

Network Working Group
INTERNET-DRAFT
Category: Proposed Standard
Expires: April 23, 2012
Updates: 4072

Bernard Aboba
Microsoft Corporation
Jouni Malinen
Devicescape Software
Paul Congdon
Hewlett Packard Company
Joseph Salowey
Cisco Systems
22 October 2011

RADIUS Attributes for IEEE 802 Networks
draft-aboba-radext-wlan-15.txt

Abstract

RFC 3580 provides guidelines for the use of the Remote Authentication Dialin User Service (RADIUS) within IEEE 802 local area networks (LANs). This document proposes additional attributes for use within IEEE 802 networks, as well as providing clarifications on the usage of the EAP-Key-Name attribute, updating RFC 4072. The attributes defined in this document are usable both within RADIUS and Diameter.

Status of This Memo

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

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

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

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

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

This Internet-Draft will expire on April 23, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1.	Introduction	4
1.1	Terminology	4
1.2	Requirements Language	5
2.	RADIUS attributes	5
2.1	Allowed-Called-Station-Id	5
2.2	EAP-Key-Name	7
2.3	EAP-Peer-Id	8
2.4	EAP-Server-Id	9
2.5	Mobility-Domain-Id	10
2.6	Preauth-Timeout	10
2.7	Network-Id-Name	11
2.8	Access-Info	12
3.	Table of attributes	13
4.	Diameter Considerations	14
5.	IANA Considerations	15
6.	Security Considerations	15
7.	References	15
7.1	Normative References	15
7.2	Informative References	16
	ACKNOWLEDGMENTS	17
	AUTHORS' ADDRESSES	18

1. Introduction

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

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

1.1. Terminology

This document uses the following terms:

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

1.2. Requirements Language

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

2. RADIUS attributes

2.1. Allowed-Called-Station-Id

Description

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

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

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

Code

TBD1

Length

>=3

String

The String field is one or more octets, containing the layer 2 endpoint that the user's call is allowed to be terminated on, as specified in the definition of Called-Station-Id in [RFC2865] Section 5.30 and [RFC3580] Section 3.20. In the case of IEEE 802, the Allowed-Called-Station-Id Attribute is used to store the Medium Access Control (MAC) address in ASCII format (upper case only), with octet values separated by a "-". Example: "00-10-A4-23-19-C0". Where restrictions on both the network and authenticator MAC address usage are intended, the network name

MUST be appended to the authenticator MAC address, separated from the MAC address with a ":". Example: "00-10-A4-23-19-C0:AP1". Where no MAC address restriction is intended, the MAC address field MUST be omitted, but the network name field MUST be included. Example: "AP1". Within IEEE 802.11 [IEEE-802.11], the SSID constitutes the network name; within IEEE 802.1X [IEEE-802.1X], the Network-Id Name (NID-Name) constitutes the network name. Since a NID-Name can be up to 253 octets in length, when used with [IEEE-802.1X], there may not be sufficient room within the Allowed-Called-Station-Id Attribute to include a MAC address.

If the user attempts to connect to the NAS from a Called-Station-Id that does not match one of the Allowed-Called-Station-Id attributes, then the user MUST NOT be permitted to access the network.

The Allowed-Called-Station-Id Attribute can be useful in the following situations:

- [1] Where users can connect to a NAS without an Access-Request being sent by the NAS to the RADIUS server (e.g. where key caching is supported within IEEE 802.11 or IEEE 802.1X [IEEE-802.1X]). To avoid elevation of privilege attacks, key cache entries are typically only usable within the network to which the user originally authenticated (e.g. the originally selected network name is implicitly attached to the key cache entry). Also, if it is desired that access to a network name not be available from a particular authenticator MAC address, then the authenticator can be set up not to advertise that particular network name.
- [2] Where pre-authentication may be supported (e.g. IEEE 802.1X pre-authentication). In this situation, the network name typically will not be included in a Called-Station-Id Attribute within the Access-Request, so that the RADIUS server will not know the network that the user is attempting to access. As a result, the RADIUS server may desire to restrict the networks to which the user can subsequently connect.
- [3] Where the network portion of the Called-Station-Id is present within an Access-Request, the RADIUS server can desire to authorize access to a network different from the one that the user selected.

2.2. EAP-Key-Name

Description

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

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

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

Code

102 [RFC4072]

Length

>=3

String

The String field is one or more octets, containing the EAP Session-Id, as defined in "Extensible Authentication Protocol (EAP) Key Management Framework" [RFC5247]. Since the NAS operates as a pass-through in EAP, it cannot know the EAP Session-Id before receiving it from the RADIUS server. As a result, an EAP-Key-Name Attribute sent in an Access-Request MUST only contain a single NUL character. A RADIUS server receiving an Access-Request with an EAP-Key-Name Attribute containing anything other than a single NUL character MUST silently discard the Attribute. In addition, the RADIUS server SHOULD include this Attribute in an Access-Accept or CoA-Request only if an EAP-Key-Name Attribute was present in the Access-Request.

2.3. EAP-Peer-Id

Description

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

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

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

Code

TBD2

Length

>=3

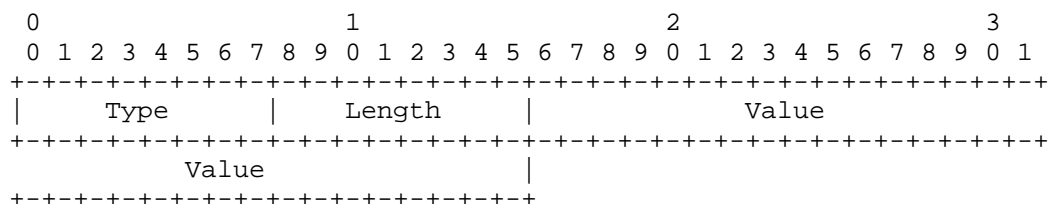
String

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

2.5. Mobility-Domain-Id

Description

A single Mobility-Domain-Id Attribute MAY be included in an Access-Request or Accounting-Request, in order to enable the NAS to provide the RADIUS server with the Mobility Domain Identifier (MDID), defined in IEEE 802.11r [IEEE-802.11r]. A summary of the Mobility-Domain-Id Attribute format is shown below. The fields are transmitted from left to right.



Code

TBD4

Length

6

Value

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

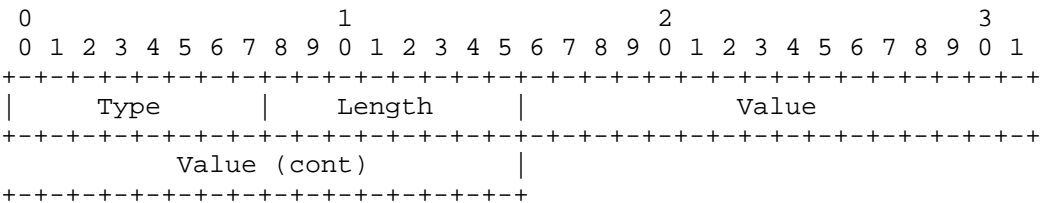
2.6. Preauth-Timeout

Description

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

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

This Attribute MAY be sent by the server to the NAS in an Access-Accept. A summary of the Preauth-Timeout Attribute format is shown below. The fields are transmitted from left to right.



Code

TBD5

Length

6

Value

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

2.7. Network-Id-Name

Description

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

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

Req	Req	#	Attribute
0+	0	TBD1	Allowed-Called-Station-Id
0-1	0	102	EAP-Key-Name
0	0+	TBD2	EAP-Peer-Id
0	0+	TBD3	EAP-Server-Id
0	0-1	TBD4	Mobility-Domain-Id
0	0	TBD5	Preauth-Timeout
0	0-1	TBD6	Network-Id-Name
0-1	0-1	TBD7	Access-Info

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

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

4. Diameter Considerations

The EAP-Key-Name Attribute is already defined as a RADIUS Attribute within Diameter EAP [RFC4072]. When used in Diameter, the other attributes defined in this specification can be used as Diameter AVPs from the Code space 1-255 (RADIUS Attribute compatibility space). No additional Diameter Code values are therefore allocated. The data types and flag rules for the attributes are as follows:

Attribute Name	Value Type	AVP Flag rules				Encr
		MUST	MAY	SHLD NOT	MUST NOT	
Allowed-Called-Station-Id	UTF8String	M	P		V	Y
EAP-Peer-Id	UTF8String	M	P		V	Y
EAP-Server-Id	UTF8String	M	P		V	Y
Mobility-Domain-Id	Unsigned32		P		V	Y
Preauth-Timeout	Unsigned32	M	P		V	Y
Network-Id-Name	UTF8String	M	P		V	Y
Access-Info	Unsigned32	M	P		V	Y

The attributes in this specification have no special translation requirements for Diameter to RADIUS or RADIUS to Diameter gateways; they are copied as is, except for changes relating to headers, alignment, and padding. See also [RFC3588] Section 4.1 and [RFC4005] Section 9.

What this specification says about the applicability of the attributes for RADIUS Access-Request packets applies in Diameter to AA-Request [RFC4005] or Diameter-EAP-Request [RFC4072]. What is said about Access-Challenge applies in Diameter to AA-Answer [RFC4005] or Diameter-EAP-Answer [RFC4072] with Result-Code AVP set to DIAMETER_MULTI_ROUND_AUTH.

What is said about Access-Accept applies in Diameter to AA-Answer or Diameter-EAP-Answer messages that indicate success. Similarly, what is said about RADIUS Access-Reject packets applies in Diameter to AA-Answer or Diameter-EAP-Answer messages that indicate failure.

What is said about COA-Request applies in Diameter to Re-Auth-Request [RFC4005]. What is said about Accounting-Request applies to Diameter Accounting- Request [RFC4005] as well.

5. IANA Considerations

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

Attribute	Type
=====	=====
Allowed-Called-Station-Id	TBD1
EAP-Peer-Id	TBD2
EAP-Server-Id	TBD3
Mobility-Domain-Id	TBD4
Preauth-Timeout	TBD5
Network-Id-Name	TBD6
Access-Info	TBD7

6. Security Considerations

Since this document describes the use of RADIUS for purposes of authentication, authorization, and accounting in IEEE 802 networks, it is vulnerable to all of the threats that are present in other RADIUS applications. For a discussion of these threats, see [RFC2607], [RFC2865], [RFC3162], [RFC3579], [RFC3580] and [RFC5176].

7. References

7.1. Normative references

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

[IEEE-802.11]

Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-2007, 2007.

[IEEE-802.11r]

Amendment to Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 2: Fast BSS Transition, IEEE 802.11r-2008, July 2008.

[IEEE-802.1X]

IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control, IEEE 802.1X-2010, February 2010.

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

[RFC2865] Rigney, C., Rubens, A., Simpson, W. and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[RFC4072] Eronen, P., Hiller, T. and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.

[RFC5247] Aboba, B., Simon, D. and P. Eronen, "EAP Key Management Framework", RFC 5247, August 2008.

7.2. Informative references

[RFC2607] Aboba, B. and J. Vollbrecht, "Proxy Chaining and Policy Implementation in Roaming", RFC 2607, June 1999.

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

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS Support for Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

- [RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G. and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, September 2003.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4005] Calhoun, P., Zorn, G., Spence, D., and D. Mitton, "Diameter Network Access Server Application", RFC 4005, August 2005.
- [RFC5176] Chiba, M., Dommetry, G., Eklund, M., Mitton, D. and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, January 2008.

Acknowledgments

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

Authors' Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: bernard_aboba@hotmail.com

Jouni Malinen
Devicescape Software, Inc.
900 Cherry Avenue
San Bruno, CA 94066

EMail: jkm@devicescape.com
Phone: +1 650 829 2600
Fax: +1 650 829 2601

Paul Congdon
Hewlett Packard Company
HP ProCurve Networking
8000 Foothills Blvd, M/S 5662
Roseville, CA 95747

Phone: +1 916 785 5753
Fax: +1 916 785 8478
E-Mail: paul_congdon@hp.com

Joseph Salowey
Cisco Systems

E-Mail: jsalowey@cisco.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/lid-abstracts.txt>.

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

This Internet-Draft will expire on November 8, 2014

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

1.1. Terminology

This document uses the following terms:

RADIUS/DTLS

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

RADIUS/DTLS client

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

implement RADIUS/DTLS.

RADIUS/DTLS server

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

RADIUS/UDP

RADIUS over UDP, as defined in [RFC2865].

RADIUS/TLS

RADIUS over TLS, as defined in [RFC6614].

silently discard

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

1.2. Requirements Language

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

1.3. Document Status

This document is an Experimental RFC.

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

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

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

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

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

2. Building on Existing Foundations

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

2.1. Changes to RADIUS

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

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

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

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

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

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

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

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

All other aspects of RADIUS are unchanged.

2.2. Similarities with RADIUS/TLS

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

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

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

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

2.2.1. Changes from RADIUS/TLS to RADIUS/DTLS

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

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

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

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

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

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

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

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

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

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

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

Section 6 applies to RADIUS/DTLS.

3. Interaction with RADIUS/UDP

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

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

This section describes how clients and servers should use

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

3.1. DTLS Port and Packet Types

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

3.2. Server Behavior

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

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

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

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

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

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

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

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

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

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

4. Client Behavior

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

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

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

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

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

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

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

5. Session Management

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

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

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

5.1. Server Session Management

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

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

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

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

DTLS Session

Any information required to maintain and manage the DTLS session.

Last Traffic

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

DTLS Data

An implementation-specific variable which may contain information

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

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

5.1.1. Session Opening and Closing

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.2. Client Session Management

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

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

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

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

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

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

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

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

6. Implementation Guidelines

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

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

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

6.1. Client Implementations

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

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

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

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

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

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

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

6.2. Server Implementations

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

7. Diameter Considerations

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

8. IANA Considerations

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

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

9. Implementation Status

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

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

9.1. Radsecproxy

Organization: Radsecproxy

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

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

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

Licensing: Freely distributable with acknowledgement

Implementation experience: No comments from implementors.

9.2. jradius

Organization: Coova

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

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

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

Licensing: Freely distributable with requirement to redistribute source.

Implementation experience: No comments from implementors.

10. Security Considerations

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

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

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

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

DoS issues which result from tracking additional information.

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

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

10.1. Crypto-Agility

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

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

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

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

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

10.2. Legacy RADIUS Security

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

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

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

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

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

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

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

10.3. Resource Exhaustion

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

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

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

10.4. Client-Server Authentication with DTLS

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

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

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

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

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

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

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

sufficient information to authenticate connecting clients.

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

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

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

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

10.5. Network Address Translation

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

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

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

10.6. Wildcard Clients

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

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

10.7. Session Closing

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

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

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

10.8. Client Subsystems

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

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

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

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

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

11. References

11.1. Normative references

[RFC2865]

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

[RFC3539]

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

[RFC5077]

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

[RFC5080]

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

[RFC5246]

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

[RFC5997]

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

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

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

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

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

11.2. Informative references

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

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

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

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

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

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

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

[MD5Attack]

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

[MD5Break]

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

Acknowledgments

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

Authors' Addresses

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

Email: aland@freeradius.org

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

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

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

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

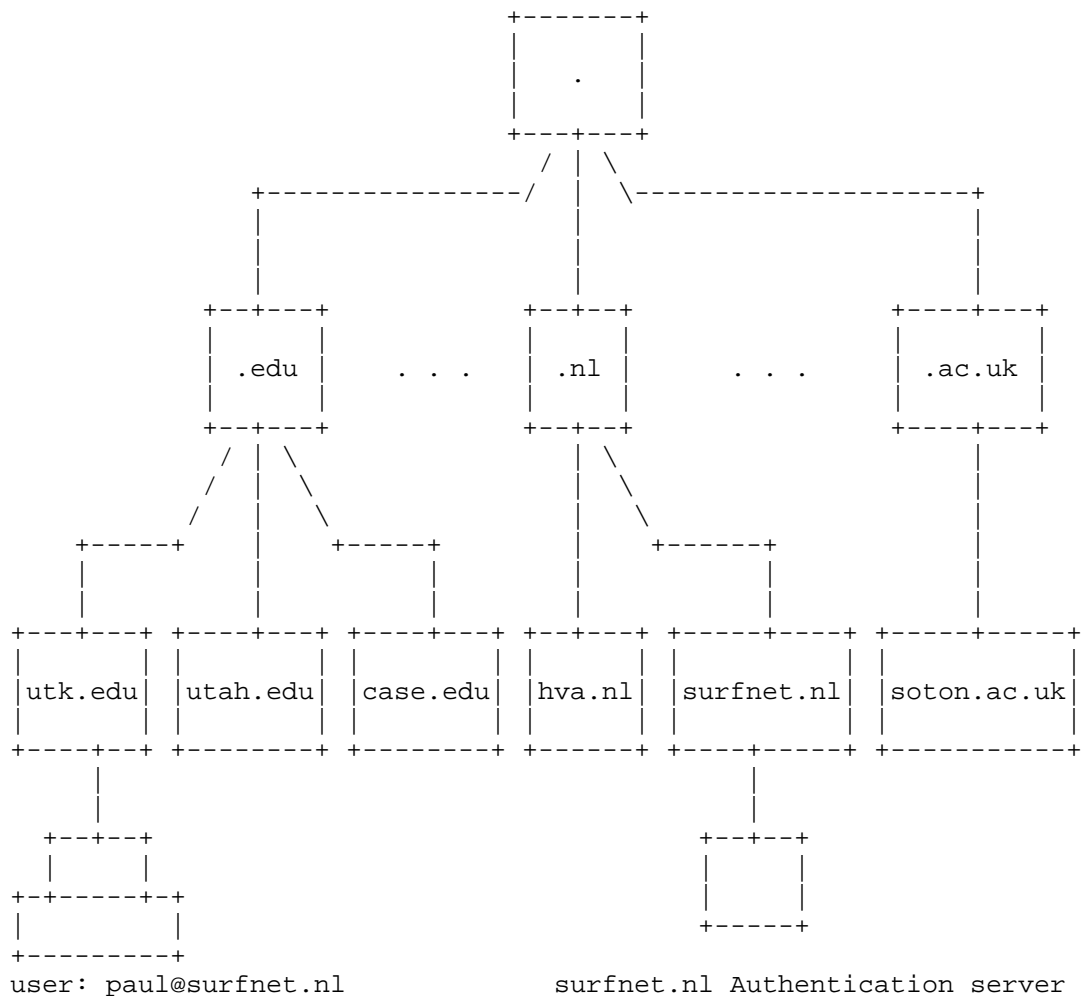


Figure 1: RADIUS hierarchy based on Top-Level Domain partitioning

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

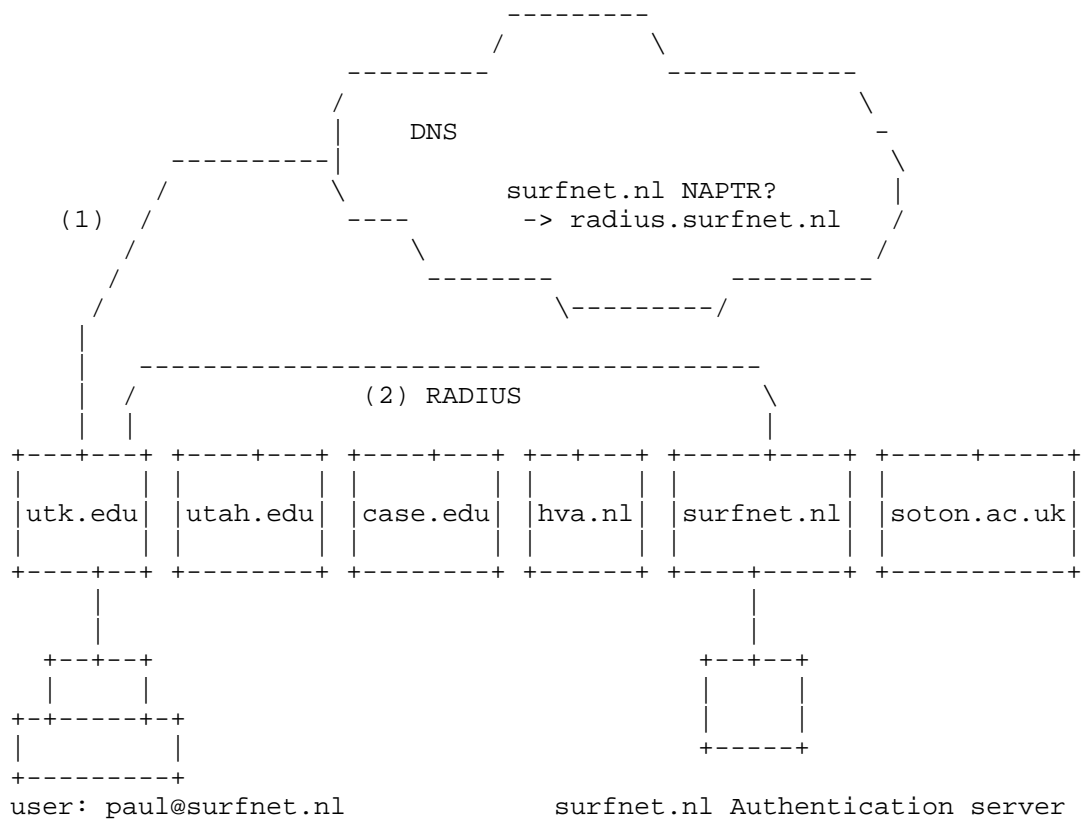


Figure 2: RADIUS hierarchy based on Top-Level Domain partitioning

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

1.1. Requirements Language

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

1.2. Terminology

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

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

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

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

1.3. Document Status

This document is an Experimental RFC.

