. TLS 1.2 [RFC5246]]) is REQUIRED. To prevent known attacks on TLS versions prior to 1.1, implementations MUST NOT negotiate TLS versions prior to 1.1.

   * Support for certificate-based mutual authentication is REQUIRED.

   * Negotiation of mutual authentication is REQUIRED.

   * Negotiation of a ciphersuite providing for confidentiality as well as integrity protection is REQUIRED. Failure to comply with this requirement can lead to severe security problmes, like user passwords being recoverable by third parties. See

Section 6 for details.

* Support for and negotiation of compression is OPTIONAL.

* Support for TLS-PSK mutual authentication [RFC4279] is OPTIONAL.

* RADIUS/TLS implementations MUST at a minimum support negotiation of the TLS_RSA_WITH_3DES_EDE_CBC_SHA), and SHOULD support TLS_RSA_WITH_RC4_128_SHA and TLS_RSA_WITH_AES_128_CBC_SHA as well (see Section 3.3).

* In addition, RADIUS/TLS implementations MUST support negotiation of the mandatory-to-implement ciphersuites required by the versions of TLS that they support.

3. Peer authentication can be performed in any of the following three operation models:

   * TLS with X.509 certificates using PKIX trust models (this model is mandatory to implement):

     + Implementations MUST allow to configure a list of trusted Certification Authorities for incoming connections.

     + Certificate validation MUST include the verification rules as per [RFC5280].

     + Implementations SHOULD indicate their trusted Certification Authorities (CAs). For TLS 1.2, this is done using [RFC5246] section 7.4.4 "certificate authorities" (server side) and [RFC6066] Section 6 "Trusted CA Indication" (client side). See also Section 3.2.

     + Peer validation always includes a check on whether the locally configured expected DNS name or IP address of the server that is contacted matches its presented certificate. DNS names and IP addresses can be contained in the Common Name (CN) or subjectAltName entries. For verification, only one of these entries is to be considered. The following precedence applies: for DNS name validation, subjectAltName:DNS has precedence over CN; for IP address validation, subjectAltName:iPAddr has precedence over CN. Implementors of this specification are advised to read [RFC6125] Section 6 for more details on DNS name validation.

+ Implementations MAY allow to configure a set of additional properties of the certificate to check for a peer's authorisation to communicate (e.g. a set of allowed values in subjectAltName:URI or a set of allowed X509v3 Certificate Policies)

+ When the configured trust base changes (e.g. removal of a CA from the list of trusted CAs; issuance of a new CRL for a given CA) implementations MAY re-negotiate the TLS session to re-assess the connecting peer's continued authorisation.

* TLS with X.509 certificates using certificate fingerprints (this model is optional to implement): Implementations SHOULD allow to configure a list of trusted certificates, identified via fingerprint of the DER encoded certificate octets. Implementations MUST support SHA-1 as the hash algorithm for the fingerprint. To prevent attacks based on hash collisions, support for a more contemporary hash function such as SHA-256 is RECOMMENDED.

* TLS using TLS-PSK (this model is optional to implement)

4. start exchanging RADIUS datagrams (note Section 3.4 (1)). The shared secret to compute the (obsolete) MD5 integrity checks and attribute encryption MUST be "radsec" (see Section 3.4 (2)).

2.4. Connecting Client Identity

In RADIUS/UDP, clients are uniquely identified by their IP address. Since the shared secret is associated with the origin IP address, if more than one RADIUS client is associated with the same IP address, then those clients also must utilize the same shared secret, a practice which is inherently insecure as noted in [RFC5247].

RADIUS/TLS supports multiple operation modes.

In TLS-PSK operation, a client is uniquely identified by its TLS identifier.

In TLS-X.509 mode using fingerprints, a client is uniquely identified by the fingerprint of the presented client certificate.

In TLS-X.509 mode using PKIX trust models, a client is uniquely identified by the tuple (serial number of presented client certificate;Issuer).

Note well: having identified a connecting entity does not mean the

server necessarily wants to communicate with that client. E.g. if the Issuer is not in a trusted set of Issuers, the server may decline to perform RADIUS transactions with this client.