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

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

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

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

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

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

2. Definitions

2.1. DNS Resource Record (RR) definition

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

2.1.1. S-NAPTR

2.1.1.1. Registration of Application Service and Protocol Tags

This specification defines three S-NAPTR service tags:

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

Figure 3: List of Service Tags

This specification defines two S-NAPTR protocol tags:

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

Figure 4: List of Protocol Tags

Note well:

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

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

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

2.1.1.2. Definition of Conditions for Retry/Failure

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

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

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

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

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

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

2.1.1.3. Server Identification and Handshake

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

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

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

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

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

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

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

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

2.1.1.3.2. Other mechanism: Trust Roots + policyOID

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

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

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

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

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

2.1.1.3.3. Other mechanism: DNSSEC / DANE

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

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

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

Example:

```
Realm = "example.com"

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

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

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

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

2.1.1.3.4. Client Authentication and Authorisation

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

2.1.2. SRV

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

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

Figure 5: List of SRV Labels

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

2.1.1.3. Optional name mangling

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

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

Examples:

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

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

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

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

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

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

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

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

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

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

2.2. Definition of the X.509 certificate property SubjectAltName:otherName:NAIRealm

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Examples:

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

Figure 6: Examples for NAI realm vs. certificate matching

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

3. DNS-based NAPTR/SRV Peer Discovery

3.1. Applicability

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

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

3.2. Configuration Variables

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

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

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

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

3.3. Terms

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

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

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

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

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

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

3.4. Realm to RADIUS server resolution algorithm

3.4.1. Input

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

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

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

- Usage of separators other than @.

- Encoding of User-Name in local encodings.

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

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

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

3.4.2. Output

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

3.4.3. Algorithm

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

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

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

3.4.4. Validity of results

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

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

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

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

3.4.5. Delay considerations

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

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

3.4.6. Example

Assume

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

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

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

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

May the configuration variables be

DNS_TIMEOUT = 3 seconds

MIN_EFF_TTL = 60 seconds

BACKOFF_TIME = 3600 seconds

If DNS contains the following records:

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

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

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

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

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

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

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

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

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


```
(TTL = 47) 50 50 "s" "aaa+auth:radius.tls.tcp" ""
_myradius._tcp.xn--tu-mnchen-t9a.example.

(TTL = 522) 50 50 "s" "fooservice:bar.dccp" ""
_abc123._def.xn--tu-mnchen-t9a.example.
```
7. Result:


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


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

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

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

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

4. Operations and Manageability Considerations

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

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

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

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

5. Security Considerations

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

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

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

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

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

6. Privacy Considerations

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

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

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

With RADIUS and dynamic discovery,

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

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

7. IANA Considerations

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

- o S-NAPTR Application Service Tags registry

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

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

Service Name: radiustls; radiusdtls

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

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

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

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

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

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

IANA is requested to assign the following identifiers:

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

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

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

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.
- [RFC5176] Chiba, M., Dommety, G., Eklund, M., Mitton, D., and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, January 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5580] Tschofenig, H., Adrangi, F., Jones, M., Lior, A., and B. Aboba, "Carrying Location Objects in RADIUS and Diameter", RFC 5580, August 2009.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.

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

[I-D.ietf-radext-nai]

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

8.2. Informative References

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

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

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

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

[I-D.wierenga-ietf-eduroam]

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

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

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

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS

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

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

-- Service Name Object Identifier

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

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

-- Service Name

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

ub-naiRealm-length INTEGER ::= 255

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

END
```

Authors' Addresses

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

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

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

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

Network Working Group
INTERNET-DRAFT
Category: Proposed Standard
Updates: 2865, 2866, 3575, 5176, 6158
<draft-ietf-radext-radius-extensions-13.txt>
Expires: August 25, 2013
25 February 2013

Alan DeKok
Network RADIUS
Avi Lior

Remote Authentication Dial In User Service (RADIUS) Protocol
Extensions
draft-ietf-radext-radius-extensions-13.txt

Abstract

The Remote Authentication Dial In User Service (RADIUS) protocol is nearing exhaustion of its current 8-bit Attribute Type space. In addition, experience shows a growing need for complex grouping, along with attributes which can carry more than 253 octets of data. This document defines changes to RADIUS which address all of the above problems.

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on June 26, 2013.

Copyright Notice

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

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

Table of Contents

1.	Introduction	5
1.1.	Caveats and Limitations	6
1.1.1.	Failure to Meet Certain Goals	6
1.1.2.	Implementation Recommendations	6
1.2.	Terminology	7
1.3.	Requirements Language	8
2.	Extensions to RADIUS	9
2.1.	Extended Type	10
2.2.	Long Extended Type	11
2.3.	TLV Data Type	14
2.3.1.	TLV Nesting	16
2.4.	EVS Data Type	16
2.5.	Integer64 Data Type	18
2.6.	Vendor-ID Field	18
2.7.	Attribute Naming and Type Identifiers	19
2.7.1.	Attribute and TLV Naming	19
2.7.2.	Attribute Type Identifiers	19
2.7.3.	TLV Identifiers	20
2.7.4.	VSA Identifiers	20
2.8.	Invalid Attributes	21
3.	Attribute Definitions	22
3.1.	Extended-Type-1	23
3.2.	Extended-Type-2	23
3.3.	Extended-Type-3	24
3.4.	Extended-Type-4	25
3.5.	Long-Extended-Type-1	26
3.6.	Long-Extended-Type-2	27
4.	Vendor Specific Attributes	28
4.1.	Extended-Vendor-Specific-1	29
4.2.	Extended-Vendor-Specific-2	30
4.3.	Extended-Vendor-Specific-3	31
4.4.	Extended-Vendor-Specific-4	32
4.5.	Extended-Vendor-Specific-5	33
4.6.	Extended-Vendor-Specific-6	35
5.	Compatibility with traditional RADIUS	36
5.1.	Attribute Allocation	36
5.2.	Proxy Servers	37
6.	Guidelines	38
6.1.	Updates to RFC 6158	38
6.2.	Guidelines for Simple Data Types	38
6.3.	Guidelines for Complex Data Types	39
6.4.	Design Guidelines For the New Types	40
6.5.	TLV Guidelines	41
6.6.	Allocation Request Guidelines	41
6.7.	Allocation Requests Guidelines for TLVs	42
6.8.	Implementation Guidelines	43

6.9. Vendor Guidelines	43
7. Rationale for This Design	43
7.1. Attribute Audit	44
8. Diameter Considerations	45
9. Examples	45
9.1. Extended Type	46
9.2. Long Extended Type	47
10. IANA Considerations	50
10.1. Attribute Allocations	50
10.2. RADIUS Attribute Type Tree	50
10.3. Allocation Instructions	51
10.3.1. Requested Allocation from the Standard Space ..	52
10.3.2. Requested Allocation from the short extended sp	52
10.3.3. Requested Allocation from the long extended spa	52
10.3.4. Allocation Preferences	52
10.3.5. Extending the Type Space via TLV Data Type	53
10.3.6. Allocation within a TLV	53
10.3.7. Allocation of Other Data Types	54
11. Security Considerations	54
12. References	54
12.1. Normative references	54
12.2. Informative references	55
Appendix A - Extended Attribute Generator Program	56

1. Introduction

Under current allocation pressure, we expect that the RADIUS Attribute Type space will be exhausted by 2014 or 2015. We therefore need a way to extend the type space, so that new specifications may continue to be developed. Other issues have also been shown with RADIUS. The attribute grouping method defined in [RFC2868] has been shown to be impractical, and a more powerful mechanism is needed. Multiple attributes have been defined which transport more than the 253 octets of data originally envisioned with the protocol. Each of these attributes is handled as a "special case" inside of RADIUS implementations, instead of as a general method. We therefore also need a standardized method of transporting large quantities of data. Finally, some vendors are close to allocating all of the Attributes within their Vendor-Specific Attribute space. It would be useful to leverage changes to the base protocol for extending the Vendor-Specific Attribute space.

We satisfy all of these requirements through the following changes given in this document:

- * defining an "Extended Type" format, which adds 8 bits of "Extended Type" to the RADIUS Attribute Type space, by using one octet of the "Value" field. This method gives us a general way of extending the Attribute Type Space. (Section 2.1)
- * allocating 4 attributes as using the format of "Extended Type". This allocation extends the RADIUS Attribute Type Space by approximately 1000 values. (Sections 3.1, 3.2, 3.3, and 3.4)
- * defining a "Long Extended Type" format, which inserts an additional octet between the "Extended Type" octet, and the "Value" field. This method gives us a general way of adding additional functionality to the protocol. (Section 2.2)
- * defining a method which uses the additional octet in the "Long Extended Type" to indicate data fragmentation across multiple Attributes. This method provides a standard way for an Attribute to carry more than 253 octets of data. (Section 2.2)
- * allocating 2 attributes as using the format "Long Extended Type". This allocation extends the RADIUS Attribute Type Space by an additional 500 values. (Sections 3.5 and 3.6)
- * defining a new "Type Length Value" (TLV) data type. The data type allows an attribute to carry TLVs as "sub-attributes", which can in turn encapsulate other TLVs as "sub-sub-attributes." This change creates a standard way to group a set of Attributes. (Section 2.3)

- * defining a new "extended Vendor-Specific" (EVS) data type. The data type allows an attribute to carry Vendor-Specific Attributes (VSAs) inside of the new attribute formats. (Section 2.4)
- * defining a new "integer64" data type. The data type allows counters which track more than 2^{32} octets of data. (Section 2.5)
- * allocating 6 attributes using the new EVS data type. This allocation extends the Vendor-Specific Attribute Type space by over 1500 values. (Sections 4.1 through 4.6)
- * Define the "Vendor-ID" for Vendor-Specific attributes to encompass the entire 4 octets of the Vendor field. [RFC2865] Section 5.26 defined it to be 3 octets, with the high octet being zero. (Section 2.5)
- * Describing compatibility with existing RADIUS systems. (Section 5)
- * Defining guidelines for the use of these changes for IANA, implementations of this specification, and for future RADIUS specifications. (Section 6)

As with any protocol change, the changes defined here are the result of a series of compromises. We have tried to find a balance between flexibility, space in the RADIUS message, compatibility with existing deployments, and implementation difficulty.

1.1. Caveats and Limitations

This section describes some caveats and limitations with the proposal.

1.1.1. Failure to Meet Certain Goals

One goal which was not met by the above modifications is to have an incentive for standards to use the new space. That incentive is being provided by the exhaustion of the standard space.

1.1.2. Implementation Recommendations

It is RECOMMENDED that implementations support this specification. It is RECOMMENDED that new specifications use the formats defined in this specification.

The alternative to the above recommendations is a circular argument of not implementing this specification because no other standards reference it, and also not defining new standards referencing this specification because no implementations exist.

As noted earlier, the "standard space" is almost entirely allocated. Ignoring the looming crisis benefits no one.

1.2. Terminology

This document uses the following terms:

Silently discard

This means the implementation discards the packet without further processing. The implementation MAY provide the capability of logging the error, including the contents of the silently discarded packet, and SHOULD record the event in a statistics counter.

Invalid attribute

This means that the Length field of an Attribute is valid (as per [RFC2865], Section 5, top of page 25), but the contents of the Attribute do not follow the correct format. For example, an Attribute of type "address" which encapsulates more than four, or less than four, octets of data. See Section 2.8 for a more complete definition.

Standard space

Codes in the RADIUS Attribute Type Space that are allocated by IANA and that follow the format defined in Section 5 of [RFC2865].

Extended space

Codes in the RADIUS Attribute Type Space that require the extensions defined in this document, and are an extension of the standard space, but which cannot be represented within the standard space.

Short extended space

Codes in the extended space which use the "Extended Type" format.

Long extended space

Codes in the extended space which use the "Long Extended Type" format.

The following terms are used here with the meanings defined in BCP 26 [RFC5226]: "name space", "assigned value", "registration", "Private Use", "Reserved", "Unassigned", "IETF Review", and "Standards Action".

1.3. Requirements Language

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

2. Extensions to RADIUS

This section defines two new attribute formats; "Extended Type"; and "Long Extended Type". It defines a new Type-Length-Value (TLV) data type, an Extended-Vendor-Specific (EVS) data type, and an Integer64 data type. It defines a new method for naming attributes and identifying Attributes using the new attribute formats. It finally defines the new term "invalid attribute", and describes how it affects implementations.

The new attribute formats are designed to be compatible with the attribute format given in [RFC2865] Section 5. The meaning and interpretation of the Type and Length fields is unchanged from that specification. This reuse allows the new formats to be compatible with RADIUS implementations which do not implement this specification. Those implementations can simply ignore the Value field of an attribute, or forward it verbatim.

The changes to the attribute format come about by "stealing" one or more octets from the Value field. This change has the effect that the Value field of [RFC2865] Section 5 contains both the new octets given here, and any attribute-specific Value. The result is that Values in this specification are limited to less than 253 octets in size. This limitation is overcome through the use of the "Long Extended Type" format.

We reiterate that the formats given in this document do not insert new data into an attribute. Instead, we "steal" one octet of Value, so that the definition of the Length field remains unchanged. The new attribute formats are designed to be compatible with the attribute format given in [RFC2865] Section 5. The meaning and interpretation of the Type and Length fields is unchanged from that specification. This reuse allows the new formats to be compatible RADIUS implementations which do not implement this specification. Those implementations can simply ignore the Value field of an attribute, or forward it verbatim.

The changes to the attribute format come about by "stealing" one or more octets from the Value field. This change has the effect that the Value field of [RFC2865] Section 5 contains both the new octets given here, and any attribute-specific Value. The result is that Values in this specification are limited to less than 253 octets in size. This limitation is overcome through the use of the "Long Extended Type" format.

2.1. Extended Type

This section defines a new attribute format, called "Extended Type". A summary of the Attribute format is shown below. The fields are transmitted from left to right.

```

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

```

Type

This field is identical to the Type field of the Attribute format defined in [RFC2865] Section 5.

Length

The Length field is one octet, and indicates the length of this Attribute including the Type, Length, Extended-Type, and Value fields. Permitted values are between 4 and 255. If a client or server receives an Extended Attribute with a Length of 2 or 3, then that Attribute MUST be considered to be an "invalid attribute", and handled as per Section 2.8, below.

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified according to the policies and rules described in Section 10. Unlike the Type field defined in [RFC2865] Section 5, no values are allocated for experimental or implementation-specific use. Values 241-255 are reserved and MUST NOT be used.

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

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

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

Value

This field is similar to the Value field of the Attribute format

defined in [RFC2865] Section 5. The format of the data MUST be a valid RADIUS data type.

The Value field is one or more octets.

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

The addition of the Extended-Type field decreases the maximum length for attributes of type "text" or "string" from 253 to 252 octets. Where an Attribute needs to carry more than 252 octets of data, the "Long Extended Type" format MUST be used.

Experience has shown that the "experimental" and "implementation specific" attributes defined in [RFC2865] Section 5 have had little practical value. We therefore do not continue that practice here with the Extended-Type field.

2.2. Long Extended Type

This section defines a new attribute format, called "Long Extended Type". It leverages the "Extended Type" format in order to permit the transport of attributes encapsulating more than 253 octets of data. A summary of the Attribute format is shown below. The fields are transmitted from left to right.

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

Type

This field is identical to the Type field of the Attribute format defined in [RFC2865] Section 5.

Length

The Length field is one octet, and indicates the length of this Attribute including the Type, Length, Extended-Type, and Value fields. Permitted values are between 5 and 255. If a client or server receives a "Long Extended Type" with a Length of 2, 3, or 4, then that Attribute MUST be considered to be an "invalid attribute", and be handled as per Section 2.8, below.

Note that this Length is limited to the length of this fragment. There is no field which gives an explicit value for the total size of the fragmented attribute.

Extended-Type

This field is identical to the Extended-Type field defined above in Section 2.1.

M (More)

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

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

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

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

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

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

Value

This field is similar to the Value field of the Attribute format defined in [RFC2865] Section 5. It may contain a complete set of data (when the Length field has value less than 255), or it may contain a fragment of data.

The Value field is one or more octets.

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

Any interpretation of the resulting data MUST occur after the fragments have been reassembled. The length of the data MUST be taken as the sum of the lengths of the fragments (i.e. Value fields) from which it is constructed. The format of the data SHOULD be a valid RADIUS data type. If the reassembled data does not match the expected format, all fragments MUST be treated as "invalid attributes", and the reassembled data MUST be discarded.

We note that the maximum size of a fragmented attribute is limited only by the RADIUS packet length limitation (i.e. 4096 octets, not counting various headers and overhead). Implementations MUST be able to handle the case where one fragmented attribute completely fills the packet.

This definition increases the RADIUS Attribute Type space as above, but also provides for transport of Attributes which could contain more than 253 octets of data.

Note that [RFC2865] Section 5 says:

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

These requirements also apply to the "Long Extended Type" attribute, including fragments. Implementations MUST be able to process non-contiguous fragments -- that is, fragments which are mixed together with other attributes of a different Type. This will allow them to accept packets, so long as the attributes can be correctly decoded.

2.3. TLV Data Type

We define a new data type in RADIUS, called "tlv". The "tlv" data type is an encapsulation layer which permits the "Value" field of an Attribute to contain new sub-Attributes. These sub-Attributes can in turn contain "Value"s of data type TLV. This capability both extends the attribute space, and permits "nested" attributes to be used. This nesting can be used to encapsulate or group data into one or more logical containers.

The "tlv" data type re-uses the RADIUS attribute format, as given below:

```

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

```

TLV-Type

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

As with Extended-Type above, the TLV-Type is meaningful only within the context defined by "Type" fields of the encapsulating Attributes. That is, the field may be thought of as defining a new type space of the form "Type.Extended-Type.TLV-Type". Where TLVs are nested, the type space is of the form "Type.Extended-Type.TLV-Type.TLV-Type", etc.

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

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

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

TLV-Length

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

attribute", and handled as per Section 2.8, below.

TLV-Value

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

The TLV-Value field SHOULD encapsulate a standard RADIUS data type. Non-standard data types SHOULD NOT be used within TLV-Value fields. We note that the TLV-Value field MAY also contain one or more attributes of data type "tlv", which allows for simple grouping and multiple layers of nesting.

The TLV-Value field is limited to containing 253 or fewer octets of data. Specifications which require a TLV to contain more than 253 octets of data are incompatible with RADIUS, and need to be redesigned. Specifications which require the transport of empty Values (i.e. Length = 2) are incompatible with RADIUS, and need to be redesigned.

The TLV-Value field MUST NOT contain data using the "Extended Type" formats defined in this document. The base Extended Attributes format allows for sufficient flexibility that nesting them inside of a TLV offers little additional value.

This TLV definition is compatible with the suggested format of the "String" field of the Vendor-Specific attribute, as defined in [RFC2865] Section 5.26, though that specification does not discuss nesting.

Vendors MAY use attributes of type "tlv" in any Vendor Specific Attribute. It is RECOMMENDED to use type "tlv" for VSAs, in preference to any other format.

If multiple TLVs with the same TLV-Type are present, the order of TLVs with the same TLV-Type MUST be preserved by any proxies. The order of TLVs of different TLV-Types is not required to be preserved. A RADIUS server or client MUST NOT have any dependencies on the order of TLVs of different TLV-Types. A RADIUS server or client MUST NOT require TLVs of the same TLV-type to be contiguous.

The interpretation of multiple TLVs of the same TLV-Type MUST be that of a logical "and", unless otherwise specified. That is, multiple TLVs are interpreted as specifying an unordered set of values. Specifications SHOULD NOT define TLVs to be interpreted as a logical "or". Doing so would mean that a RADIUS client or server would make an arbitrary, and non-deterministic choice among the values.

2.3.1. TLV Nesting

TLVs may contain other TLVs. When this occurs, the "container" TLV MUST be completely filled by the "contained" TLVs. That is, the "container" TLV-Length field MUST be exactly two (2) more than the sum of the "contained" TLV-Length fields. If the "contained" TLVs over-fill the "container" TLV, the "container" TLV MUST be considered to be an "invalid attribute", and handled as described in Section 2.8, below.

The depth of TLV nesting is limited only by the restrictions on the TLV-Length field. The limit of 253 octets of data results in a limit of 126 levels of nesting. However, nesting depths of more than 4 are NOT RECOMMENDED. They have not been demonstrated to be necessary in practice, and they appear to make implementations more complex. Reception of packets with such deeply nest TLVs may indicate implementation errors or deliberate attacks. Where implementations do not support deep nesting of TLVs, it is RECOMMENDED that the unsupported layers are treated as "invalid attributes".

2.4. EVS Data Type

We define a new data type in RADIUS, called "evs", for "Extended Vendor-Specific". The "evs" data type is an encapsulation layer which permits the "Value" field of an Attribute to contain a Vendor-Id, followed by a Vendor-Type, and then vendor-defined data. This data can in turn contain valid RADIUS data types, or any other data as determined by the vendor.

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

Where [RFC2865] Section 5.26 makes a recommendation for the format of the data following the Vendor-Id, we give a strict definition. Experience has shown that many vendors have not followed the [RFC2865] recommendations, leading to interoperability issues. We hope here to give vendors sufficient flexibility as to meet their needs, while minimizing the use of non-standard VSA formats.

The "evs" data type MAY be used in Attributes having the format of "Extended Type" or "Long Extended Type". It MUST NOT be used in any other Attribute definition, including standard RADIUS Attributes, TLVs, and VSAs.

A summary of the "evs" data type format is shown below. The fields

are transmitted from left to right.

```

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

```

Vendor-Id

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

Vendor-Type

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

Vendor-Value

The Vendor-Value field is one or more octets. It SHOULD encapsulate a standard RADIUS data type. Using non-standard data types is NOT RECOMMENDED. We note that the Value field may be of data type "tlv". However, it MUST NOT be of data type "evs", as the use cases are unclear for one vendor delegating Attribute Type space to another vendor.

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

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

Note that unlike the format described in [RFC2865] Section 5.26, this data type has no "Vendor length" field. The length of the Vendor-Value field is implicit, and is determined by taking the "Length" of the encapsulating RADIUS Attribute, and subtracting the length of the attribute header (2 octets), the extended type (1 octet), the Vendor-Id (4 octets), and the Vendor-type (1 octet). i.e. For "Extended Type" attributes, the length of the Vendor-Value field is eight (8) less than the value of the Length field. For "Long Extended Type" attributes, the length of the Vendor-Value field is nine (9) less than the value of the Length field.

2.5. Integer64 Data Type

We define a new data type in RADIUS, called "integer64", which carries a 64-bit unsigned integer in network byte order.

This data type is intended to be used in any situation where there is a need to have counters which can count past 2^{32} . The expected use of this data type is within Accounting-Request packets, but this data type SHOULD be used in any packet where 32-bit integers are expected to be insufficient.

The "integer64" data type can be used in Attributes of any format, standard space, extended attributes, TLVs, and VSAs.

A summary of the "integer64" data type format is shown below. The fields are transmitted from left to right.

```

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

```

Attributes having data type "integer64" MUST have the relevant Length field set to eight more than the length of the Attribute header. For standard space Attributes and TLVs, this means that the Length field MUST be set to ten (10). For "Extended Type" Attributes, the Length field MUST be set to eleven (11). For "Long Extended Type" Attributes, the Length field MUST be set to twelve (12).

2.6. Vendor-ID Field

We define the Vendor-ID field of Vendor-Specific Attributes to encompass the entire 4 octets of the Vendor field. [RFC2865] Section 5.26 defined it to be 3 octets, with the high octet being zero. This change has no immediate impact on RADIUS, as the maximum Private Enterprise Code defined is still within 16 bits.

However, it is best to make advance preparations for changes in the protocol. As such, it is RECOMMENDED that all implementations support four (4) octets for the Vendor-ID field, instead of three (3).

2.7. Attribute Naming and Type Identifiers

Attributes have traditionally been identified by a unique name and number. For example, the attribute named "User-Name" has been allocated number one (1). This scheme needs to be extended in order to be able to refer to attributes of Extended Type, and to TLVs. It will also be used by IANA for allocating RADIUS Attribute Type values.

The names and identifiers given here are intended to be used only in specifications. The system presented here may not be useful when referring to the contents of a RADIUS packet. It imposes no requirements on implementations, as implementations are free to reference RADIUS Attributes via any method they choose.

2.7.1. Attribute and TLV Naming

RADIUS specifications traditionally use names consisting of one or more words, separated by hyphens, e.g. "User-Name". However, these names are not allocated from a registry, and there is no restriction other than convention on their global uniqueness.

Similarly, vendors have often used their company name as the prefix for VSA names, though this practice is not universal. For example, for a vendor named "Example", the name "Example-Attribute-Name" SHOULD be used instead of "Attribute-Name". The second form can conflict with attributes from other vendors, whereas the first form cannot.

It is therefore RECOMMENDED that specifications give names to Attributes which attempt to be globally unique across all RADIUS Attributes. It is RECOMMENDED that vendors use their name as a unique prefix for attribute names, e.g. Livingston-IP-Pool instead of IP-Pool. It is RECOMMENDED that implementations enforce uniqueness on names, which would otherwise lead to ambiguity and problems.

We recognise that these suggestions may sometimes be difficult to implement in practice.

TLVs SHOULD be named with a unique prefix that is shared among related attributes. For example, a specification that defines a set of TLVs related to time could create attributes named "Time-Zone", "Time-Day", "Time-Hour", "Time-Minute", etc.

2.7.2. Attribute Type Identifiers

The RADIUS Attribute Type space defines a context for a particular "Extended-Type" field. The "Extended-Type" field allows for 256

possible type code values, with values 1 through 240 available for allocation. We define here an identification method that uses a "dotted number" notation similar to that used for Object Identifiers (OIDs), formatted as "Type.Extended-Type".

For example, an attribute within the Type space of 241, having Extended-Type of one (1), is uniquely identified as "241.1". Similarly, an attribute within the Type space of 246, having Extended-Type of ten (10), is uniquely identified as "246.10".

2.7.3. TLV Identifiers

We can extend the Attribute reference scheme defined above for TLVs. This is done by leveraging the "dotted number" notation. As above, we define an additional TLV type space, within the "Extended Type" space, by appending another "dotted number" in order to identify the TLV. This method can be repeated in sequence for nested TLVs.

For example, let us say that "245.1" identifies RADIUS Attribute Type 245, containing an "Extended Type" of one (1), which is of type "tlv". That attribute will contain 256 possible TLVs, one for each value of the TLV-Type field. The first TLV-Type value of one (1) can then be identified by appending a ".1" to the number of the encapsulating attribute ("241.1"), to yield "241.1.1". Similarly, the sequence "245.2.3.4" identifies RADIUS attribute 245, containing an "Extended Type" of two (2) which is of type "tlv", which in turn contains a TLV with TLV-Type number three (3), which in turn contains another TLV, with TLV-Type number four (4).

2.7.4. VSA Identifiers

There has historically been no method for numerically addressing VSAs. The "dotted number" method defined here can also be leveraged to create such an addressing scheme. However, as the VSAs are completely under the control of each individual vendor, this section provides a suggested practice, but does not define a standard of any kind.

The Vendor-Specific Attribute has been assigned the Attribute number 26. It in turn carries a 24-bit Vendor-Id, and possibly additional VSAs. Where the VSAs follow the [RFC2865] Section 5.26 recommended format, a VSA can be identified as "26.Vendor-Id.Vendor-Type".

For example, Livingston has Vendor-Id 307, and has defined an attribute "IP-Pool" as number 6. This VSA can be uniquely identified as 26.307.6, but it cannot be uniquely identified by name, as other vendors may have used the same name.

Note that there are few restrictions on the size of the numerical values in this notation. The Vendor-Id is a 24-bit number, and the VSA may have been assigned from a 16-bit vendor-specific Attribute Type space. Implementations SHOULD be capable of handling 32-bit numbers at each level of the "dotted number" notation.

For example, the company USR has been allocated Vendor-Id 429, and has defined a "Version-Id" attribute as number 32768. This VSA can be uniquely identified as 26.429.32768, and again cannot be uniquely identified by name.

Where a VSA is a TLV, the "dotted number" notation can be used as above: 26.Vendor-Id.Vendor-Type.TLV1.TLV2.TLV3 where "TLVn" are the numerical values assigned by the vendor to the different nested TLVs.

2.8. Invalid Attributes

The term "invalid attribute" is new to this specification. It is defined to mean that the Length field of an Attribute permits the packet to be accepted as not being "malformed". However, the Value field of the attribute does not follow the format required by the data type defined for that Attribute, and therefore the attribute is "malformed". In order to distinguish the two cases, we refer to "malformed" packets, and "invalid attributes".

For example, an implementation receives a packet which is well-formed. That packet contains an Attribute allegedly of data type "address", but which has Length not equal to four. In that situation, the packet is well formed, but the attribute is not. Therefore, it is an "invalid attribute".

A similar analysis can be performed when an attribute carries TLVs. The encapsulating attribute may be well formed, but the TLV may be an "invalid attribute". The existence of an "invalid attribute" in a packet or attribute MUST NOT result in the implementation discarding the entire packet, or treating the packet as a negative acknowledgment. Instead, only the "invalid attribute" is treated specially.

When an implementation receives an "invalid attribute", it SHOULD be silently discarded, except when the implementation is acting as a proxy (see Section 5.2 for discussion of proxy servers). If it is not discarded, it MUST NOT be handled in the same manner as a well-formed attribute. For example, receiving an Attribute of data type "address" containing less than four octets, or more than four octets of data means that the Attribute MUST NOT be treated as being of data type "address". The reason here is that if the attribute does not carry an IPv4 address, the receiver has no idea what format the data

is in, and it is therefore not an IPv4 address.

For Attributes of type "Long Extended Type", an Attribute is considered to be an "invalid attribute" when it does not match the criteria set out in Section 2.2, above.

For Attributes of type "TLV", an Attribute is considered to be an "invalid attribute" when the TLV-Length field allows the encapsulating Attribute to be parsed, but the TLV-Value field does not match the criteria for that TLV. Implementations SHOULD NOT treat the "invalid attribute" property as being transitive. That is, the Attribute encapsulating the "invalid attribute" SHOULD NOT be treated as an "invalid attribute". That encapsulating Attribute might contain multiple TLVs, only one of which is an "invalid attribute".

However, a TLV definition may require particular sub-TLVs to be present, and/or to have specific values. If a sub-TLV is missing, or contains incorrect value(s), or is an "invalid attribute", then the encapsulating TLV SHOULD be treated as an "invalid attribute". This requirement ensures that strongly connected TLVs are handled either as a coherent whole, or are ignored entirely.

It is RECOMMENDED that Attributes with unknown Type, Ext-Type, TLV-Type, or VSA-Type are treated as "invalid attributes". This recommendation is compatible with the suggestion in [RFC2865] Section 5, that implementations "MAY ignore Attributes with an unknown Type".

3. Attribute Definitions

We define four (4) attributes of "Extended Type", which are allocated from the "Reserved" Attribute Type codes of 241, 242, 243, and 244. We also define two (2) attributes of "Long Extended Type", which are allocated from the "Reserved" Attribute Type codes of 245 and 246.

Type	Name
----	----
241	Extended-Type-1
242	Extended-Type-2
243	Extended-Type-3
244	Extended-Type-4
245	Long-Extended-Type-1
246	Long-Extended-Type-2

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

3.1. Extended-Type-1

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 241.{1-255}.

A summary of the Extended-Type-1 Attribute format is shown below. The fields are transmitted from left to right.

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

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.

```

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

```

Type

242 for Extended-Type-2.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 242.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

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

3.3. Extended-Type-3

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 243.{1-255}.

A summary of the Extended-Type-3 Attribute format is shown below. The fields are transmitted from left to right.

```

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

```

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

243 for Extended-Type-3.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 243.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

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

3.4. Extended-Type-4**Description**

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type Space of 244.{1-255}.

A summary of the Extended-Type-4 Attribute format is shown below. The fields are transmitted from left to right.

```

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

```

Type

244 for Extended-Type-4.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 244.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The Value field is one or more octets.

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

3.5. Long-Extended-Type-1

Description

This attribute encapsulates attributes of the "Long Extended Type" format, in the RADIUS Attribute Type Space of 245.{1-255}.

A summary of the Long-Extended-Type-1 Attribute format is shown below. The fields are transmitted from left to right.

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

Type

245 for Long-Extended-Type-1

Length

>= 5

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this

field are specified in the 245.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Value

The Value field is one or more octets.

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

3.6. Long-Extended-Type-2

Description

This attribute encapsulates attributes of the "Long Extended Type" format, in the RADIUS Attribute Type Space of 246.{1-255}.

A summary of the Long-Extended-Type-2 Attribute format is shown below. The fields are transmitted from left to right.

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

Type

246 for Long-Extended-Type-2

Length

>= 5

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 246.{1-255} RADIUS Attribute Type Space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Value

The Value field is one or more octets.

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

4. Vendor Specific Attributes

We define six new attributes which can carry Vendor Specific information. We define four (4) attributes of the "Extended Type" format, with Type codes (241.26, 242.26, 243.26, 244.26), using the "evs" data type. We also define two (2) attributes using "Long Extended Type" format, with Type codes (245.26, 246.26), which are of the "evs" data type.

Type.Extended-Type	Name
-----	----
241.26	Extended-Vendor-Specific-1
242.26	Extended-Vendor-Specific-2
243.26	Extended-Vendor-Specific-3
244.26	Extended-Vendor-Specific-4

245.26 Extended-Vendor-Specific-5
 246.26 Extended-Vendor-Specific-6

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

4.1. Extended-Vendor-Specific-1

Description

This attribute defines a RADIUS Type Code of 241.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-1 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |      Length      | Extended-Type | Vendor-Id ...
+-----+-----+-----+-----+-----+-----+-----+-----+
| ... Vendor-Id (cont) | Vendor-Type |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value ....
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type.Extended-Type

241.26 for Extended-Vendor-Specific-1

Length

>= 9

Vendor-Id

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

Vendor-Type

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

Value

The Value field is one or more octets. The actual format of the information is site or application specific, and a robust

implementation SHOULD support the field as undistinguished octets.

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

The length of the Value field is eight (8) less than the value of the Length field.

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

4.2. Extended-Vendor-Specific-2

Description

This attribute defines a RADIUS Type Code of 242.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-2 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Type      |      Length      | Extended-Type | Vendor-Id ...
+-----+-----+-----+-----+-----+-----+-----+-----+
| ... Vendor-Id (cont) | Vendor-Type |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value ....
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type.Extended-Type

242.26 for Extended-Vendor-Specific-2

Length

>= 9

Vendor-Id

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

Vendor-Type

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

sole discretion of the Vendor.

Value

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

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

The length of the Value field is eight (8) less than the value of the Length field.

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

4.3. Extended-Vendor-Specific-3

Description

This attribute defines a RADIUS Type Code of 243.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-3 Attribute format is shown below. The fields are transmitted from left to right.

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      | Extended-Type | Vendor-Id ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ... Vendor-Id (cont) | Vendor-Type |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Value ....
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type.Extended-Type

243.26 for Extended-Vendor-Specific-3

Length

>= 9

Vendor-Id

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

Vendor-Type

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

Value

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

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

The length of the Value field is eight (8) less than the value of the Length field.