There are numerous trust models in PKIX environments, and it is beyond the scope of this document to define how a particular deployment determines whether a client is trustworthy. Implementations which want to support a wide variety of trust models should expose as many details of the presented certificate to the administrator as possible so that the trust model can be implemented by the administrator. As a suggestion, at least the following parameters of the X.509 client certificate should be exposed:

- Originating IP address
- Certificate Fingerprint
- Issuer
- Subject
- all X509v3 Extended Key Usage
- all X509v3 Subject Alternative Name
- all X509v3 Certificate Policies

In TLS-PSK operation, at least the following parameters of the TLS connection should be exposed:

- Originating IP address
- TLS Identifier

2.5. RADIUS Datagrams

Authentication, Accounting and Authorization packets are sent according to the following rules:

RADIUS/TLS clients transmit the same packet types on the connection they initiated as a RADIUS/UDP client would (see Section 3.4 (3) and (4)). E.g. they send

- Access-Request
- Accounting-Request

- Status-Server
- Disconnect-ACK
- Disconnect-NAK
- ...

and they receive

- Access-Accept
- Accounting-Response
- Disconnect-Request
- ...

RADIUS/TLS servers transmit the same packet types on connections they have accepted as a RADIUS/UDP server would. E.g. they send

- Access-Challenge
- Access-Accept
- Access-Reject
- Accounting-Response
- Disconnect-Request
- ...

and they receive

- Access-Request
- Accounting-Request
- Status-Server
- Disconnect-ACK
- ...

Due to the use of one single TCP port for all packet types, it is required for a RADIUS/TLS server to signal to a connecting peer which types of packets are supported on a server. See also section

Section 3.4 for a discussion of signaling.

- When receiving an unwanted packet of type 'CoA-Request' or 'Disconnect-Request', it needs to be replied to with a 'CoA-NAK' or 'Disconnect-NAK' respectively. The NAK SHOULD contain an attribute Error-Cause with the value 406 ("Unsupported Extension"); see [RFC5176] for details.

- When receiving an unwanted packet of type 'Accounting-Request', the RADIUS/TLS server SHOULD reply with an Accounting-Response containing an Error-Cause attribute with value 406 "Unsupported Extension" as defined in [RFC5176]. A RADIUS/TLS accounting client receiving such an Accounting-Response SHOULD log the error and stop sending Accounting-Request packets.

3. Informative: Design Decisions

This section explains the design decisions that led to the rules defined in the previous section.

3.1. Implications of Dynamic Peer Discovery

One mechanism to discover RADIUS over TLS peers dynamically via DNS is specified in [I-D.ietf-radext-dynamic-discovery]. While this mechanism is still under development and therefore is not a normative dependency of RADIUS/TLS, the use of dynamic discovery has potential future implications that are important to understand.

Readers of this document who are considering the deployment of DNS-based dynamic discovery are thus encouraged to read [I-D.ietf-radext-dynamic-discovery] and follow its future development.

3.2. X.509 Certificate Considerations

(1) If a RADIUS/TLS client is in possession of multiple certificates from different CAs (i.e. is part of multiple roaming consortia) and dynamic discovery is used, the discovery mechanism possibly does not yield sufficient information to identify the consortium uniquely (e.g. DNS discovery). Subsequently, the client may not know by itself which client certificate to use for the TLS handshake. Then it is necessary for the server to signal which consortium it belongs to, and which certificates it expects. If there is no risk of confusing multiple roaming consortia, providing this information in the handshake is not crucial.

(2) If a RADIUS/TLS server is in possession of multiple certificates from different CAs (i.e. is part of multiple roaming consortia), it

will need to select one of its certificates to present to the RADIUS/TLS client. If the client sends the Trusted CA Indication, this hint can make the server select the appropriate certificate and prevent a handshake failure. Omitting this indication makes it impossible to deterministically select the right certificate in this case. If there is no risk of confusing multiple roaming consortia, providing this indication in the handshake is not crucial.

3.3. Ciphersuites and Compression Negotiation Considerations

Not all TLS ciphersuites in [RFC5246] are supported by available TLS tool kits, and licenses may be required in some cases. The existing implementations of RADIUS/TLS use OpenSSL as cryptographic backend, which supports all of the ciphersuites listed in the rules in the normative section.

The TLS ciphersuite TLS_RSA_WITH_3DES_EDE_CBC_SHA is mandatory-to-implement according to [RFC4346] and thus has to be supported by RADIUS/TLS nodes.

The two other ciphersuites in the normative section are widely implemented in TLS toolkits and are considered good practice to implement.