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

4.4. Extended-Vendor-Specific-4

Description

This attribute defines a RADIUS Type Code of 244.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-3 Attribute format is shown below. The fields are transmitted from left to right.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Type   |   Length   | Extended-Type | Vendor-Id ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
... Vendor-Id (cont) | Vendor-Type |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Value ....
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type.Extended-Type

244.26 for Extended-Vendor-Specific-4

Length

>= 9

Vendor-Id

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

Vendor-Type

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

Value

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

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

The length of the Value field is eight (8) less than the value of the Length field.

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

4.5. Extended-Vendor-Specific-5

Description

This attribute defines a RADIUS Type Code of 245.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-5 Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
Type										Length										Extended-Type										M										Reserved									
Vendor-Id																																																	
Vendor-Type										Value																																							

+-----+

Type.Extended-Type

245.26 for Extended-Vendor-Specific-5

Length

>= 10 (first fragment)
>= 5 (subsequent fragments)

When a VSA is fragmented across multiple Attributes, only the first Attribute contains the Vendor-Id and Vendor-Type fields. Subsequent Attributes contain fragments of the Value field only.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Vendor-Id

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

Vendor-Type

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

Value

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

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

Implementations supporting this specification MUST use the

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

4.6. Extended-Vendor-Specific-6

Description

This attribute defines a RADIUS Type Code of 246.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-6 Attribute format is shown below. The fields are transmitted from left to right.

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
Type										Length										Extended-Type										M										Reserved									
										Vendor-Id																																							
Vendor-Type										Value																																							

Type.Extended-Type

246.26 for Extended-Vendor-Specific-6

Length

>= 10 (first fragment)
>= 5 (subsequent fragments)

When a VSA is fragmented across multiple Attributes, only the first Attribute contains the Vendor-Id and Vendor-Type fields. Subsequent Attributes contain fragments of the Value field only.

M (More)

The More field is one (1) bit in length, and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long, and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Vendor-Id

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

Vendor-Type

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

Value

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

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

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

5. Compatibility with traditional RADIUS

There are a number of potential compatibility issues with traditional RADIUS, as defined in [RFC6158] and earlier. This section describes them.

5.1. Attribute Allocation

Some vendors have used Attribute Type codes from the "Reserved" space, as part of vendor-defined dictionaries. This practice is considered anti-social behavior, as noted in [RFC6158]. These vendor definitions conflict with the attributes in the RADIUS Attribute Type space. The conflicting definitions may make it difficult for implementations to support both those Vendor Attributes, and the new Extended Attribute formats.

We RECOMMEND that RADIUS client and server implementations delete all references to these improperly defined attributes. Failing that, we RECOMMEND that RADIUS server implementations have a per-client configurable flag which indicates which type of attributes are being sent from the client. If the flag is set to "Non-Standard Attributes", the conflicting attributes can be interpreted as being improperly defined Vendor Specific Attributes. If the flag is set the "IETF Attributes", the attributes MUST be interpreted as being of the Extended Attributes format. The default SHOULD be to interpret the

attributes as being of the Extended Attributes format.

Other methods of determining how to decode the attributes into a "correct" form are NOT RECOMMENDED. Those methods are likely to be fragile and prone to error.

We RECOMMEND that RADIUS server implementations re-use the above flag to determine which type of attributes to send in a reply message. If the request is expected to contain the improperly defined attributes, the reply SHOULD NOT contain Extended Attributes. If the request is expected to contain Extended Attributes, the reply MUST NOT contain the improper Attributes.

RADIUS clients will have fewer issues than servers. Clients MUST NOT send improperly defined Attributes in a request. For replies, clients MUST interpret attributes as being of the Extended Attributes format, instead of the improper definitions. These requirements impose no change in the RADIUS specifications, as such usage by vendors has always been in conflict with the standard requirements and the standards process.

Existing clients that send these improperly defined attributes usually have a configuration setting which can disable this behavior. We RECOMMEND that vendors ship products with the default set to "disabled". We RECOMMEND that administrators set this flag to "disabled" on all equipment that they manage.

5.2. Proxy Servers

RADIUS Proxy servers will need to forward Attributes having the new format, even if they do not implement support for the encoding and decoding of those attributes. We remind implementers of the following text in [RFC2865] Section 2.3:

The forwarding server MUST NOT change the order of any attributes of the same type, including Proxy-State.

This requirement solves some of the issues related to proxying of the new format, but not all. The reason is that proxy servers are permitted to examine the contents of the packets that they forward. Many proxy implementations not only examine the attributes, but they refuse to forward attributes which they do not understand (i.e. attributes for which they have no local dictionary definitions).

This practice is NOT RECOMMENDED. Proxy servers SHOULD forward attributes, even ones which they do not understand, or which are not in a local dictionary. When forwarded, these attributes SHOULD be sent verbatim, with no modifications or changes. This requirement

includes "invalid attributes", as there may be some other system in the network which understands them.

The only exception to this recommendation is when local site policy dictates that filtering of attributes has to occur. For example, a filter at a visited network may require removal of certain authorization rules which apply to the home network, but not to the visited network. This filtering can sometimes be done even when the contents of the attributes are unknown, such as when all Vendor-Specific Attributes are designated for removal.

As seen in [EDUROAM] many proxies do not follow these practices for unknown Attributes. Some proxies filter out unknown attributes or attributes which have unexpected lengths (24%, 17/70), some truncate the attributes to the "expected" length (11%, 8/70), some discard the request entirely (1%, 1/70), with the rest (63%, 44/70) following the recommended practice of passing the attributes verbatim. It will be difficult to widely use the Extended Attributes format until all non-conformant proxies are fixed. We therefore RECOMMEND that all proxies which do not support the Extended Attributes (241 through 246) define them as being of data type "string", and delete all other local definitions for those attributes.

This last change should enable wider usage of the Extended Attributes format.

6. Guidelines

This specification proposes a number of changes to RADIUS, and therefore requires a set of guidelines, as has been done in [RFC6158]. These guidelines include suggestions around design, interaction with IANA, usage, and implementation of attributes using the new formats.

6.1. Updates to RFC 6158

This specification updates [RFC6158] by adding the data types "evs", "tlv" and "integer64"; defining them to be "basic" data types; and permitting their use subject to the restrictions outlined below.

The recommendations for the use of the new data types and attribute formats are given below.

6.2. Guidelines for Simple Data Types

[RFC6158] Section A.2.1 says in part:

- * Unsigned integers of size other than 32 bits.

SHOULD be replaced by an unsigned integer of 32 bits. There is insufficient justification to define a new size of integer.

We update that specification to permit unsigned integers of 64 bits, for the reasons defined above in Section 2.5. The updated text is as follows:

- * Unsigned integers of size other than 32 or 64 bits.
SHOULD be replaced by an unsigned integer of 32 or 64 bits.
There is insufficient justification to define a new size of integer.

That section later continues with the following list item:

- * Nested attribute-value pairs (AVPs).
Attributes should be defined in a flat typespace.

We update that specification to permit nested TLVs, as defined in this document:

- * Nested attribute-value pairs (AVPs) using the extended attribute format MAY be used. All other nested AVP or TLV formats MUST NOT be used.

The [RFC6158] recommendations for "basic" data types apply to the three types listed above. All other recommendations given in [RFC6158] for "basic" data types remain unchanged.

6.3. Guidelines for Complex Data Types

[RFC6158] Section 2.1 says:

Complex data types MAY be used in situations where they reduce complexity in non-RADIUS systems or where using the basic data types would be awkward (such as where grouping would be required in order to link related attributes).

Since the extended attribute format allows for grouping of complex types via TLVs, the guidelines for complex data types need to be updated as follows:

[RFC6158], Section 3.2.4, describes situations in which complex data types might be appropriate. They SHOULD NOT be used even in those situations, without careful consideration of the described limitations. In all other cases not covered by the complex data type exceptions, complex data types MUST NOT be used. Instead,

complex data types MUST be decomposed into TLVs.

The checklist in Appendix A.2.2 is similarly updated to add a new requirement at the top of that section,

Does the attribute:

- * define a complex type which can be represented via TLVs?

If so, this data type MUST be represented via TLVs.

Note that this requirement does not over-ride Section A.1, which permits the transport of complex types in certain situations.

All other recommendations given in [RFC6158] for "complex" data types remain unchanged.

6.4. Design Guidelines For the New Types

This section gives design guidelines for specifications defining attributes using the new format. The items listed below are not exhaustive. As experience is gained with the new formats, later specifications may define additional guidelines.

- * The data type "evs" MUST NOT be used for standard RADIUS Attributes, or for TLVs, or for VSAs.
- * The data type "tlv" SHOULD NOT be used for standard RADIUS attributes.
- * [RFC2866] "tagged" attributes MUST NOT be defined in the Extended-Type space. The "tlv" data type should be used instead to group attributes.
- * The "integer64" data type MAY be used in any RADIUS attribute. The use of 64-bit integers was not recommended in [RFC6158], but their utility is now evident.
- * Any attribute which is allocated from the "long extended space" of data type "text", "string", or "tlv" can potentially carry more than 251 octets of data. Specifications defining such attributes SHOULD define a maximum length to guide implementations.

All other recommendations given in [RFC6158] for attribute design guidelines apply to attributes using the "short extended space" and "long extended space".

6.5. TLV Guidelines

The following items give design guidelines for specifications using TLVs.

- * when multiple attributes are intended to be grouped or managed together, the use of TLVs to group related attributes is RECOMMENDED.
- * more than 4 layers (depth) of TLV nesting is NOT RECOMMENDED.
- * Interpretation of an attribute depends only on its type definition (e.g. Type.Extended-Type.TLV-Type), and not on its encoding or location in the RADIUS packet.
- * Where a group of TLVs is strictly defined, and not expected to change, and totals less than 247 octets of data, they SHOULD request allocation from the "short extended space".
- * Where a group of TLVs is loosely defined, or is expected to change, they SHOULD request allocation from the "long extended space".

All other recommendations given in [RFC6158] for attribute design guidelines apply to attributes using the TLV format.

6.6. Allocation Request Guidelines

The following items give guidelines for allocation requests made in a RADIUS specification.

- * Discretion is recommended when requesting allocation of attributes. The new space is much larger than the old one, but it is not infinite.
- * Specifications which allocate many attributes MUST NOT request that allocation be made from the standard space. That space is under allocation pressure, and the extended space is more suitable for large allocations. As a guideline, we suggest that one specification allocating twenty percent (20%) or more of the standard space would meet the above criteria.
- * Specifications which allocate many related attributes SHOULD define one or more TLVs to contain related attributes.
- * Specifications SHOULD request allocation from a specific space. The IANA considerations given in Section 9, below, give instruction to IANA, but authors should assist IANA where possible.

- * Specifications of an attribute which encodes 252 octets or less of data MAY request allocation from the "short extended space".
- * Specifications of an attribute which always encode less than 253 octets of data MUST NOT request allocation from the long extended space. The standard space, or the short extended space MUST be used instead.
- * Specifications of an attribute which encodes 253 octets or more of data MUST request allocation from the "long extended space".
- * When the extended space is nearing exhaustion, a new specification will have to be written which requests allocation of one or more RADIUS Attributes from the "Reserved" portion of the standard space, values 247-255, using an appropriate format ("Short Extended Type", or "Long Extended Type")

An allocation request made in a specification SHOULD use one of the following formats when allocating an attribute type code:

- * TBDn - request allocation of an attribute from the "standard space". The value "n" should be "1" or more, to track individual attributes which are to be allocated.
- * SHORT-TBDn - request allocation of an attribute from the "short extended space". The value "n" should be "1" or more, to track individual attributes which are to be allocated.
- * LONG-TBDn - request allocation of an attribute from the "long extended space". The value "n" should be "1" or more, to track individual attributes which are to be allocated.

These guidelines should help specification authors and IANA communicate effectively and clearly.

6.7. Allocation Requests Guidelines for TLVs

Specifications may allocate a new attribute of type TLV, and at the same time, allocate sub-attributes within that TLV. These specifications SHOULD request allocation of specific values for the sub-TLV. The "dotted number" notation MUST be used.

For example, a specification may request allocation of a TLV as SHORT-TBD1. Within that attribute, it could request allocation of three sub-TLVs, as SHORT-TBD1.1, SHORT-TBD1.2, and SHORT-TBD1.3.

Specifications may request allocation of additional sub-TLVs within an existing attribute of type TLV. Those specifications SHOULD use

the "TBDn" format for every entry in the "dotted number" notation.

For example, a specification may request allocation within an existing TLV, with "dotted number" notation MM.NN. Within that attribute, the specification could request allocation of three sub-TLVs, as MM.NN.TBD1, MM.NN.TBD2, and MM.NN.TBD3.

6.8. Implementation Guidelines

- * RADIUS client implementations SHOULD support this specification, in order to permit the easy deployment of specifications using the changes defined herein.
- * RADIUS server implementations SHOULD support this specification, in order to permit the easy deployment of specifications using the changes defined herein.
- * RADIUS proxy servers MUST follow the specifications in section 5.2

6.9. Vendor Guidelines

- * Vendors SHOULD use the existing Vendor-Specific Attribute Type space in preference to the new Extended-Vendor-Specific attributes, as this specification may take time to become widely deployed.
- * Vendors SHOULD implement this specification. The changes to RADIUS are relatively small, and are likely to quickly be used in new specifications.

7. Rationale for This Design

The path to extending the RADIUS protocol has been long and arduous. A number of proposals have been made and discarded by the RADEXT working group. These proposals have been judged to be either too bulky, too complex, too simple, or to be unworkable in practice. We do not otherwise explain here why earlier proposals did not obtain working group consensus.

The changes outlined here have the benefit of being simple, as the "Extended Type" format requires only a one octet change to the Attribute format. The downside is that the "Long Extended Type" format is awkward, and the 7 Reserved bits will likely never be used for anything.

7.1. Attribute Audit

An audit of almost five thousand publicly available attributes [ATTR] (2010), shows the statistics summarized below. The attributes include over 100 Vendor dictionaries, along with the IANA assigned attributes:

Count	Data Type
-----	-----
2257	integer
1762	text
273	IPv4 Address
225	string
96	other data types
35	IPv6 Address
18	date
10	integer64
4	Interface Id
3	IPv6 Prefix
4683	Total

The entries in the "Data Type" column are data types recommended by [RFC6158], along with "integer64". The "other data types" row encompasses all other data types, including complex data types and data types transporting opaque data.

We see that over half of the attributes encode less than 16 octets of data. It is therefore important to have an extension mechanism which adds as little as possible to the size of these attributes. Another result is that the overwhelming majority of attributes use simple data types.

Of the attributes defined above, 177 were declared as being inside of a TLV. This is approximately 4% of the total. We did not investigate whether additional attributes were defined in a flat name space, but could have been defined as being inside of a TLV. We expect that the number could be as high as 10% of attributes.

Manual inspection of the dictionaries shows that approximately 20 (or 0.5%) attributes have the ability to transport more than 253 octets of data. These attributes are divided between VSAs, and a small number of standard Attributes such as EAP-Message.

The results of this audit and analysis is reflected in the design of the extended attributes. The extended format has minimal overhead, it permits TLVs, and it has support for "long" attributes.

8. Diameter Considerations

The attribute formats defined in this specification need to be transported in Diameter. While Diameter supports attributes longer than 253 octets and grouped attributes, we do not use that functionality here. Instead, we define the simplest possible encapsulation method.

The new formats MUST be treated the same as traditional RADIUS attributes when converting from RADIUS to Diameter, or vice versa. That is, the new attribute space is not converted to any "extended" Diameter attribute space. Fragmented attributes are not converted to a single long Diameter attribute. The new EVS data types are not converted to Diameter attributes with the "V" bit set.

In short, this document mandates no changes for existing RADIUS to Diameter, or Diameter to RADIUS gateways.

9. Examples

A few examples are presented here in order to illustrate the encoding of the new attribute formats. These examples are not intended to be exhaustive, as many others are possible. For simplicity, we do not show complete packets, only attributes.

The examples are given using a domain-specific language implemented by the program given in Appendix A. The language is line oriented, and composed of a sequence of lines matching the grammar ([RFC5234]) given below:

```
Identifier = 1*DIGIT *( "." 1*DIGIT )

HEXCHAR = HEXDIG HEXDIG

STRING = DQUOTE 1*CHAR DQUOTE

TLV = "{" SP 1*DIGIT SP DATA SP "}"

DATA = (HEXCHAR *(SP HEXCHAR)) / (TLV *(SP TLV)) / STRING

LINE = Identifier SP DATA
```

The program has additional restrictions on its input that are not reflected in the above grammar. For example, the portions of the Identifier which refer to Type and Extended-Type are limited to values between 1 and 255. We trust that the source code in Appendix A is clear, and that these restrictions do not negatively affect the comprehensibility of the examples.

The program reads the input text, and interprets it as a set of instructions to create RADIUS Attributes. It then prints the hex encoding of those attributes. It implements the minimum set of functionality which achieves that goal. This minimalism means that it does not use attribute dictionaries; it does not implement support for RADIUS data types; it can be used to encode attributes with invalid data fields; and there is no requirement for consistency from one example to the next. For example, it can be used to encode a User-Name attribute which contains non-UTF8 data, or a Framed-IP-Address which contains 253 octets of ASCII data. As a result, it MUST NOT be used to create RADIUS Attributes for transport in a RADIUS message.

However, the program correctly encodes the RADIUS attribute fields of "Type", "Length", "Extended-Type", "More", "Reserved", "Vendor-Id", "Vendor-Type", and "Vendor-Length". It encodes RADIUS attribute data types "evs" and TLV. It can therefore be used to encode example attributes from inputs which are humanly readable.

We do not give examples of "malformed" or "invalid attributes". We also note that the examples show format, rather than consistent meaning. A particular Attribute Type code may be used to demonstrate two different formats. In real specifications, attributes have a static definitions based on their type code.

The examples given below are strictly for demonstration purposes only, and do not provide a standard of any kind.

9.1. Extended Type

The following are a series of examples of the "Extended Type" format.

Attribute encapsulating textual data.

```
241.1 "bob"
-> f1 06 01 62 6f 62
```

Attribute encapsulating a TLV with TLV-Type of one (1).

```
241.2 { 1 23 45 }
-> f1 07 02 01 04 23 45
```

Attribute encapsulating two TLVs, one after the other.

```
241.2 { 1 23 45 } { 2 67 89 }
-> f1 0b 02 01 04 23 45 02 04 67 89
```

Attribute encapsulating two TLVs, where the second TLV is itself

encapsulating a TLV.

```
241.2 { 1 23 45 } { 3 { 1 ab cd } }  
-> f1 0d 02 01 04 23 45 03 06 01 04 ab cd
```

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

```
241.2 { 1 23 45 } { 3 { 1 ab cd } { 2 "foo" } }  
-> f1 12 02 01 04 23 45 03 0b 01 04 ab cd 02 05 66 6f 6f
```

Attribute encapsulating a TLV, which in turn encapsulates a TLV, to a depth of 5 nestings.

```
241.1 { 1 { 2 { 3 { 4 { 5 cd ef } } } } }  
-> f1 0f 01 01 0c 02 0a 03 08 04 06 05 04 cd ef
```

Attribute encapsulating an extended Vendor Specific attribute, with Vendor-Id of 1, and Vendor-Type of 4, which in turn encapsulates textual data.

```
241.26.1.4 "test"  
-> f1 0c 1a 00 00 00 01 04 74 65 73 74
```

Attribute encapsulating an extended Vendor Specific attribute, with Vendor-Id of 1, and Vendor-Type of 5, which in turn encapsulates a TLV with TLV-Type of 3, which encapsulates textual data.

```
241.26.1.5 { 3 "test" }  
-> f1 0e 1a 00 00 00 01 05 03 06 74 65 73 74
```

9.2. Long Extended Type

The following are a series of examples of the "Long Extended Type" format.