3.4. RADIUS Datagram Considerations

(1) After the TLS session is established, RADIUS packet payloads are exchanged over the encrypted TLS tunnel. In RADIUS/UDP, the packet size can be determined by evaluating the size of the datagram that arrived. Due to the stream nature of TCP and TLS, this does not hold true for RADIUS/TLS packet exchange. Instead, packet boundaries of RADIUS packets that arrive in the stream are calculated by evaluating the packet's Length field. Special care needs to be taken on the packet sender side that the value of the Length field is indeed correct before sending it over the TLS tunnel, because incorrect packet lengths can no longer be detected by a differing datagram boundary. See section 2.6.4 of [I-D.ietf-radext-tcp-transport] for more details.

(2) Within RADIUS/UDP [RFC2865], a shared secret is used for hiding of attributes such as User-Password, as well as in computation of the Response Authenticator. In RADIUS accounting [RFC2866], the shared secret is used in computation of both the Request Authenticator and the Response Authenticator. Since TLS provides integrity protection and encryption sufficient to substitute for RADIUS application-layer security, it is not necessary to configure a RADIUS shared secret. The use of a fixed string for the obsolete shared secret eliminates possible node misconfigurations.

(3) RADIUS/UDP [RFC2865] uses different UDP ports for authentication, accounting and dynamic authorisation changes. RADIUS/TLS allocates a single port for all RADIUS packet types. Nevertheless, in RADIUS/TLS the notion of a client which sends authentication requests and processes replies associated with it's users' sessions and the notion of a server which receives requests, processes them and sends the appropriate replies is to be preserved. The normative rules about acceptable packet types for clients and servers mirror the packet flow behaviour from RADIUS/UDP.

(4) RADIUS/UDP [RFC2865] uses negative ICMP responses to a newly allocated UDP port to signal that a peer RADIUS server does not support reception and processing of the packet types in [RFC5176]. These packet types are listed as to be received in RADIUS/TLS implementations. Note well: it is not required for an implementation to actually process these packet types; it is only required to send the NAK as defined above.

(5) RADIUS/UDP [RFC2865] uses negative ICMP responses to a newly allocated UDP port to signal that a peer RADIUS server does not support reception and processing of RADIUS Accounting packets. There is no RADIUS datagram to signal an Accounting NAK. Clients may be misconfigured to send Accounting packets to a RADIUS/TLS server which does not wish to process their Accounting packet. To prevent a regression of detectability of this situation, the Accounting-Response + Error-Cause sgnaling was introduced.

4. Compatibility with other RADIUS transports

Ongoing work in the IETF defines multiple alternative transports to the classic UDP transport model as defined in [RFC2865], namely RADIUS over TCP [I-D.ietf-radext-tcp-transport], RADIUS over Datagram Transport Layer Security (DTLS) [I-D.ietf-radext-dtls] and this present document on RADIUS over TLS.

RADIUS/TLS does not specify any inherent backwards compatibility to RADIUS/UDP or cross compatibility to the other transports, i.e. an implementation which implements RADIUS/TLS only will not be able to receive or send RADIUS packet payloads over other transports. An implementation wishing to be backward or cross compatible (i.e. wishes to serve clients using other transports than RADIUS/TLS) will need to implement these other transports along with the RADIUS/TLS transport and be prepared to send and receive on all implemented transports, which is called a multi-stack implementation.

If a given IP device is able to receive RADIUS payloads on multiple transports, this may or may not be the same instance of software, and it may or may not serve the same purposes. It is not safe to assume

that both ports are interchangeable. In particular, it can not be assumed that state is maintained for the packet payloads between the transports. Two such instances MUST be considered separate RADIUS server entities.

5. Diameter Compatibility

Since RADIUS/TLS is only a new transport profile for RADIUS, compatibility of RADIUS/TLS - Diameter [RFC3588] vs. RADIUS/UDP [RFC2865] - Diameter [RFC3588] is identical. The considerations regarding payload size in [I-D.ietf-radext-tcp-transport] apply.

6. Security Considerations

The computational resources to establish a TLS tunnel are significantly higher than simply sending mostly unencrypted UDP datagrams. Therefore, clients connecting to a RADIUS/TLS node will more easily create high load conditions and a malicious client might create a Denial-of-Service attack more easily.

Some TLS ciphersuites only provide integrity validation of their payload, and provide no encryption. This specification forbids the use of such ciphersuites. Since the RADIUS payload’s shared secret is fixed to the well-known term "radsec" (see Section 2.3 (4)) , failure to comply with this requirement will expose the entire datagram payload in plain text, including User-Password, to intermediate IP nodes.

By virtue of being based on TCP, there are several generic attack vectors to slow down or prevent the TCP connection from being established; see [RFC4953] for details. If a TCP connection is not up when a packet is to be processed, it gets re-established, so such attacks in general lead only to a minor performance degradation (the time it takes to re-establish the connection). There is one notable exception where an attacker might create a bidding-down attack though: If peer communication between two devices is configured for both RADIUS/TLS (i.e TLS security over TCP as a transport, shared secret fixed to "radsec") and RADIUS/UDP (i.e. shared secret security with a secret manually configured by the administrator), and where the RADIUS/UDP transport is the failover option if the TLS session cannot be established, a bidding-down attack can occur if an adversary can maliciously close the TCP connection, or prevent it from being established. Situations where clients are configured in such a way are likely to occur during a migration phase from RADIUS/ UDP to RADIUS/TLS. By preventing the TLS session setup, the attacker can reduce the security of the packet payload from the selected TLS cipher suite packet encryption to the classic MD5 per-attribute encryption. The situation should be avoided by disabling the weaker

RADIUS/UDP transport as soon as the new RADIUS/TLS connection is established and tested. Disabling can happen at either the RADIUS client or server side:

- Client side: de-configure the failover setup, leaving RADIUS/TLS as the only communication option
- Server side: de-configure the RADIUS/UDP client from the list of valid RADIUS clients

RADIUS/TLS provides authentication and encryption between RADIUS peers. In the presence of proxies, the intermediate proxies can still inspect the individual RADIUS packets, i.e. "end-to-end" encryption is not provided. Where intermediate proxies are untrusted, it is desirable to use other RADIUS mechanisms to prevent RADIUS packet payload from inspection by such proxies. One common method to protect passwords is the use of the Extensible Authentication Protocol (EAP) and EAP methods which utilize TLS.

When using certificate fingerprints to identify RADIUS/TLS peers, any two certificates which produce the same hash value (i.e. which have a hash collision) will be considered the same client. It is therefore important to make sure that the hash function used is cryptographically uncompromised so that an attacker is very unlikely to be able to produce a hash collision with a certificate of his choice. While this specification mandates support for SHA-1, a later revision will likely demand support for more contemporary hash functions because as of issuance of this document there are already attacks on SHA-1.

7. IANA Considerations

No new RADIUS attributes or packet codes are defined. IANA is requested to update the already-assigned TCP port number 2083 in the following ways:

- Reference: list the RFC number of this document as the reference
- Assignment Notes: add the text "The TCP port 2083 was already previously assigned by IANA for "RadSec", an early implementation of RADIUS/TLS, prior to issuance of this RFC. This early implementation can be configured to be compatible to RADIUS/TLS as specified by the IETF. See RFC (RFC number of this document), Appendix A for details."

8. Notes to the RFC Editor

[I-D.ietf-radext-tcp-transport] is currently in the publication queue because it has a normative reference on this draft; it has no other blocking dependencies. The two drafts should be published as an RFC simultaneously, ideally with consequtive numbers. The references in this draft to [I-D.ietf-radext-tcp-transport] should be changed to references to the corresponding RFC prior to publication.

This section, "Notes to the RFC Editor" should be deleted from the draft prior to publication.

9. Acknowledgements

RADIUS/TLS was first implemented as "RADSec" by Open Systems Consultants, Currumbin Waters, Australia, for their "Radiator" RADIUS server product (see [radsec-whitepaper]).

Funding and input for the development of this Internet Draft was provided by the European Commission co-funded project "GEANT2" [geant2] and further feedback was provided by the TERENA Task Force Mobility [terena].

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. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

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

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[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.

[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.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[I-D.ietf-radext-tcp-transport] DeKok, A., "RADIUS Over TCP", draft-ietf-radext-tcp-transport-09 (work in progress), October 2010.

10.2. Informative References

[I-D.ietf-radext-dtls] DeKok, A., "DTLS as a Transport Layer for RADIUS", draft-ietf-radext-dtls-01 (work in progress), October 2010.

[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-03 (work in

progress), July 2011.

[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.

[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.

[radsec-whitepaper] Open System Consultants, "RadSec - a secure, reliable RADIUS Protocol", May 2005, <http://www.open.com.au/radiator/radsec-whitepaper.pdf>.

[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/40867l85727r7084/>.

[radsecproxy-impl] Venaas, S., "radsecproxy Project Homepage", 2007, <http://software.uninett.no/radsecproxy/>.

[eduroam] Trans-European Research and Education Networking Association, "eduroam Homepage", 2007, <http://www.eduroam.org/>.