Attribute encapsulating textual data.

```
245.1 "bob"  
-> f5 07 01 00 62 6f 62
```

Attribute encapsulating a TLV with TLV-Type of one (1).

```
245.2 { 1 23 45 }  
-> f5 08 02 00 01 04 23 45
```

Attribute encapsulating two TLVs, one after the other.


```

          cccccccccc"
-> 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 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
    cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc

```

Attribute encapsulating an extended Vendor Specific attribute, with Vendor-Id of 1, and Vendor-Type of 6, which in turn encapsulates more than 251 octets of data.

As the VSA encapsulates more than 251 octets of data, it is split into two RADIUS attributes. The first attribute has the More field set, and carries the Vendor-Id and Vendor-Type. The second attribute has the More field clear, and carries the rest of the data portion of the VSA. Note that the second attribute does not include the Vendor-Id and Vendor-Type fields.

The "Data" portions are indented for readability.

```

245.26.1.6 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccc
            cccccccccc"
-> 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 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 f5 18 1a 00 bb
bb bb bb bb cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc

```

10. IANA Considerations

This document updates [RFC3575] in that it adds new IANA considerations for RADIUS Attributes. These considerations modify and extend the IANA considerations for RADIUS, rather than replacing them.

The IANA considerations of this document are limited to the "RADIUS Attribute Types" registry. Some Attribute Type values which were previously marked "Reserved" are now allocated, and the registry is extended from a simple 8-bit array to a tree-like structure, up to a maximum depth of 125 nodes. Detailed instructions are given below.

10.1. Attribute Allocations

IANA is requested to move the following Attribute Type values from "Reserved", to "Allocated", with the corresponding names:

- * 241 Extended-Type-1
- * 242 Extended-Type-2
- * 243 Extended-Type-3
- * 244 Extended-Type-4
- * 245 Long-Extended-Type-1
- * 246 Long-Extended-Type-2

These values serve as an encapsulation layer for the new RADIUS Attribute Type tree.

10.2. RADIUS Attribute Type Tree

Each of the Attribute Type values allocated above extends the "RADIUS Attribute Types" to an N-ary tree, via a "dotted number" notation. Allocation of an Attribute Type value "TYPE" using the new Extended type format results in allocation of 255 new Attribute Type values, of format "TYPE.1" through "TYPE.255". Value twenty-six (26) is assigned as "Extended-Vendor-Specific-*". Values "TYPE.241" through "TYPE.255" are marked "Reserved". All other values are "Unassigned".

The initial set of Attribute Type values and names assigned by this document is given below.

* 241	Extended-Attribute-1
* 241.{1-25}	Unassigned
* 241.26	Extended-Vendor-Specific-1
* 241.{27-240}	Unassigned
* 241.{241-255}	Reserved
* 242	Extended-Attribute-2
* 242.{1-25}	Unassigned
* 242.26	Extended-Vendor-Specific-2
* 242.{27-240}	Unassigned
* 243	Extended-Attribute-3
* 242.{241-255}	Reserved
* 243.{1-25}	Unassigned
* 243.26	Extended-Vendor-Specific-3
* 243.{27-240}	Unassigned
* 243.{241-255}	Reserved
* 244	Extended-Attribute-4
* 244.{1-25}	Unassigned
* 244.26	Extended-Vendor-Specific-4
* 244.{27-240}	Unassigned
* 244.{241-255}	Reserved
* 245	Extended-Attribute-5
* 245.{1-25}	Unassigned
* 245.26	Extended-Vendor-Specific-5
* 245.{27-240}	Unassigned
* 245.{241-255}	Reserved
* 246	Extended-Attribute-6
* 246.{1-25}	Unassigned
* 245.26	Extended-Vendor-Specific-6
* 246.{27-240}	Unassigned
* 246.{241-255}	Reserved

As per [RFC5226], the values marked "Unassigned" above are available via for assignment by IANA in future RADIUS specifications. The values marked "Reserved" are reserved for future use.

The Extended-Vendor-Specific spaces (TYPE.26) are for Private Use, and allocations are not managed by IANA.

Allocation of Reserved entries in the extended space requires Standards Action.

All other allocations in the extended space require IETF Review.

10.3. Allocation Instructions

This section defines what actions IANA needs to take when allocating new attributes. Different actions are required when allocating attributes from the standard space, attributes of Extended Type

format, attributes of the "Long Extended Type" format, preferential allocations, attributes of data type TLV, attributes within a TLV, and attributes of other data types.

10.3.1. Requested Allocation from the Standard Space

Specifications can request allocation of an Attribute from within the standard space (e.g. Attribute Type Codes 1 through 255), subject to the considerations of [RFC3575] and this document.

10.3.2. Requested Allocation from the short extended space

Specifications can request allocation of an Attribute which requires the format Extended Type, by specifying the short extended space. In that case, IANA should assign the lowest Unassigned number from the Attribute Type space with the relevant format.

10.3.3. Requested Allocation from the long extended space

Specifications can request allocation of an Attribute which requires the format "Long Extended Type", by specifying the extended space (long). In that case, IANA should assign the lowest Unassigned number from the Attribute Type space with the relevant format.

10.3.4. Allocation Preferences

Specifications which make no request for allocation from a specific Type Space should have Attributes allocated using the following criteria:

- * when the standard space has no more Unassigned attributes, all allocations should be performed from the extended space.
- * specifications which allocate a small number of attributes (i.e. less than ten) should have all allocations made from the standard space.
- * specifications which would allocate a more than twenty percent of the remaining standard space attributes should have all allocations made from the extended space.
- * specifications which request allocation of an attribute of data type TLV should have that attribute allocated from the extended space.
- * specifications which request allocation of an attribute which can transport 253 or more octets of data should have

that attribute allocated from within the long extended space,
We note that Section 6.5, above requires specifications to
request this allocation.

There is otherwise no requirement that all attributes within a
specification be allocated from one type space or another.
Specifications can simultaneously allocate attributes from both the
standard space and the extended space.

10.3.5. Extending the Type Space via TLV Data Type

When specifications request allocation of an attribute of data type
"tlv", that allocation extends the Attribute Type Tree by one more
level. Allocation of an Attribute Type value "TYPE.TLV", with Data
Type TLV, results in allocation of 255 new Attribute Type values, of
format "TYPE.TLV.1" through "TYPE.TLV.255". Values 254-255 are
marked "Reserved". All other values are "Unassigned". Value 26 has
no special meaning.

For example, if a new attribute "Example-TLV" of data type "tlv" is
assigned the identifier "245.1", then the extended tree will be
allocated as below:

```
* 245.1           Example-TLV
* 245.1.{1-253}   Unassigned
* 245.1.{254-255} Reserved
```

Note that this example does not define an "Example-TLV" attribute.

The Attribute Type Tree can be extended multiple levels in one
specification when the specification requests allocation of nested
TLVs, as discussed below.

10.3.6. Allocation within a TLV

Specifications can request allocation of Attribute Type values within
an Attribute of Data Type TLV. The encapsulating TLV can be
allocated in the same specification, or it can have been previously
allocated.

Specifications need to request allocation within a specific Attribute
Type value (e.g. "TYPE.TLV.*"). Allocations are performed from the
smallest Unassigned value, proceeding to the largest Unassigned
value.

Where the Attribute being allocated is of Data Type TLV, the
Attribute Type tree is extended by one level, as given in the

previous section. Allocations can then be made within that level.

10.3.7. Allocation of Other Data Types

Attribute Type value allocations are otherwise allocated from the smallest Unassigned value, proceeding to the largest Unassigned value. e.g. Starting from 241.1, proceeding through 241.255, then to 242.1, through 242.255, etc.

11. Security Considerations

This document defines new formats for data carried inside of RADIUS, but otherwise makes no changes to the security of the RADIUS protocol.

Attacks on cryptographic hashes are well known, and are getting better with time, as discussed in [RFC4270]. The security of the RADIUS protocol is dependent on MD5 [RFC1311], which has security issues as discussed in [RFC6151]. It is not known if the issues described in [RFC6151] apply to RADIUS. For other issues, we incorporate by reference the security considerations of [RFC6158] Section 5.

As with any protocol change, code changes are required in order to implement the new features. These code changes have the potential to introduce new vulnerabilities in the software. Since the RADIUS server performs network authentication, it is an inviting target for attackers. We RECOMMEND that access to RADIUS servers be kept to a minimum.

12. References

12.1. Normative references

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.
- [RFC5176] Chiba, M, et. al., "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176,

January 2008.

[RFC5226] Narten, T. and Alvestrand, H, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, May 2008.

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

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

12.2. Informative references

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

[RFC2868] Zorn, G., et al, " RADIUS Attributes for Tunnel Protocol Support", RFC 2868, June 2000.

[RFC4270] Hoffman, P, and Schneier, B, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[RFC5234] Crocker, D. (Ed.), and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, October 2005.

[RFC6151] Turner, S. and Chen, L., "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

[EDUROAM] Internal Eduroam testing page, data retrieved 04 August 2010.

[ATTR] <http://github.com/alandekok/freeradius-server/tree/master/share/>, data retrieved September 2010.

Acknowledgments

This document is the result of long discussions in the IETF RADEXT working group. The authors would like to thank all of the participants who contributed various ideas over the years. Their feedback has been invaluable, and has helped to make this specification better.

Appendix A - Extended Attribute Generator Program

This section contains "C" program source which can be used for testing. It reads a line-oriented text file, parses it to create RADIUS formatted attributes, and prints the hex version of those attributes to standard output.

The input accepts a grammar similar to that given in Section 9, with some modifications for usability. For example, blank lines are allowed, lines beginning with a '#' character are interpreted as comments, numbers (RADIUS Types, etc.) are checked for minimum / maximum values, and RADIUS Attribute lengths are enforced.

The program is included here for demonstration purposes only, and does not define a standard of any kind.

```
-----
/*
 * Copyright (c) 2010 IETF Trust and the persons identified as
 * authors of the code. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * Neither the name of Internet Society, IETF or IETF Trust, nor the
 * names of specific contributors, may be used to endorse or promote
 * products derived from this software without specific prior written
 * permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
```

```
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* Author: Alan DeKok <aland@networkradius.com>
*/
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>

static int encode_tlv(char *buffer, uint8_t *output, size_t outlen);

static const char *hextab = "0123456789abcdef";

static int encode_data_string(char *buffer,
                              uint8_t *output, size_t outlen)
{
    int length = 0;
    char *p;

    p = buffer + 1;

    while (*p && (outlen > 0)) {
        if (*p == '"') {
            return length;
        }

        if (*p != '\\') {
            *(output++) = *(p++);
            outlen--;
            length++;
            continue;
        }

        switch (p[1]) {
        default:
            *(output++) = p[1];
            break;

        case 'n':
            *(output++) = '\n';
            break;

        case 'r':
            *(output++) = '\r';
```

```
        break;

    case 't':
        *(output++) = '\t';
        break;
    }

    outlen--;
    length++;
}

fprintf(stderr, "String is not terminated\n");
return 0;
}

static int encode_data_tlv(char *buffer, char **endp,
                           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;
    }

    *endp = p + 1;
    *p = '\0';

    p = buffer + 1;
    while (isspace((int) *p)) p++;

    length = encode_tlv(p, output, outlen);
    if (length == 0) return 0;

    return length;
}
```



```
static int encode_data(char *p, uint8_t *output, size_t outlen)
{
    int length;

    if (!isspace((int) *p)) {
        fprintf(stderr, "Invalid character following attribute "
            "definition\n");
        return 0;
    }

    while (isspace((int) *p)) p++;

    if (*p == '{') {
        int sublen;
        char *q;

        length = 0;

        do {
            while (isspace((int) *p)) p++;
            if (!*p) {
                if (length == 0) {
                    fprintf(stderr, "No data\n");
                    return 0;
                }

                break;
            }

            sublen = encode_data_tlv(p, &q, output, outlen);
            if (sublen == 0) return 0;

            length += sublen;
            output += sublen;
            outlen -= sublen;
            p = q;
        } while (*q);

        return length;
    }

    if (*p == '"') {
        length = encode_data_string(p, output, outlen);
        return length;
    }

    length = 0;
    while (*p) {
```

```
    char *c1, *c2;

    while (isspace((int) *p)) p++;

    if (!*p) break;

    if(!(c1 = memchr(hextab, tolower((int) p[0]), 16)) ||
        !(c2 = memchr(hextab, tolower((int) p[1]), 16))) {
        fprintf(stderr, "Invalid data starting at "
            "\"%s\"\n", p);
        return 0;
    }

    *output = ((c1 - hextab) << 4) + (c2 - hextab);
    output++;
    length++;
    p += 2;

    outlen--;
    if (outlen == 0) {
        fprintf(stderr, "Too much data\n");
        return 0;
    }
}

if (length == 0) {
    fprintf(stderr, "Empty string\n");
    return 0;
}

return length;
}

static int decode_attr(char *buffer, char **endptr)
{
    long attr;

    attr = strtol(buffer, endptr, 10);
    if (*endptr == buffer) {
        fprintf(stderr, "No valid number found in string "
            "starting with \"%s\"\n", buffer);
        return 0;
    }

    if (**endptr) {
        fprintf(stderr, "Nothing follows attribute number\n");
        return 0;
    }
}
```

```
    if ((attr <= 0) || (attr > 256)) {
        fprintf(stderr, "Attribute number is out of valid "
            "range\n");
        return 0;
    }

    return (int) attr;
}

static int decode_vendor(char *buffer, char **endptr)
{
    long vendor;

    if (*buffer != '.') {
        fprintf(stderr, "Invalid separator before vendor id\n");
        return 0;
    }

    vendor = strtol(buffer + 1, endptr, 10);
    if (*endptr == (buffer + 1)) {
        fprintf(stderr, "No valid vendor number found\n");
        return 0;
    }

    if (!**endptr) {
        fprintf(stderr, "Nothing follows vendor number\n");
        return 0;
    }

    if ((vendor <= 0) || (vendor > (1 << 24))) {
        fprintf(stderr, "Vendor number is out of valid range\n");
        return 0;
    }

    if (**endptr != '.') {
        fprintf(stderr, "Invalid data following vendor number\n");
        return 0;
    }
    (*endptr)++;

    return (int) vendor;
}

static int encode_tlv(char *buffer, uint8_t *output, size_t outlen)
{
    int attr;
    int length;
    char *p;
```

```
    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    output[0] = attr;
    output[1] = 2;

    if (*p == '.') {
        p++;
        length = encode_tlv(p, output + 2, outlen - 2);
    } else {
        length = encode_data(p, output + 2, outlen - 2);
    }

    if (length == 0) return 0;
    if (length > (255 - 2)) {
        fprintf(stderr, "TLV data is too long\n");
        return 0;
    }

    output[1] += length;

    return length + 2;
}

static int encode_vsa(char *buffer, uint8_t *output, size_t outlen)
{
    int vendor;
    int attr;
    int length;
    char *p;

    vendor = decode_vendor(buffer, &p);
    if (vendor == 0) return 0;

    output[0] = 0;
    output[1] = (vendor >> 16) & 0xff;
    output[2] = (vendor >> 8) & 0xff;
    output[3] = vendor & 0xff;

    length = encode_tlv(p, output + 4, outlen - 4);
    if (length == 0) return 0;
    if (length > (255 - 6)) {
        fprintf(stderr, "VSA data is too long\n");
        return 0;
    }
}
```

```
        return length + 4;
    }

static int encode_evs(char *buffer, uint8_t *output, size_t outlen)
{
    int vendor;
    int attr;
    int length;
    char *p;

    vendor = decode_vendor(buffer, &p);
    if (vendor == 0) return 0;

    attr = decode_attr(p, &p);
    if (attr == 0) return 0;

    output[0] = 0;
    output[1] = (vendor >> 16) & 0xff;
    output[2] = (vendor >> 8) & 0xff;
    output[3] = vendor & 0xff;
    output[4] = attr;

    length = encode_data(p, output + 5, outlen - 5);
    if (length == 0) return 0;

    return length + 5;
}

static int encode_extended(char *buffer,
                           uint8_t *output, size_t outlen)
{
    int attr;
    int length;
    char *p;

    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    output[0] = attr;

    if (attr == 26) {
        length = encode_evs(p, output + 1, outlen - 1);
    } else {
        length = encode_data(p, output + 1, outlen - 1);
    }
    if (length == 0) return 0;
    if (length > (255 - 3)) {
        fprintf(stderr, "Extended Attr data is too long\n");
    }
}
```

```
        return 0;
    }

    return length + 1;
}

static int encode_extended_flags(char *buffer,
                                uint8_t *output, size_t outlen)
{
    int attr;
    int length, total;
    char *p;

    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    /* output[0] is the extended attribute */
    output[1] = 4;
    output[2] = attr;
    output[3] = 0;

    if (attr == 26) {
        length = encode_evs(p, output + 4, outlen - 4);
        if (length == 0) return 0;

        output[1] += 5;
        length -= 5;
    } else {
        length = encode_data(p, output + 4, outlen - 4);
    }
    if (length == 0) return 0;

    total = 0;
    while (1) {
        int sublen = 255 - output[1];

        if (length <= sublen) {
            output[1] += length;
            total += output[1];
            break;
        }

        length -= sublen;

        memmove(output + 255 + 4, output + 255, length);
        memcpy(output + 255, output, 4);

        output[1] = 255;
    }
}
```

```
        output[3] |= 0x80;

        output += 255;
        output[1] = 4;
        total += 255;
    }

    return total;
}

static int encode_rfc(char *buffer, uint8_t *output, size_t outlen)
{
    int attr;
    int length, sublen;
    char *p;

    attr = decode_attr(buffer, &p);
    if (attr == 0) return 0;

    length = 2;
    output[0] = attr;
    output[1] = 2;

    if (attr == 26) {
        sublen = encode_vsa(p, output + 2, outlen - 2);
    } else if ((*p == ' ') || ((attr < 241) || (attr > 246))) {
        sublen = encode_data(p, output + 2, outlen - 2);
    } else {
        if (*p != '.') {
            fprintf(stderr, "Invalid data following "
                "attribute number\n");
            return 0;
        }

        if (attr < 245) {
            sublen = encode_extended(p + 1,
                output + 2, outlen - 2);
        } else {
            /*
             *   Not like the others!
             */
            return encode_extended_flags(p + 1, output, outlen);
        }
    }
    if (sublen == 0) return 0;
}
```

```
    if (sublen > (255 - 2)) {
        fprintf(stderr, "RFC Data is too long\n");
        return 0;
    }

    output[1] += sublen;
    return length + sublen;
}

int main(int argc, char *argv[])
{
    int lineno;
    size_t i, outlen;
    FILE *fp;
    char input[8192], buffer[8192];
    uint8_t output[4096];

    if ((argc < 2) || (strcmp(argv[1], "-") == 0)) {
        fp = stdin;
    } else {
        fp = fopen(argv[1], "r");
        if (!fp) {
            fprintf(stderr, "Error opening %s: %s\n",
                    argv[1], strerror(errno));
            exit(1);
        }
    }

    lineno = 0;
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        char *p = strchr(buffer, '\n');

        lineno++;

        if (!p) {
            if (!feof(fp)) {
                fprintf(stderr, "Line %d too long in %s\n",
                        lineno, argv[1]);
                exit(1);
            }
        } else {
            *p = '\0';
        }

        p = strchr(buffer, '#');
        if (p) *p = '\0';

        p = buffer;
```



```
while (isspace((int) *p)) p++;
if (!*p) continue;

strcpy(input, p);
outlen = encode_rfc(input, output, sizeof(output));
if (outlen == 0) {
    fprintf(stderr, "Parse error in line %d of %s\n",
        lineno, input);
    exit(1);
}

printf("%s -> ", buffer);
for (i = 0; i < outlen; i++) {
    printf("%02x ", output[i]);
}
printf("\n");
}

if (fp != stdin) fclose(fp);

return 0;
}
```