[geant2] Delivery of Advanced Network Technology to Europe, "European Commission Information Society and Media: GEANT2", 2008, <http://www.geant2.net/>.

[terena] TERENA, "Trans-European Research and Education Networking Association", 2008, <http://www.terena.org/>.

Appendix A. Implementation Overview: Radiator

Radiator implements the RadSec protocol for proxying requests with the <Authby RADSEC> and <ServerRADSEC> clauses in the Radiator configuration file.

The <AuthBy RADSEC> clause defines a RadSec client, and causes Radiator to send RADIUS requests to the configured RadSec server using the RadSec protocol.

The <ServerRADSEC> clause defines a RadSec server, and causes Radiator to listen on the configured port and address(es) for connections from <Authby RADSEC> clients. When an <Authby RADSEC> client connects to a <ServerRADSEC> server, the client sends RADIUS requests through the stream to the server. The server then handles the request in the same way as if the request had been received from a conventional UDP RADIUS client.

Radiator is compliant to RADIUS/TLS if the following options are used:

<AuthBy RADSEC>

* Protocol tcp

* UseTLS

* TLS_CertificateFile

* Secret radsec

<ServerRADSEC>

* Protocol tcp

* UseTLS

* TLS_RequireClientCert

* Secret radsec

As of Radiator 3.15, the default shared secret for RadSec connections is configurable and defaults to "mysecret" (without quotes). For compliance with this document, this setting needs to be configured for the shared secret "radsec". The implementation uses TCP keepalive socket options, but does not send Status-Server packets. Once established, TLS connections are kept open throughout the server instance lifetime.

Appendix B. Implementation Overview: radsecproxy

The RADIUS proxy named radsecproxy was written in order to allow use of RadSec in current RADIUS deployments. This is a generic proxy that supports any number and combination of clients and servers, supporting RADIUS over UDP and RadSec. The main idea is that it can be used on the same host as a non-RadSec client or server to ensure RadSec is used on the wire, however as a generic proxy it can be used in other circumstances as well.

The configuration file consists of client and server clauses, where there is one such clause for each client or server. In such a clause one specifies either "type tls" or "type udp" for RadSec or UDP transport. For RadSec the default shared secret "mysecret" (without quotes), the same as Radiator, is used. For compliance with this document, this setting needs to be configured for the shared secret "radsec". A secret may be specified by putting say "secret somesharedsecret" inside a client or server clause.

In order to use TLS for clients and/or servers, one must also specify

where to locate CA certificates, as well as certificate and key for the client or server. This is done in a TLS clause. There may be one or several TLS clauses. A client or server clause may reference a particular TLS clause, or just use a default one. One use for multiple TLS clauses may be to present one certificate to clients and another to servers.

If any RadSec (TLS) clients are configured, the proxy will at startup listen on port 2083, as assigned by IANA for the OSC RadSec implementation. An alternative port may be specified. When a client connects, the client certificate will be verified, including checking that the configured FQDN or IP address matches what is in the certificate. Requests coming from a RadSec client are treated exactly like requests from UDP clients.

The proxy will at startup try to establish a TLS connection to each (if any) of the configured RadSec (TLS) servers. If it fails to connect to a server, it will retry regularly. There is some back-off where it will retry quickly at first, and with longer intervals later. If a connection to a server goes down it will also start retrying regularly. When setting up the TLS connection, the server certificate will be verified, including checking that the configured FQDN or IP address matches what is in the certificate. Requests are sent to a RadSec server just like they would to a UDP server.

The proxy supports Status-Server messages. They are only sent to a server if enabled for that particular server. Status-Server requests are always responded to.

This RadSec implementation has been successfully tested together with Radiator. It is a freely available open-source implementation. For source code and documentation, see [radsecproxy-impl].

Appendix C. Assessment of Crypto-Agility Requirements

The RADIUS Crypto-Agility Requirements [RFC6421] defines numerous classification criteria for protocols that strive to enhance the security of RADIUS. It contains mandatory (M) and recommended (R) criteria which crypto-agile protocols have to fulfill. The authors believe that the following assessment about the crypto-agility properties of RADIUS/TLS are true.