Author's Address

Alan DeKok
Network RADIUS SARL
57bis blvd des Alpes
38240 Meylan
France

Email: aland@networkradius.com
URI: <http://networkradius.com>

Avi Lior
Email: avi.ietf@lior.org

RADIUS EXTensions Working Group
Internet-Draft
Intended status: Experimental
Expires: January 3, 2014

A. Perez-Mendez
R. Marin-Lopez
F. Pereniguez-Garcia
G. Lopez-Millan
University of Murcia
D. Lopez
Telefonica I+D
A. DeKok
Network RADIUS
July 2, 2013

Support of fragmentation of RADIUS packets
draft-perez-radext-radius-fragmentation-06

Abstract

The Remote Authentication Dial-In User Service (RADIUS) protocol is limited to a total packet size of 4096 octets. Provisions exist for fragmenting large amounts of authentication data across multiple packets, via Access-Challenge. No similar provisions exist for fragmenting large amounts of authorization data. This document specifies how existing RADIUS mechanisms can be leveraged to provide that functionality. These mechanisms are largely compatible with existing implementations, and are designed to be invisible to proxies, and "fail-safe" to legacy clients and servers.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 3, 2014.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Requirements Language	4
2. Scope of this document	4
3. Overview	5
4. Fragmentation of packets	7
4.1. Pre-authorization	8
4.2. Post-authorization	12
5. Chunk size	14
6. Allowed large packet size	15
7. Handling special attributes	16
7.1. Proxy-State attribute	16
7.2. State attribute	17
7.3. Service-Type attribute	17
7.4. Rebuilding the original large packet	17
8. New attribute definition	18
8.1. Frag-Status attribute	18
8.2. Proxy-State-Len attribute	19
8.3. Table of attributes	20
9. Operation with proxies	20
9.1. Legacy proxies	20
9.2. Updated proxies	21
10. Security Considerations	22
11. IANA Considerations	23
12. References	23
12.1. Normative References	23
12.2. Informative References	24
Authors' Addresses	24

1. Introduction

The RADIUS [RFC2865] protocol carries authentication, authorization, and accounting information between a Network Access Server (NAS) and an Authentication Server (AS). Information is exchanged between the NAS and the AS through RADIUS packets. Each RADIUS packet is composed of a header, and zero or more attributes, up to a maximum packet size of 4096 octets. The protocol is a request/response protocol, as described in the operational model ([RFC6158], Section 3.1).

The above packet size limitation mean that peers desiring to send large amounts of data must fragment it across multiple packets. For example, RADIUS-EAP [RFC3579] defines how an EAP exchange occurs across multiple Access-Request / Access-Challenge sequences. No such exchange is possible for accounting or authorization data. [RFC6158] Section 3.1 suggests that exchanging large amounts authorization data is unnecessary in RADIUS. Instead, the data should be referenced by name. This requirement allows large policies to be pre-provisioned, and then referenced in an Access-Accept. In some cases, however, the authorization data sent by the server is large and highly dynamic. In other cases, the NAS needs to send large amounts of authorization data to the server. Both of these cases are un-met by the requirements in [RFC6158]. As noted in that document, the practical limit on RADIUS packet sizes is governed by the Path MTU (PMTU), which may be significantly smaller than 4096 octets. The combination of the two limitations means that there is a pressing need for a method to send large amounts of authorization data between NAS and AS, with no accompanying solution.

[RFC6158] recommends three approaches for the transmission of large amount of data within RADIUS. However, they are not applicable to the problem statement of this document for the following reasons:

- o The first approach does not talk about large amounts of data sent from the NAS to a server. Leveraging EAP (request/challenge) to send the data is not feasible, as EAP already fills packet to PMTU, and not all authentications use EAP. Moreover, as noted for NAS-Filter-Rule ([RFC4849]), this approach does entirely solve the problem of sending large amounts of data from a server to a NAS.
- o The second approach is not usable either, as using names rather than values is difficult when the nature of the data to be sent is highly dynamic (e.g. SAML sentences or NAS-Filter-Rule attributes). URLs could be used as a pointer to the location of the actual data, but their use would require them to be (a) dynamically created and modified, (b) securely accessed and (c) accessible from remote systems. Satisfying these constraints

would require the modification of several networking systems (e.g. firewalls and web servers). Furthermore, the set up of an additional trust infrastructure (e.g. PKI) would be required to allow secure retrieving of the information from the web server.

- o PMTU discovery does not solve the problem, as it does not allow to send data larger than the minimum of (PMTU or 4096) octets.

This document provides a mechanism to allow RADIUS peers to exchange large amounts of authorization data exceeding the 4096 octet limit, by fragmenting it across several client/server exchanges. The proposed solution does not impose any additional requirements to the RADIUS system administrators (e.g. need to modify firewall rules, set up web servers, configure routers, or modify any application server). It maintains compatibility with intra-packet fragmentation mechanisms (like those defined in [RFC3579] or in [I-D.ietf-radext-radius-extensions]). It is also transparent to existing RADIUS proxies, which do not implement this specification. The only systems needing to implement the draft are the ones which either generate, or consume the fragmented data being transmitted. Intermediate proxies just pass the packets without changes. Nevertheless, if a proxy supports this specification, it MAY re-assemble the data in order to either examine and/or modify it.

1.1. Requirements Language

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

2. Scope of this document

This specification describes how a RADIUS client and a RADIUS server can exchange large amounts of data exceeding the 4096 octet limit. Specifically, its scope is limited to the exchange of authorization data, as other exchanges do not require of such a mechanism. In particular, authentication exchanges have already been defined to overcome this limitation (e.g. RADIUS-EAP). Moreover, as they represent the most critical part of a RADIUS conversation, its preferable to not introduce any modification to their operation that may affect existing equipment.

There is no need to fragment accounting packets either. While the accounting process can send large amounts of data, that data is typically composed of many small updates. That is, there is no demonstrated need to send indivisible blocks of more than 4K of data. The need to send large amounts of data per user session often

originates from the need for flow-based accounting. In this use-case, the client may send accounting data for many thousands of flows, where all those flows are tied to one user session. The existing Acct-Multi-Session-Id attribute defined in [RFC2866] Section 5.11 has been proven to work here.

Similarly, there is no need to fragment CoA packets. Instead, the CoA client MUST send a CoA-Request packet containing session identification attributes, along with Service-Type = Additional-Authorization, and a State attribute. Implementations not supporting fragmentation will respond with a CoA-NAK, and an Error-Cause of Unsupported-Service.

Implementations supporting this specification may not be able to change authorization data for a particular session. In that case, they MUST respond with a CoA-NAK, as above. Otherwise, the implementation MUST start fragmentation via Access-Request, using the methods defined here.

The above requirement solves a number of issues. It clearly separates session identification from authorization. Without this separation, it is difficult to both identify a session, and change its authorization using the same attribute. It also ensures that the authorization process is the same for initial authentication, and for CoA.

When a sessions authorization is changed, the CoA server MUST continue the existing service until the new authorization parameters are applied. The change of service SHOULD be done atomically. If the CoA server is unable to apply the new authorization, it MUST terminate the user session.

3. Overview

Authorization exchanges can occur either before or after end user authentication has been completed. An authorization exchange before authentication allows a RADIUS client to provide the RADIUS server with information that MAY modify how the authentication process will be performed (e.g. it MAY affect the selection of the EAP method). An authorization exchange after authentication allows the RADIUS server to provide the RADIUS client with information about the end user, the results of the authentication process and/or obligations to be enforced. In this specification we refer to the "pre-authentication" as the exchange of authorization information before the end user authentication has started, while the term "post-authentication" is used to refer to an authorization exchange happening after this authentication process.

In this specification we refer to the "size limit" as the practical limit on RADIUS packet sizes. This limit is the minimum of 4096 octets, and the current PMTU. We define below a method which uses Access-Request and Access-Accept in order to exchange fragmented data. The NAS and server exchange a series of Access-Request / Access-Accept packets, until such time as all of the fragmented data has been transported. Each packet contains a Frag-Status attribute which lets the other party know if fragmentation is desired, ongoing, or finished. Each packet may also contain the fragmented data, or instead be an "ACK" to a previous fragment from the other party. Each Access-Request contains a User-Name attribute, allowing it to be proxied if necessary. Each Access-Request may also contain a State attribute, which serves to tie it to a previous Access-Accept. Each Access-Accept contains a State attribute, for use by the NAS in a later Access-Request. Each Access-Accept contains a Service-Type indicating that the service being provided is fragmentation, and that the Access-Accept should not be interpreted as providing network access to the end user.

When a RADIUS client or server need to send data that exceeds the size limit, the mechanism proposed in this document is used. Instead of encoding one large RADIUS packet, a series of smaller RADIUS packets of the same type are encoded. Each smaller packet is called a "chunk" in this specification, in order to distinguish it from traditional RADIUS packets. The encoding process is a simple linear walk over the attributes to be encoded. This walk preserves the order of the attributes, as required by [RFC2865]. The number of attributes encoded in a particular chunk depends on the size limit, the size of each attribute, the number of proxies between client and server, and the overhead for fragmentation signalling attributes. Specific details are given in Section 5. A new attribute called Frag-Status (Section 8.1) signals the fragmentation status.

After the first chunk is encoded, it is sent to the other party. The packet is identified as a chunk via the Frag-Status attribute. The other party then requests additional chunks, again using the Frag-Status attribute. This process is repeated until all the attributes have been sent from one party to the other. When all the chunks have been received, the original list of attributes is reconstructed and processed as if it had been received in one packet.

When multiple chunks are sent, a special situation may occur for Extended Type attributes as defined in [I-D.ietf-radext-radius-extensions]. The fragmentation process may split a fragmented attribute across two or more chunks, which is not permitted by that specification. We address this issue by defining a new field in the Reserved field of the "Long Extended Type" attribute format. This field is one bit in size, and is called "T" for

Truncation. It indicates that the attribute is intentionally truncated in this chunk, and is to be continued in the next chunk of the sequence. The combination of the flags "M" and "T" indicates that the attribute is fragmented (flag M), but that all the fragments are not available in this chunk (flag T).

This last situation is expected to be the most common occurrence in chunks. Typically, packet fragmentation will occur as a consequence of a desire to send one or more large (and therefore fragmented) attributes. The large attribute will likely be split into two or more pieces. Where chunking does not split a fragmented attribute, no special treatment is necessary.

The setting of the "T" flag is the only case where the chunking process affects the content of an attribute. Even then, the "Value" fields of all attributes remain unchanged. Any per-packet security attributes such as Message-Authenticator are calculated for each chunk independently. There are neither integrity nor security checks performed on the "original" packet.

Each RADIUS packet sent or received as part of the chunking process MUST be a valid packet, subject to all format and security requirements. This requirement ensures that a "transparent" proxy not implementing this specification can receive and send compliant packets. That is, a proxy which simply forwards packets without detailed examination or any modification will be able to proxy "chunks".

4. Fragmentation of packets

When the NAS or the AS desires to send a packet that exceeds the size limit, it is split into chunks and sent via multiple client/server exchanges. The exchange is indicated via the Frag-Status attribute, which has value More-Data-Pending for all but the last chunk of the series. The chunks are tied together via the State attribute.

The following sections describe how to perform fragmentation for packets from the NAS to the server, followed by packets from the server to the NAS. We give the packet type, along with a RADIUS Identifier, to indicate that requests and responses are connected. We then give a list of attributes. We do not give values for most attributes, as we wish to concentrate on the fragmentation behaviour, rather than packet contents. Attribute values are given for attributes relevant to the fragmentation process. Where "long extended" attributes are used, we indicate the M (More) and T (Truncation) flags as optional square brackets after the attribute name. As no "long extended" attributes have yet been defined, we use

example attributes, named as "Example-Long-1", etc. The maximum chunk size is established in term of number of attributes (11), for sake of simplicity.

4.1. Pre-authorization

When the client needs to send a large amount of data to the server, the data to be sent is split into chunks and sent to the server via multiple Access-Request / Access-Accept exchanges. The example below shows this exchange.

The following is an Access-Request which the NAS intends to send to a server. However, due to a combination of issues (PMTU, large attributes, etc.), the content does not fit into one Access-Request packet.

```
Access-Request
  User-Name
  User-Password
  Calling-Station-Id
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
```

Figure 1: Desired Access-Request

The NAS therefore must send the attributes listed above in a series of chunks. The first chunk contains eight (8) attributes from the original Access-Request, and a Frag-Status attribute. Since last attribute is "Example-Long-1" with the "M" flag set, the chunking process also sets the "T" flag in that attribute. The Access-Request is sent with a RADIUS Identifier field having value 23. The Frag-Status attribute has value More-Data-Pending, to indicate that the NAS wishes to send more data in a subsequent Access-Request. The NAS also adds a Service-Type attribute, which indicates that it is part of the chunking process. The packet is signed with the Message-Authenticator attribute, completing the maximum number of attributes (11).

```
Access-Request (ID = 23)
  User-Name
  User-Password
  Calling-Station-Id
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [MT]
  Frag-Status = More-Data-Pending
  Service-Type = Additional-Authorization
  Message-Authenticator
```

Figure 2: Access-Request (chunk 1)

Compliant servers receiving this packet will see the Frag-Status attribute, and suspend all authorization and authentication handling until all of the chunks have been received. Non-compliant servers should also see the Service-Type requesting provisioning for an unknown service, and return Access-Reject. Other non-compliant servers may return an Access-Reject, Access-Challenge, or an Access-Accept with a particular Service-Type. Compliant NAS implementations MUST treat these responses as if they had received Access-Reject instead.

Compliant servers who wish to receive all of the chunks will respond with the following packet. The value of the State here is arbitrary, and serves only as a unique token for example purposes. We only note that it MUST be globally and temporally unique.

```
Access-Accept (ID = 23)
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xabc00001
  Message-Authenticator
```

Figure 3: Access-Accept (chunk 1)

The NAS will see this response, and use the RADIUS Identifier field to associate it with an ongoing chunking session. Compliant NASes will then continue the chunking process. Non-compliant NASes will never see a response such as this, as they will never send a Frag-Status attribute. The Service-Type attribute is included in the Access-Accept in order to signal that the response is part of the chunking process. This packet therefore does not provision any network service for the end user.

The NAS continues the process by sending the next chunk, which

includes an additional six (6) attributes from the original packet. It again includes the User-Name attribute, so that non-compliant proxies can process the packet. It sets the Frag-Status attribute to More-Data-Pending, as more data is pending. It includes a Service-Type for reasons described above. It includes the State attribute from the previous Access-accept. It signs the packet with Message-Authenticator, as there are no authentication attributes in the packet. It uses a new RADIUS Identifier field.

```
Access-Request (ID = 181)
  User-Name
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [MT]
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xabc000001
  Message-Authenticator
```

Figure 4: Access-Request (chunk 2)

Compliant servers receiving this packet will see the Frag-Status attribute, and look for a State attribute. Since one exists and it matches a State sent in an Access-Accept, this packet is part of a chunking process. The server will associate the attributes with the previous chunk. Since the Frag-Status attribute has value More-Data-Request, the server will respond with an Access-Accept as before. It MUST include a State attribute, with a value different from the previous Access-Accept. This State MUST again be globally and temporally unique.

```
Access-Accept (ID = 181)
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xdef00002
  Message-Authenticator
```

Figure 5: Access-Accept (chunk 2)

The NAS will see this response, and use the RADIUS Identifier field to associate it with an ongoing chunking session. The NAS continues the chunking process by sending the next chunk, with the final attribute(s) from the original packet, and again includes the original User-Name attribute. The Frag-Status attribute is not included in the next Access-Request, as no more chunks are available

for sending. The NAS includes the State attribute from the previous Access-accept. It signs the packet with Message-Authenticator, as there are no authentication attributes in the packet. It again uses a new RADIUS Identifier field.

```
Access-Request (ID = 241)
  User-Name
  Example-Long-2
  State = 0xdef00002
  Message-Authenticator
```

Figure 6: Access-Request (chunk 3)

On reception of this last chunk, the server matches it with an ongoing session via the State attribute, and sees that there is no Frag-Status attribute present. It then process the received attributes as if they had been sent in one RADIUS packet. See Section 7.4 for further details of this process. It generates the appropriate response, which can be either Access-Accept or Access-Reject. In this example, we show an Access-Accept. The server MUST send a State attribute, which permits link the received data with the authentication process.

```
Access-Accept (ID = 241)
  State = 0x98700003
  Message-Authenticator
```

Figure 7: Access-Accept (chunk 3)

The above example shows in practice how the chunking process works. We re-iterate the implementation and security requirements here.

Each chunk is a valid RADIUS packet, and all RADIUS format and security requirements MUST be followed before any chunking process is applied.

Every chunk except for the last one from a NAS MUST include a Frag-Status attribute, with value More-Data-Pending. The last chunk MUST NOT contain a Frag-Status attribute. Each chunk except for the last from a NAS MUST include a Service-Type attribute, with value Additional-Authorization. Each chunk MUST include a User-Name attribute, which MUST be identical in all chunks. Each chunk except for the first one from a NAS MUST include a State attribute, which MUST be copied from a previous Access-Accept.

Each Access-Accept MUST include a State attribute. The value for this attribute MUST change in every new Access-Accept, and MUST be globally and temporally unique.