By virtue of being a transport profile using TLS over TCP as a transport protocol, the cryptographically agile properties of TLS are inherited, and RADIUS/TLS subsequently meets the following points:

(M) negotiation of cryptographic algorithms for integrity and auth

(M) negotiation of cryptographic algorithms for encryption

(M) replay protection

(M) define mandatory-to-implement cryptographic algorithms

(M) generate fresh session keys for use between client and server

(R) support for Perfect Forward Secrecy in session keys

(R) support X.509 certificate based operation

(R) support Pre-Shared keys

(R) support for confidentiality of the entire packet

(M/R) support Automated Key Management

The remainder of the requirements is discussed individually below in more detail:

(M) "avoid security compromise, even in situations where the existing cryptographic alogrithms used by RADIUS implementations are shown to be weak enough to provide little or no security" - The existing algorithm, based on MD5, is not of any significance in RADIUS/TLS; its compromise does not compromise the outer transport security.

(R) mandatory-to-implement alogrithms are to be NIST-Acceptable with no deprecation date - The mandatory-to-implement algorithm is TLS_RSA_WITH_3DES_EDE_CBC_SHA. This ciphersuite supports three-key 3DES operation, which is classified as Acceptable with no known deprecation date by NIST.

(M) demonstrate backward compatibility with RADIUS - There are multiple implementations supporting both RADIUS and RADIUS/TLS, and the translation between them.

(M) After legacy mechanisms have been compromised, secure algorithms MUST be used, so that backward compatibility is no longer possible - In RADIUS, communication between client and server is always a manual configuration; after a compromise, the legacy client in question can be de-configured by the same manual configuration.

(M) indicate a willingness to cede change control to the IETF - Change control of this protocol is with the IETF.

(M) be interoperable between implementations based purely on the information in the specification - At least one implementation was created exclusively based on this specification and is interoperable with other RADIUS/TLS implementations.

(M) apply to all packet types - RADIUS/TLS operates on the transport layer, and can carry all packet types.

(R) message data exchanged with Diameter SHOULD NOT be affected - The solution is Diameter-agnostic.

(M) discuss any inherent assumptions - The authors are not aware of any implicit assumptions which would be yet-unarticulated in the draft

(R) provide recommendations for transition - The Security Considerations section contains a transition path.

(R) discuss legacy interoperability and potential for bidding-down attacks - The Security Considerations section contains an corresponding discussion.

Summarizing, it is believed that this specification fulfills all the mandatory and all the recommended requirements for a crypto-agile solution and should thus be considered UNCONDITIONALLY COMPLIANT.

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
Open Systems Consultants
9 Bulbul Place
Currumbin Waters QLD 4223
AUSTRALIA

Phone: +61 7 5598 7474
Fax: +61 7 5598 7070
EMail: mikem@open.com.au
URI: http://www.open.com.au.

Stig Venaas
cisco Systems
Tasman Drive
San Jose, CA 95134
USA

EMail: stig@cisco.com

Klaas Wierenga
Cisco Systems International BV
Haarlerbergweg 13-19
Amsterdam 1101 CH
The Netherlands

Phone: +31 (0)20 3571752
Fax:
EMail: kwiereng@cisco.com
URI: http://www.cisco.com.

RADIUS Extensions Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 17, 2013

S. Winter
RESTENA

July 16, 2012

RADIUS Accounting for traffic classes
draft-winter-radext-fancyaccounting-02

Abstract

This document specifies new attributes for RADIUS Accounting to enable NAS reporting of subsets of the total traffic in a user session.

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 17, 2013.

Copyright Notice

Table of Contents

1. Introduction

RADIUS Accounting [RFC2866] defines counters for octets and packets, both in the incoming and outgoing direction. Usage of these counters enables an operator create volume-based billing models and to execute proper capacity planning on its infrastructure.

The Accounting model is based on the assumption that all traffic in a user session is treated equally; i.e. that there are no differences in the billing model of one class of traffic over another.

Actual deployments suggest that this assumption is no longer valid. In particular, different traffic classes are defined with DSCP; and billing the use of these traffic classes separately is an understandable request.

Plus, the introduction of dual-stack operation on links creates an understandable interest of getting separate statistics about the amount of IPv4 vs. IPv6 usage on a link; be it for billing or statistical reasons.

This document defines Accounting attributes that supplement (but not replace) the accounting counters in RFC2866. It utilizes the new "extended attributes" in RADIUS ([I-D.ietf-radext-radius-extensions]) to a) group accounting reports about traffic classes together and b) enable 64-Bit counts in a single attribute with the Integer64 datatype.

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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

2. Definitions

2.1. Acct-Traffic-Class attribute

The attribute Acct-Traffic-Class is a TLV container for a group of sub-attributes which specify the class of traffic that is being reported about, and the amount of traffic in a user session that falls into this class.

Attribute: 245.1 Acct-Traffic-Class

Type: TLV

Length: >3 octets

There can be multiple instances of this attribute in a Accounting-Interim-Update or a Accounting-Stop packet. The attribute MUST NOT be present in an Accounting-Start packet.

It is not required that the sum of all traffic in all instances is the total sum of octets and packets in the user's session. I.e. the traffic classes used in the Accounting packet do not need to partition the total traffic in non-overlapping segments.

The total number of octets and packets in a user session continues to be sent in the RFC2866 attributes.

2.1.1. Acct-Traffic-Class-Name attribute

The attribute Acct-Traffic-Class-Name, sub-attribute in the group Acct-Traffic-Class, defines the class of traffic for which the other attributes in the instance of Acct-Traffic-Class count octets and packets. Every group instance MUST contain exactly one Acct-Traffic-Class-Name.

Attribute: 245.1.2 Acct-Traffic-Class-Name

Type: STRING

Value: 1-250 octets

There are two options for the value of this attribute.

Option 1: Acct-Traffic-Class-Name string starting with the substring "urn:". Usage of this option implies that the traffic name is in the form of a URN and requires that a public specification of this URN exists. That specification must include the type of traffic being counted with this traffic class, and the exact definition of where in the network packets the byte-count starts and ends. This document defines a set of known, well-defined traffic accounting classes in an IANA-managed registry in Section 2.2. New values for this registry are assigned on expert review basis.

Option 2: Acct-Traffic-Class-Name string not starting with "urn:". This option is for local use of special-purpose accounting as defined by the NAS administrator, where no defined URN matches the meaning of the traffic to be counted. The meaning of the content needs to be communicated out-of-band between the NAS and RADIUS Server operator. Example: Acct-Traffic-Class-Name = "UDP traffic to AS2606".

2.1.2. Acct-Traffic-Class-Input-Octets attribute

The attribute Acct-Traffic-Class-Input-Octets, sub-attribute in the group Acct-Traffic-Class, carries the number of octets that belong to the class of traffic indicated by Acct-Traffic-Class-Name and have been sent to the entity for which the accounting packet was generated. It MUST occur at most once inside every instance of the Acct-Traffic-Class TLV. If a traffic parameter value is transmitted in this attribute in an Accouting-Request "Interim Update", then the final value of that traffic parameter MUST be reported in the corresponding Accounting-Request "Stop".

Attribute: 245.1.3 Acct-Traffic-Class-Input-Octets

Type: Integer64

Value: number of octets sent to entity, matching the class of traffic

2.1.3. Acct-Traffic-Class-Output-Octets attribute

The attribute Acct-Traffic-Class-Output-Octets, sub-attribute in the group Acct-Traffic-Class, carries the number of octets that belong to the class of traffic indicated by Acct-Traffic-Class-Name and have been sent from the entity for which the accounting packet was generated. It MUST occur at most once inside every instance of the Acct-Traffic-Class TLV. If a traffic parameter value is transmitted in this attribute in an Accouting-Request "Interim Update", then the final value of that traffic parameter MUST be reported in the corresponding Accounting-Request "Stop".

Attribute: 245.1.4 Acct-Traffic-Class-Output-Octets

Type: Integer64

Value: number of octets sent from entity, matching the class of traffic

2.1.4. Acct-Traffic-Class-Input-Packets attribute

The attribute Acct-Traffic-Class-Input-Packets, sub-attribute in the group Acct-Traffic-Class, carries the number of packets that belong to the class of traffic indicated by Acct-Traffic-Class-Name and have been sent to the entity for which the accounting packet was generated. It MUST occur at most once inside every instance of the Acct-Traffic-Class TLV. If a traffic parameter value is transmitted in this attribute in an Accouting-Request "Interim Update", then the final value of that traffic parameter MUST be reported in the

corresponding Accounting-Request "Stop".

Attribute: 245.1.5 Acct-Traffic-Class-Input-Packets

Type: Integer64

Value: number of packets sent to entity, matching the class of traffic

2.1.5. Acct-Traffic-Class-Output-Packets attribute

The attribute Acct-Traffic-Class-Output-Packets, sub-attribute in the group Acct-Traffic-Class, carries the number of packets that belong to the class of traffic indicated by Acct-Traffic-Class-Name and have been sent from the entity for which the accounting packet was generated. It MUST occur at most once inside every instance of the Acct-Traffic-Class TLV. If a traffic parameter value is transmitted in this attribute in an Accouting-Request "Interim Update", then the final value of that traffic parameter MUST be reported in the corresponding Accounting-Request "Stop".