4.2. Post-authorization

When the AS wants to send a large amount of authorization data to the NAS after authentication, the operation is very similar to the pre-authorization one. The presence of Service-Type = Additional-Authorization attribute ensures that a NAS not supporting this specification will treat that unrecognized Service-Type as though an Access-Reject had been received instead ([RFC2865] Section 5.6). If the original large Access-Accept packet contained a Service-Type attribute, it will be included with its original value in the last transmitted chunk, to avoid confusion with the one used for fragmentation signalling.

Client supporting this specification MUST include a Frag-Status = Fragmentation-Supported attribute in the first Access-Request sent to the server, in order to indicate they would accept fragmented data from the sever. This is not required if pre-authorization process was carried out, as it is implicit.

The following is an Access-Accept which the AS intends to send to a client. However, due to a combination of issues (PMTU, large attributes, etc.), the content does not fit into one Access-Accept packet.

```
Access-Accept
  User-Name
  EAP-Message
  Service-Type(Login)
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
```

Figure 8: Desired Access-Accept

The AS therefore must send the attributes listed above in a series of chunks. The first chunk contains eight (7) attributes from the original Access-Accept, and a Frag-Status attribute. Since last attribute is "Example-Long-1" with the "M" flag set, the chunking process also sets the "T" flag in that attribute. The Access-Accept

is sent with a RADIUS Identifier field having value 30 corresponding to a previous Access-Request not depicted. The Frag-Status attribute has value More-Data-Pending, to indicate that the AS wishes to send more data in a subsequent Access-Accept. The AS also adds a Service-Type attribute with value Additional-Authorization, which indicates that it is part of the chunking process. Note that the original Service-Type is not included in this chunk. Finally, a State attribute is included to allow matching subsequent requests with this conversation, and the packet is signed with the Message-Authenticator attribute, completing the maximum number of attributes of 11.

```
Access-Accept (ID = 30)
  User-Name
  EAP-Message
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [MT]
  Frag-Status = More-Data-Pending
  Service-Type = Additional-Authorization
  State = 0xcba00004
  Message-Authenticator
```

Figure 9: Access-Accept (chunk 1)

Compliant clients receiving this packet will see the Frag-Status attribute, and suspend all authorization and authentication handling until all of the chunks have been received. Non-compliant clients should also see the Service-Type indicating the provisioning for an unknown service, and will treat it as an Access-Reject.

Clients who wish to receive all of the chunks will respond with the following packet, where the value of the State attribute is taken from the received Access-Accept. They also include the User-Name attribute so that non-compliant proxies can process the packet.

```
Access-Request (ID = 131)
  User-Name
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xcba00004
  Message-Authenticator
```

Figure 10: Access-Request (chunk 1)

The AS receives this request, and uses the State attribute to associate it with an ongoing chunking session. Compliant ASes will

then continue the chunking process. Non-compliant ASes will never see a response such as this, as they will never send a Frag-Status attribute.

The AS continues the chunking process by sending the next chunk, with the final attribute(s) from the original packet. The value of the Identifier field is taken from the received Access-Request. A Frag-Status attribute is not included in the next Access-Accept, as no more chunks are available for sending. The AS includes an State attribute to allow the client to send additional authorization data. The original Service-Type attribute is included in this final chunk.

```
Access-Accept (ID = 131)
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1 [M]
  Example-Long-1
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
  Service-Type = Login
  State = 0xfda000005
  Message-Authenticator
```

Figure 11: Access-Accept (chunk 2)

On reception of this last chunk, the client matches it with an ongoing session via the Identifier field, and sees that there is no Frag-Status attribute present. It then processes the received attributes as if they had been sent in one RADIUS packet. See Section 7.4 for further details of this process.

5. Chunk size

In an ideal scenario, each intermediate chunk would be exactly the size limit in length. In this way, the number of round trips required to send a large packet would be optimal. However, this is not possible for several reasons.

1. RADIUS attributes have a variable length, and must be included completely in a chunk. Thus, it is possible that, even if there is some free space in the chunk, it is not enough to include the next attribute. This can generate up to 254 octets of spare space on every chunk.
2. RADIUS fragmentation requires the introduction of some extra attributes for signalling. Specifically, a Frag-Status attribute

(7 octets) is included on every chunk of a packet, except the last one. A RADIUS State attribute (from 3 to 255 octets) is also included in most chunks, to allow the server to bind an Access-Request with a previous Access-Challenge. User-Name attributes (from 3 to 255 octets) are introduced on every chunk the client sends as they are required by the proxies to route the packet to its destination. Together, these attributes can generate from up to 13 to 517 octets of signalling data, reducing the amount of payload information that can be sent on each chunk.

3. RADIUS packets SHOULD be adjusted to avoid exceeding the network MTU. Otherwise, IP fragmentation may occur, having undesirable consequences. Hence, maximum chunk size would be decreased from 4096 to the actual MTU of the network.
4. The inclusion of Proxy-State attributes by intermediary proxies can decrease the availability of usable space into the chunk. This is described with further detail in Section 7.1.

6. Allowed large packet size

There are no provisions for signalling how much data is to be sent via the fragmentation process as a whole. It is difficult to define what is meant by the "length" of any fragmented data. That data can be multiple attributes, which includes RADIUS attribute header fields. Or it can be one or more "large" attributes (more than 256 octets in length). Proxies can also filter these attributes, to modify, add, or delete them and their contents. These proxies act on a "packet by packet" basis, and cannot know what kind of filtering actions they take on future packets. As a result, it is impossible to signal any meaningful value for the total amount of additional data.

Unauthenticated clients are permitted to trigger the exchange of large amounts of fragmented data between the NAS and the AS, having the potential to allow Denial of Service (DoS) attacks. An attacker could initiate a large number of connections, each of which requests the server to store a large amount of data. This data could cause memory exhaustion on the server, and result in authentic users being denied access. It is worth noting that authentication mechanisms are already designed to avoid exceeding the size limit.

Hence, implementations of this specification MUST limit the total amount of data they send and/or receive via this specification. It is RECOMMENDED that the limits be set to a few tens of kilooctets. Any more than this may turn RADIUS into a generic transport protocol, which is undesired. It is RECOMMENDED that this limit be exposed to

administrators, so that it can be changed if necessary.

Implementations of this specification MUST limit the total number of round trips used during the fragmentation process. It is RECOMMENDED that the number of round trips be limited to twenty (20). Any more than this may indicate an implementation error, misconfiguration, or a denial of service (DoS) attack. It is RECOMMENDED that this limit be exposed to administrators, so that it can be changed if necessary.

7. Handling special attributes

7.1. Proxy-State attribute

RADIUS proxies may introduce Proxy-State attributes into any Access-Request packet they forward. Should they cannot add this information to the packet, they may silently discard forwarding it to its destination, leading to DoS situations. Moreover, any Proxy-State attribute received by a RADIUS server in an Access-Request packet MUST be copied into the reply packet to it. For these reasons, Proxy-State attributes require a special treatment within the packet fragmentation mechanism.

When the RADIUS server replies to an Access-Request packet as part of a conversation involving a fragmentation (either a chunk or a request for chunks), it MUST include every Proxy-State attribute received into the reply packet. This means that the server MUST take into account the size of these Proxy-State attributes in order to calculate the size of the next chunk to be sent.

However, while a RADIUS server will always know how many space MUST be left on each reply packet for Proxy-State attributes (as they are directly included by the RADIUS server), a RADIUS client cannot know this information, as Proxy-State attributes are removed from the reply packet by their respective proxies before forwarding them back. Hence, clients need a mechanism to discover the amount of space required by proxies to introduce their Proxy-State attributes. In the following we describe a new mechanism to perform such a discovery:

1. When a RADIUS client does not know how many space will be required by intermediate proxies for including their Proxy-State attributes, it SHOULD start using a conservative value (e.g. 1024 octets) as the chunk size.
2. When the RADIUS server receives a chunk from the client, it can calculate the total size of the Proxy-State attributes that have been introduced by intermediary proxies along the path. This

information MUST be returned to the client in the next reply packet, encoded into a new attribute called Proxy-State-Len.

3. The RADIUS client reacts upon the reception of this attribute by adjusting the maximum size for the next chunk accordingly.

7.2. State attribute

This RADIUS fragmentation mechanism makes use of the State attribute to link all the chunks belonging to the same fragmented packet. However, some considerations are required when the RADIUS server is fragmenting a packet that already contains a State attribute for other purposes not related with the fragmentation. If the procedure described in Section 4 is followed, two different State attributes could be included into a single chunk, incurring into two problems. First, [RFC2865] explicitly forbids that more than one State attribute appears into a single packet.

A straightforward solution consists on making the RADIUS server to send the original State attribute into the last chunk of the sequence (attributes can be re-ordered as specified in [RFC2865]). As the last chunk (when generated by the RADIUS server) does not contain any State attribute due to the fragmentation mechanism, both situations described above are avoided.

Something similar happens when the RADIUS client has to send a fragmented packet that contains a State attribute on it. The client MUST assure that this original State is included into the first chunk sent to the server (as this one never contains any State attribute due to fragmentation).

7.3. Service-Type attribute

This RADIUS fragmentation mechanism makes use of the Service-Type attribute to indicate an Access-Accept packet is not granting access to the service yet, since additional authorization exchange needs to be performed. Similarly to the State attribute, the RADIUS server has to send the original Service-Type attribute into the last Access-Accept of the RADIUS conversation to avoid ambiguity.

7.4. Rebuilding the original large packet

The RADIUS client stores the RADIUS attributes received on each chunk in order to be able to rebuild the original large packet after receiving the last chunk. However, some of these received attributes MUST NOT be stored in this list, as they have been introduced as part of the fragmentation signalling and hence, they are not part of the original packet.

- o State (except the one in the last chunk, if present)
- o Service-Type = Additional-Authorization
- o Frag-Status
- o Proxy-State-Len

Similarly, the RADIUS server MUST NOT store the following attributes as part of the original large packet:

- o State (except the one in the first chunk, if present)
- o Frag-Status
- o Proxy-State (except the ones in the last chunk)
- o User-Name (except the one in the first chunk)

8. New attribute definition

This document proposes the definition of two new extended type attributes, called Frag-Status and Proxy-State-Len. The format of these attributes follows the indications for an Extended Type attribute defined in [I-D.ietf-radext-radius-extensions].

8.1. Frag-Status attribute

This attribute is used for fragmentation signalling, and its meaning depends on the code value transported within it. The following figure represents the format of the Frag-Status attribute.

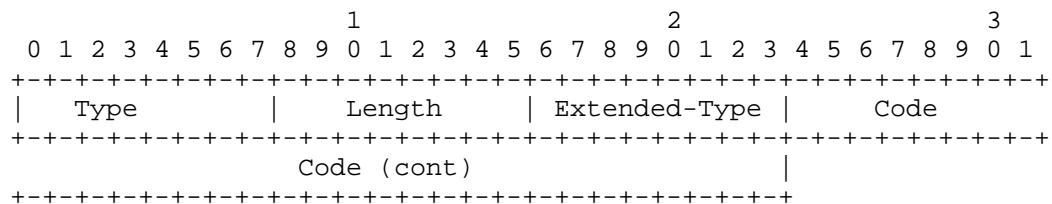


Figure 12: Frag-Status format

Type

To be assigned (TBA)

Length

7

Extended-Type

To be assigned (TBA).

Code

4 byte. Integer indicating the code. The values defined in this specifications are:

- 0 - Reserved
- 1 - Fragmentation-Supported
- 2 - More-Data-Pending
- 3 - More-Data-Request

This attribute MAY be present in Access-Request, Access-Challenge and Access-Accept packets. It MUST not be included in Access-Reject packets.

8.2. Proxy-State-Len attribute

This attribute indicates to the RADIUS client the length of the Proxy-State attributes received by the RADIUS server. This information is useful to adjust the length of the chunks sent by the RADIUS client. The format of this Proxy-State-Len attribute is the following:

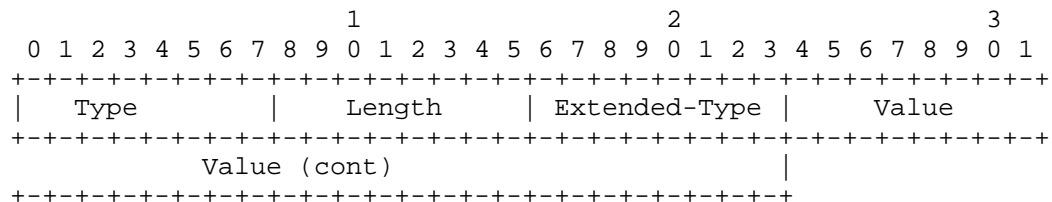


Figure 13: Proxy-State-Len format

Type

To be assigned (TBA)

Length

7

Extended-Type

To be assigned (TBA).

Value

4 octets. Total length (in octets) of received Proxy-State attributes (including headers).

This attribute MAY be present in Access-Challenge and Access-Accept packets. It MUST not be included in Access-Request or Access-Reject packets.

8.3. Table of attributes

The following table shows the different attributes defined in this document related with the kind of RADIUS packets where they can be present.

Attribute Name	Kind of packet			
	Req	Acc	Rej	Cha
Frag-Status	0-1	0-1	0	0-1
Proxy-State-Len	0	0-1	0	0-1

Figure 14

9. Operation with proxies

The fragmentation mechanism defined above is designed to be transparent to legacy proxies, as long as they do not want to modify any fragmented attribute. Nevertheless, updated proxies supporting this specification can even modify fragmented attributes.

9.1. Legacy proxies

As every chunk is indeed a RADIUS packet, legacy proxies treat them as the rest of packets, routing them to their destination. Proxies can introduce Proxy-State attributes to Access-Request packets, even if they are indeed chunks. This will not affect how fragmentation is managed. The server will include all the received Proxy-State attributes into the generated response, as described in [RFC2865].

Hence, proxies do not distinguish between a regular RADIUS packet and a chunk.

9.2. Updated proxies

Updated proxies can interact with clients and servers in order to obtain the complete large packet before start forwarding it. In this way, proxies can manipulate (modify and/or remove) any attribute of the packet, or introduce new attributes, without worrying about crossing the boundaries of the chunk size. Once the manipulated packet is ready, it is sent to the original destination using the fragmentation mechanism (if required). The following example shows how an updated proxy interacts with the NAS to obtain a large Access-Request packet, modify an attribute resulting into a even more large packet, and interacts with the AS to complete the transmission of the modified packet.

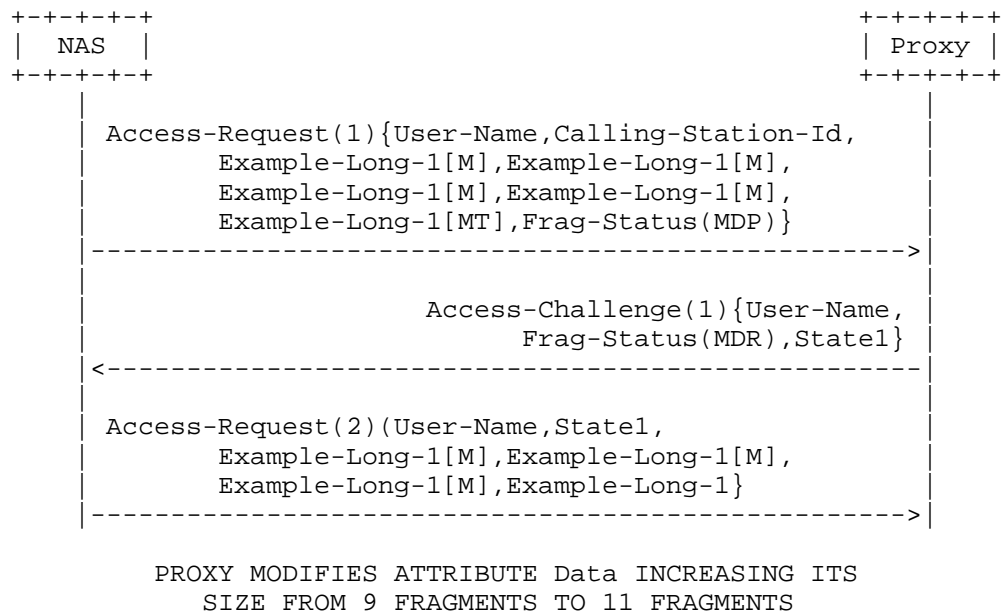


Figure 15: Updated proxy interacts with NAS

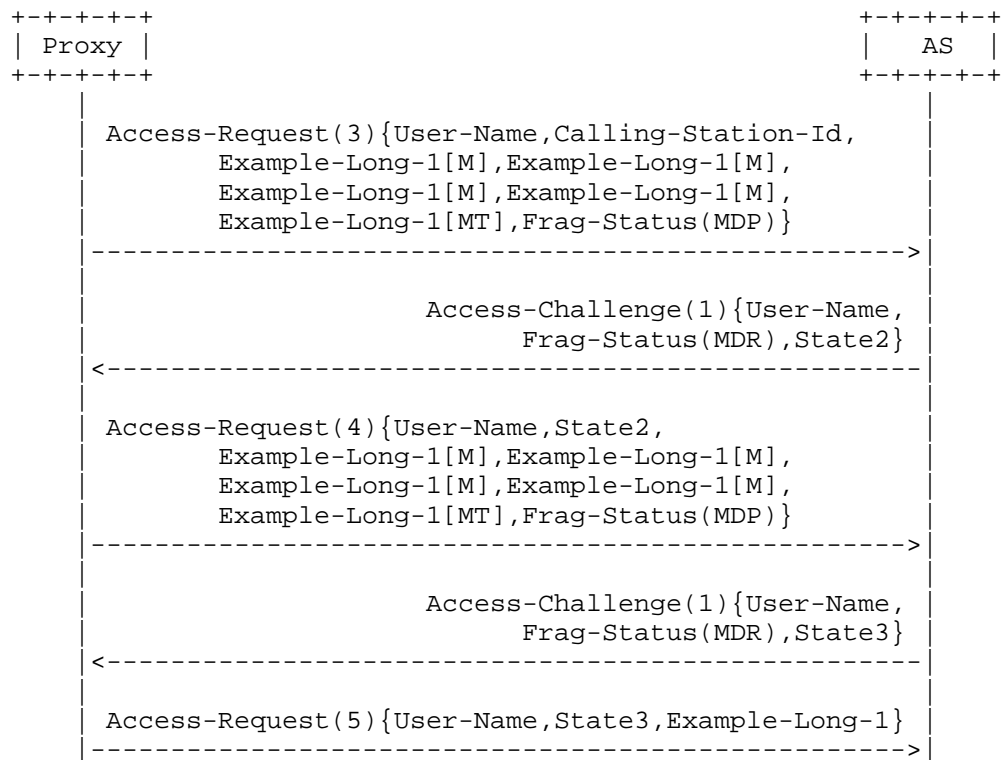


Figure 16: Updated proxy interacts with AS

10. Security Considerations

As noted in many earlier specifications ([RFC5080], [RFC6158], etc.) RADIUS security is problematic. This specification changes nothing related to the security of the RADIUS protocol. It requires that all Access-Request packets associated with fragmentation are signed using the existing Message-Authenticator attribute. This signature prevents forging and replay, to the limits of the existing security.

The ability to send bulk data from one party to another creates new security considerations. Clients and servers may have to store large amounts of data per session. The amount of this data can be significant, leading to the potential for resource exhaustion. We therefore suggest that implementations limit the amount of bulk data stored per session. The exact method for this limitation is implementation-specific. Section 6 gives some indications on what could be a reasonable limits.

The bulk data can often be pushed off to storage methods other than the memory of the RADIUS implementation. For example, it can be stored in an external database, or in files. This approach mitigates the resource exhaustion issue, as servers today already store large amounts of accounting data.

11. IANA Considerations

The authors request that Attribute Types and Attribute Values defined in this document be registered by the Internet Assigned Numbers Authority (IANA) from the RADIUS namespaces as described in the "IANA Considerations" section of [RFC3575], in accordance with BCP 26 [RFC5226]. For RADIUS packets, attributes and registries created by this document IANA is requested to place them at <http://www.iana.org/assignments/radius-types>.

This document defines the following RADIUS messages:

- o Frag-Status
- o Proxy-State-Len

Additionally, allocation of a new Service-Type value for "Additional-Authorization" is requested.

12. References

12.1. Normative References

- [I-D.ietf-radext-radius-extensions]
DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions",
draft-ietf-radext-radius-extensions-13 (work in progress),
February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575,

July 2003.

- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.
- [RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6158] DeKok, A. and G. Weber, "RADIUS Design Guidelines", BCP 158, RFC 6158, March 2011.

12.2. Informative References

- [RFC4849] Congdon, P., Sanchez, M., and B. Aboba, "RADIUS Filter Rule Attribute", RFC 4849, April 2007.

Authors' Addresses

Alejandro Perez-Mendez (Ed.)
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 46 44
Email: alex@um.es

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 85 01
Email: rafa@um.es

Fernando Pereniguez-Garcia
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 78 82
Email: pereniguez@um.es

Gabriel Lopez-Millan
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 85 04
Email: gabilm@um.es

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 84
Madrid, 28006
Spain

Phone: +34 913 129 041
Email: diego@tid.es

Alan DeKok
Network RADIUS
15 av du Granier
Meylan, 38240
France

Phone: +34 913 129 041
Email: aland@networkradius.com
URI: <http://networkradius.com>

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

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Requirements Language	3
2. Definitions	3
2.1. Acct-Traffic-Class attribute	3
2.1.1. Acct-Traffic-Class-Name attribute	4
2.1.2. Acct-Traffic-Class-Input-Octets attribute	5
2.1.3. Acct-Traffic-Class-Output-Octets attribute	5
2.1.4. Acct-Traffic-Class-Input-Packets attribute	5
2.1.5. Acct-Traffic-Class-Output-Packets attribute	6
2.2. URN values for attribute Acct-Traffic-Class-Name	6
3. Example	7
4. Attribute Occurrence Table	8
5. Security Considerations	9
6. IANA Considerations	9
7. Normative References	9

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.

2.1.1.1. 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 Accounting-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 Accounting-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 Accounting-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 Accounting-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-accounting: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 Occurrence Table

This table lists the allowed occurrences 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 Occurrence

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 unencryptedly.

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>.

Radext Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2013

L. Yeh
Huawei Technologies
M. Boucadair
France Telecom
October 15, 2012

RADIUS Accounting Extensions for Traffic Statistics
draft-yeh-radext-ext-traffic-statistics-04

Abstract

This document specifies the RADIUS extensions of attributes for the traffic statistics with different type, which can be used to support the differentiated accounting policies and traffic recording on the AAA server.

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 April 18, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology and Conventions	4
3. Deployment Scenarios	4
4. Acct-Traffic-Statistics attribute	4
4.1. Container attribute of Acct-Traffic-Statistics	5
4.2. Contained attribute of Acct-Traffic-Statistics	6
4.2.1. Acct-Traffic-Statistics.Stack-Type	6
4.2.2. Acct-Traffic-Statistics.Input-Octets	7
4.2.3. Acct-Traffic-Statistics.Output-Octets	7
4.2.4. Acct-Traffic-Statistics.Input-Packets	8
4.2.5. Acct-Traffic-Statistics.Output-Packets	9
4.2.6. Acct-Traffic-Statistics.DSCP-Type	10
5. Table of Attribute	10
6. Diameter Considerations	11
7. Security Considerations	11
8. IANA Considerations	11
9. Acknowledgements	11
10. References	12
10.1. Normative References	12
10.2. Informative References	12
Authors' Addresses	13

1. Introduction

RADIUS has been widely used as the centralized authentication, authorization and subscriber management method for the service provision in Broadband access network. [RFC3162], [RFC4818] and [I-D.ietf-radext-ipv6-access] have specified some attributes to support the service provision for IPv6 access. In the meantime, RADIUS is also a protocol for carrying accounting information between a Network Access Server (NAS) and a shared accounting server. In the scenarios of dual-stack or any other IPv6 migration technologies, there is a demand to report the separated IPv4 & IPv6 traffic statistics for the differential accounting and traffic recording.

[BBF TR-187], whose purpose is to describe the network architecture and elements requirements in the PPPoE scenario to support IPv6-only or dual-stack for Internet access service, has explicitly expressed in its Section 9.4, that the BNG must be able to support separate queues, input and output counters for IPv4 or IPv6 traffic. Note that BNG of BBF is a kind of broadband NAS of IETF. Meanwhile [BBF TR-187] suggested to use the RADIUS attributes (eg. Acct-Input-Octets (42), Acct-Output-Octets (43), Acct-Input-Packets (47), Acct-Output-Packets (48)) for the combination of IPv6 and IPv4 traffic. That means the new RADIUS attributes for reporting the separated IPv4 or IPv6 traffic statistics are required to be defined.

[I-D.hu-v6ops-radius-issues-ipv6-00] presented the same issue on 'protocol specific accounting' for the dual-stack traffic statistics, but it also limits to the PPP case.

[I-D.maglione-radext-ipv6-acct-extensions-01] and [I-D.yeh-radext-dual-stack-access-02] tried to defined a batch of attributes on the traffic statistics respectively for the IPv6-only access and dual-stack access in the traditional type space with the flat mode , while [I-D.winter-radext-fancyaccounting] indicated that the accounting attributes of Input-Octets, Output-Octets, Input-Packets and Output-Packets can be grouped in the new basic data type of TLV-nesting defined in Section 2.3 of [I-D.ietf-radext-radius-extensions] for the extended type space. According to [RFC6158] and Section 6.3 of [I-D.ietf-radext-radius-extensions], Nesting-TLV is the only recommendation for the new attribute design which intends to employ multiple fields in the complex data type now. Based on the judge of the quickly-exhausted traditional type space, the Radext Working Group tends to adopt the new data type of nesting-TLV defined for the extended type space to report the traffic statistics for the accounting extension.

This document specifies new attributes of the traffic statistics with different types for the extension of RADIUS accounting to support the

differentiated accounting policies and traffic recording on the AAA server. The reporting traffic types include the combination of stack types and the optional DSCP types.

Note: This document tries to narrow the scope of the solution space just to meet the requirements explicitly expressed by the industry without defining new RADIUS messages or introducing a new namespace for the additional interoperability.

2. Terminology and Conventions

Definitions for terms and acronyms not specifically defined in this document are defined in [RFC2865], [RFC2866], [RFC2869], [RFC3575], [RFC6158], and [I-D.ietf-radext-radius-extensions].

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in BCP 14, [RFC2119].

3. Deployment Scenarios

Figure 1 shows the typical use case of the traffic statistics reporting for the dual-stack users.

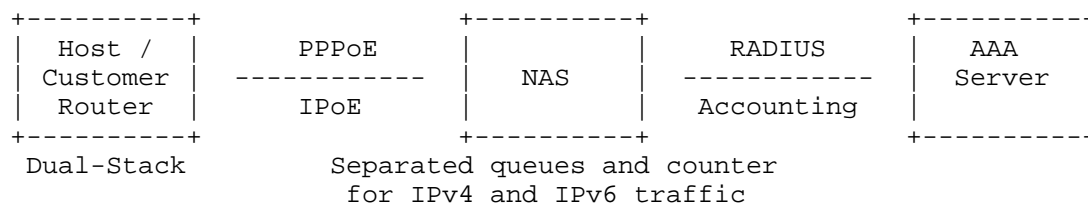


Figure 1: Traffic Statistics of Dual-Stack Users for RADIUS Accounting

Note that traffic statistics reporting is also needed in the IPv6 transition cases, such as DS-Lite, 6rd or MAP, where AFTR (Address Family Transition Router) or BR (Border Router) may act as the broadband NAS.

4. Acct-Traffic-Statistics attribute

Acct-traffic-statistics attribute is designed according to the guidelines described in [RFC6158] and Section 6 of [I-D.ietf-radext-radius-extensions]. It adopts the data structure of

the TLV nesting, has 1 container attribute, Acct-Traffic-Statistics, and 6 contained sub-attributes, Stack-Type, Input-Octets, Output-Octets, Input-Packets, Output-Packets, DSCP-Type (Differentiated Services CodePoint), and possible support the extensible types of traffic statistics in the future. The sub-attribute of Stack-Type MUST be included, one or more sub-attribute of Input-Octets, Output-Octets, Input-Packets or Output-Packets sub-attributes MUST be included and sub-attribute of DSCP-Type MAY be included in the container attribute, Acct-Traffic-Statistics. Because each of the sub-attributes has its own type code, the appearance of the contained sub-attributes is not necessary in a fixed order.

4.1. Container attribute of Acct-Traffic-Statistics

Description

The container attribute of Acct-Traffic-Statistics, which includes sub-attributes of Stack-Type, or DSCP-Type, and Input-Octets, Output-Octets, Input-Packets or Output-Packets, reports how many octets or packets of the specified traffic type, from the user or sent to the user, from the starting of the associated service provided. The sub-attributes of Stack-Type or DSCP-Type indicates the specified traffic type. One or more Acct-traffic-statistics attribute can be presented in Accounting-Request(4) message while the Acct-Status-Type(40) is set to Interim-Update or Stop.

The format of Acct-Traffic-Statistics attribute format is shown as 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									
										Value (cont.)...																													

Type

241, which indicates the extended type space.

Length

The length of the whole attribute in octet.

Extended-Type

TBA for Acct-Traffic-Statistics (See Section 8)

Value

The Value of the container attribute are the sub-attributes defined in Section 4.2 in TLV nesting mode. At least the sub-attribute of Stack-Type, and one of Input-Octets, Output-Octets, Input-Packets or Output-Packets sub-attributes MUST be included.

4.2. Contained attribute of Acct-Traffic-Statistics

4.2.1. Acct-Traffic-Statistics.Stack-Type

Description

Acct-Traffic-Statistics.Stack-Type sub-attribute indicates the type of the separated and combined traffic for IPv4 and IPv6.

The format of Acct-Traffic-Statistics.Stack-Type sub-attribute is shown as below. The fields are transmitted from left to right.

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

Type

TBA.1 for Acct-Traffic-Statistics.Stack-Type (See Section 8)

Length

6

Value

Enumerated Data Type in 4-Octet unsigned integer defined in [RFC6158]. The beginning 3 Octets are reserved for future use, and are set to 0x00 now. The decimal value of the last octet is defined as follows:

0 Combined traffic of IPv4 and IPv6

1 IPv4-only traffic

2 IPv6-only traffic

4.2.2. Acct-Traffic-Statistics.Input-Octets

Description

Acct-Traffic-Statistics.Input-Octets sub-attribute indicates how many octets in IP layer received from the user (or subscriber device) from the starting of the service authorized.

The format of Acct-Traffic-Statistics.Input-Octets sub-attribute is shown as below. The fields are transmitted from left to right.

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

Type

TBA.2 for Acct-Traffic-Statistics.Input-Octets (See Section 8)

Length

10

Value

Integer64 data type in 8-Octet unsigned integer defined in [I-D.ietf-radext-radius-extensions].

4.2.3. Acct-Traffic-Statistics.Output-Octets

Description

Acct-Traffic-Statistics.Output-Octets sub-attribute indicates how many octets in IP layer forwarded to the user (or subscriber device) from the starting of the service authorized.

The format of Acct-Traffic-Statistics.Output-Octets sub-attribute is shown as below. The fields are transmitted from left to right.

```

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

```

Type

TBA.3 for Acct-Traffic-Statistics.Output-Octets (See Section 8)

Length

10

Value

Integer64 data type in 8-Octet unsigned integer defined in [I-D.ietf-radext-radius-extensions].

4.2.4. Acct-Traffic-Statistics.Input-Packets

Description

Acct-Traffic-Statistics.Input-Packets sub-attribute indicates how many packets in IP layer received from the user (or subscriber device) from the starting of the service authorized.

The format of Acct-Traffic-Statistics.Input-Packets sub-attribute is shown as below. The fields are transmitted from left to right.

```

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

```

Type

TBA.4 for Acct-Traffic-Statistics.Input-Packets (See Section 8)

Length

10

Value

Integer64 data type in 8-Octet unsigned integer defined in [I-D.ietf-radext-radius-extensions].

4.2.5. Acct-Traffic-Statistics.Output-Packets

Description

Acct-Traffic-Statistics.Input-Packets sub-attribute indicates how many packets in IP layer forwarded to the user (or subscriber device) from the starting of the service authorized.

The format of Acct-Traffic-Statistics.Output-Packets sub-attribute is shown as below. The fields are transmitted from left to right.

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

Type

TBA.5 for Acct-Traffic-Statistics.Output-Packets (See Section 8)

Length

10

Value

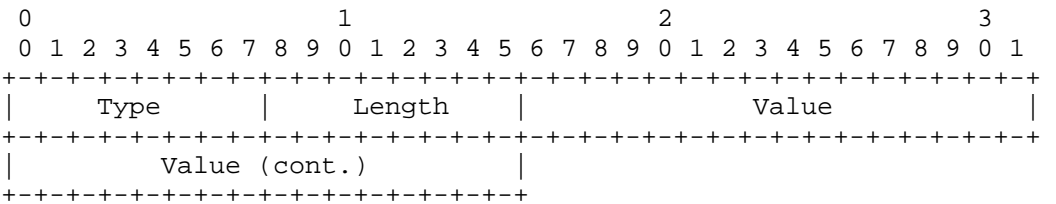
Integer64 data type in 8-Octet unsigned integer defined in [I-D.ietf-radext-radius-extensions].

4.2.6. Acct-Traffic-Statistics.DSCP-Type

Description

Acct-Traffic-Statistics.DSCP-Type sub-attribute indicates the DSCP type of the separated or combined IPv4 and IPv6 traffic.

The format of Acct-Traffic-Statistics.DSCP-Type sub-attribute is shown as below. The fields are transmitted from left to right.



Type

TBA.6 for Acct-Traffic-Statistics.DSCP-Type (See Section 8)

Length

6

Value

Enumerated Data Type in 4-Octet unsigned integer defined in [RFC6158]. The beginning 3 Octets are reserved for future use, and are set to 0x00 now. The first 2 bits of the last octet are reserved for future use, and are set to 00 now. The last 6 bits of the last octet is used to contain the DSCP value as per [RFC2474].

5. Table of Attribute

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

Req-uest	Acc-ept	Rej-ect	Chall-enge	Accounting # Request	Attribute
0	0	0	0	0+ TBA	Acct-Traffic-Statistics

The meaning of the above table entries is as follows:

- 0 This attribute MUST NOT be present.
- 0+ Zero or more instances of this attribute MAY be present.
- 0-1 Zero or one instance of this attribute MAY be present.
- 1 Exactly one instance of this attribute MUST be present.
- 1+ One or more of these attributes MUST be present.

6. Diameter Considerations

Given that the Attributes defined in this document are allocated from the RADIUS extended type space (see Section 8), no special handling is required by Diameter entities.

7. Security Considerations

Security issues related RADIUS are described in Section 8 of [RFC2865] and Section 5 of [RFC3162].

8. IANA Considerations

The authors of this document request to assign new Radius type codes for Acct-Traffic-Statistics and its following sub-attributes.

Acct-Traffic-Statistics.Stack-Type
Acct-Traffic-Statistics.Input-Octets
Acct-Traffic-Statistics.Output-Octets
Acct-Traffic-Statistics.Input-Packets
Acct-Traffic-Statistics.Output-Packets
Acct-Traffic-Statistics.DSCP-Type

These type codes should be allocated from the RADIUS extended type space based on Section 10 of [I-D.ietf-radext-radius-extensions] and "IETF Review" policy [RFC5226].

9. Acknowledgements

Thanks to Roberta Maglione, Jie Hu for their efforts in the history to bring this problem to IETF, to Stefan Winter, Alan DeKok, Peter Deacon for their valuable comments in the mailing list of Radext.

10. References

10.1. Normative References

- [I-D.ietf-radext-radius-extensions]
DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", draft-ietf-radext-radius-extensions-06 (work in progress), June 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.
- [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.
- [RFC2869] Rigney, C., Willats, W., and P. Calhoun, "RADIUS Extensions", RFC 2869, June 2000.
- [RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.
- [RFC4818] Salowey, J. and R. Droms, "RADIUS Delegated-IPv6-Prefix Attribute", RFC 4818, April 2007.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6158] DeKok, A. and G. Weber, "RADIUS Design Guidelines", BCP 158, RFC 6158, March 2011.

10.2. Informative References

- [BBF TR-187]
Broadband Forum, "IPv6 for PPP Broadband Access, Issue 1", May 2010.

- [I-D.hu-v6ops-radius-issues-ipv6-00]
Hu, J., Yan, L., Wang, Q., and J. Qin, "RADIUS issues in IPv6 deployments", February 2011.
- [I-D.ietf-radext-ipv6-access]
Dec, W., Sarikaya, B., Zorn, G., Miles, D., and B. Lourdelet, "RADIUS attributes for IPv6 Access Networks", draft-ietf-radext-ipv6-access-11 (work in progress), August 2012.
- [I-D.maglione-radext-ipv6-acct-extensions-01]
Maglione, R., Krishnan, S., Kavanagh, A., Varga, B., and J. Kaippallimalil, "RADIUS Accounting Extensions for IPv6", January 2011.
- [I-D.winter-radext-fancyaccounting]
Winter, S., "RADIUS Accounting for traffic classes", draft-winter-radext-fancyaccounting-02 (work in progress), July 2012.
- [I-D.yeh-radext-dual-stack-access-02]
Yeh, L. and T. Tsou, "RADIUS Attributes for Dual Stack Access", March 2011.

Authors' Addresses

Leaf Y. Yeh
Huawei Technologies
Shenzhen
P. R. China

Email: leaf.y.yeh@huawei.com

Mohamed Boucadair
France Telecom
Rennes,
France

Email: mohamed.boucadair@orange.com