Attribute: 245.1.6 Acct-Traffic-Class-Output-Packets

Type: Integer64

Value: number of packets sent from entity, matching the class of traffic

2.2. URN values for attribute Acct-Traffic-Class-Name

The following URN values are defined for RADIUS Accounting Traffic Classes:

Name: "urn:ietf:radius-accounting:ip:4"

Purpose: volume count of IPv4 payloads

Start of byte count: 1st byte of the IP header of the packet

End of byte count: last byte of IP layer of the packet

Name: "urn:ietf:radius-accounting:ip:6"

Purpose: volume count of IPv6 payloads

Start of byte count: 1st byte of the IP header of the packet

End of byte count: last byte of IP layer of the packet

Name: "urn:ietf:radius-accounting:dscp:0"

Purpose: volume count of packet payloads with DSCP = 0

Start of byte count: 1st byte of the IP header of the packet

End of byte count: last byte of IP layer of the packet

Name: "urn:ietf:radius-accounting:tcp"

Purpose: volume count of TCP packets

Start of byte count: 1st byte of the TCP header of the packet

End of byte count: last byte of TCP layer of the packet

Name: "urn:ietf:radius-accounting:udp"

Purpose: volume count of UDP payloads

Start of byte count: 1st byte of the UDP header of the packet

End of byte count: last byte of UDP layer of the packet

(more values to be added...)

3. Example

A NAS is configured to create statistics regarding IPv6 usage of CPE for statistical reasons, and of the amount of HTTP traffic sent to the example.com web site for billing reasons.

User john@example.com starts a user session, transfers 1200 Bytes in 10 packets via IPv6 to the internet, and receives 4500 Bytes in 30 packets over IPv6 from the internet.

In the same session, The user visits the IPv4-only example.com web site by sending 6000 bytes in 4 packets to the web site, and receiving 450000 Bytes in 35 packets from the web site.

Then, the user terminates the session and an Accounting-Stop packet is generated.

The NAS sends the recorded octet and packet values to his RADIUS Accounting server. Since there is no URN value for "Traffic on TCP/80 to example.com, all IP versions" for use in the Acct-Traffic-

Class-Name attribute, the NAS has been configured to indicate this class of traffic in a corresponding custom string. The relevant attributes in the Accounting-Stop packet are:

Acct-Traffic-Class

Acct-Traffic-Class-Name = "urn:ietf:radius-accunting:ip:6"

Acct-Traffic-Class-Input-Octets = 4500

Acct-Traffic-Class-Output-Octets = 1200

Acct-Traffic-Class-Input-Packets = 30

Acct-Traffic-Class-Output-Packets = 10

Acct-Traffic-Class

Acct-Traffic-Class-Name = "Traffic on TCP/80 to example.com, all IP versions"

Acct-Traffic-Class-Input-Octets = 450000

Acct-Traffic-Class-Output-Octets = 6000

Acct-Traffic-Class-Input-Packets = 35

Acct-Traffic-Class-Output-Packets = 4

4. Attribute Occurence Table

This table lists the allowed occurences of the previously defined attributes in Accounting packets.

Start	Interim	Stop	Reply	Attribute
0	0-n	0-s	0	Acct-Traffic-Class
0	0-m	0-t	0	Acct-Traffic-Class-Name
0	0-o	0-u	0	Acct-Traffic-Class-Input-Octets
0	0-p	0-v	0	Acct-Traffic-Class-Output-Octets
0	0-q	0-w	0	Acct-Traffic-Class-Input-Packets
0	0-r	0-x	0	Acct-Traffic-Class-Output-Packets

Figure 1: Attribute Occurence

Note 1: since all sub-attributes occur at most once inside any given Acct-Traffic-Class TLV, the sub-attributes can not occur more often than the TLV itself. I.e. m<n, o<n, p<n, q<n, and r<n.

Note 2: if Acct-Traffic-Class TLVs and their sub-attributes have been sent in Interim-Updates, they MUST also occur in the subsequent Stop packet; while the Stop packet MAY contain additional Acct-Traffic-Class instances. I.e. in the table above: s>=n, t>=m, u>=o, v>=p,w>=q, and x>=r.

5. Security Considerations

Reveals user's traffic usage patterns. Shouldn't be sent unencrpyptedly.

6. IANA Considerations

This document has actions for IANA. TBD later.

7. Normative References

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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-06 (work in progress), June 2012.

Author's Address

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.