

Network Working Group
Internet-Draft
Updates: 6613, 6614 (if approved)
Intended status: Experimental
Expires: August 17, 2014

S. Hartman
Painless Security
February 13, 2014

Larger Packets for RADIUS over TCP
draft-hartman-radext-bigger-packets-01.txt

Abstract

The RADIUS over TLS experiment described in RFC 6614 has opened RADIUS to new use cases where the 4096-octet maximum RADIUS packet proves problematic. This specification extends the RADIUS over TCP experiment to permit larger RADIUS packets. This specification compliments other ongoing work to permit fragmentation of RADIUS authorization information.

Status of this Memo

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

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

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

This Internet-Draft will expire on August 17, 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	3
1.1. Requirements notation	3
2. Changes to Packet Processing	4
2.1. Status-Server Considerations	4
3. Forward and backward Compatibility	5
3.1. Rationale	6
3.2. Discovery	6
4. Too Big Response	8
5. Response Length Attribute	9
6. IANA Considerations	10
7. Security Considerations	11
8. References	12
8.1. Normative References	12
8.2. References	12
Author's Address	13

1. Introduction

The Remote Access Dial-In User Server (RADIUS) over TLS [RFC6614] experiment provides strong confidentiality and integrity for RADIUS [RFC2865]. This enhanced security has opened new opportunities for using RADIUS to convey additional authorization information. As an example, [I-D.ietf-abfab-aaa-saml] describes a mechanism for using RADIUS to carry Security Assertion Markup Language (SAML) messages in RADIUS. Many attributes carried in these SAML messages will require confidentiality or integrity such as that provided by TLS.

These new use cases involve carrying additional information in RADIUS packets. The maximum packet length of 4096 octets is proving insufficient for some SAML messages and for other structures that may be carried in RADIUS.

One approach is to fragment a RADIUS message across multiple packets at the RADIUS layer. RADIUS Fragmentation [I-D.ietf-radext-radius-fragmentation] provides a mechanism to split authorization information across multiple RADIUS messages. That mechanism is necessary in order to split authorization information across existing unmodified proxies.

However, there are some significant disadvantages to RADIUS fragmentation. First, RADIUS is a lock-step protocol, and only one fragment can be in transit at a time as part of a given request. Also, there is no current mechanism to discover the path Maximum Transmission Unit (MTU) across the entire path that the fragment will travel. As a result, fragmentation is likely both at the RADIUS layer and at the transport layer. When TCP is used, much better transport characteristics can be achieved by fragmentation only at the TCP layer.

1.1. Requirements notation

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. Changes to Packet Processing

The maximum length of a RADIUS message is increased from 4096 to 65535. A RADIUS Server implementing this specification MUST be able to receive a packet of maximum length. Servers MAY have a maximum size over which they choose to return an error as discussed in Section 4 rather than processing a received packet; this size MUST be at least 4096 octets.

Clients implementing this specification MUST be able to receive a packet of maximum length; that is clients MUST NOT close a TCP connection simply because a large packet is sent over it. Clients MAY include the Response-Length attribute defined in Section 5 to indicate the maximum size of a packet that they can successfully process. Clients MAY silently discard a packet greater than some configured size; this size MUST be at least 4096 octets. Clients MUST NOT retransmit an unmodified request whose response is larger than the client can process as subsequent responses will likely continue to be too large.

Proxies SHOULD be able to process and forward packets of maximum length. Proxies MUST include the Response-Length attribute when forwarding a request received over a transport with a 4096-octet maximum length over a transport with a higher maximum length.

2.1. Status-Server Considerations

This section extends processing of Status-Server messages as described in section 4.1 and 4.2 of [RFC5997].

Clients implementing this specification SHOULD include the Response-Length attribute in Status-Request messages. Servers are already required to ignore unknown attributes received in this message. by including the attribute, the client indicates how large of a response it can process to its Status-Server request. It is very unlikely that a response to Status-Server is greater than 4096 octets. However the client also indicates support for this specification which triggers server behavior below.

If a server implementing this specification receives a Response-Length attribute in a Status-Server request, it MUST include a Response-Length attribute indicating the maximum size request it can process in its response to the Status-Server request.

3. Forward and backward Compatibility

An implementation of [RFC6613] will silently discard any packet larger than 4096 octets and will close the TCP connection. This section provides guidelines for interoperability with these implementations. These guidelines are stated at the SHOULD level. In some environments support for large packets will be important enough that roaming or other agreements will mandate their support. In these environments, all implementations might be required to support this specification removing the need for interoperability with RFC 6613. It is likely that these guidelines will be relaxed to the MAY level and support for this specification made a requirement if RADIUS over TLS and TCP are moved to the standards track in the future.

Clients SHOULD provide configuration for the maximum size of a request sent to each server. Servers SHOULD provide configuration for the maximum size of a response sent to each client. If dynamic discovery mechanisms are supported, configuration SHOULD be provided for the maximum size of clients and servers in each dynamic discovery category.

If a client sends a request larger than 4096 octets and the TCP connection is closed without a response, the client SHOULD treat the request as if a request too big error (Section 4) specifying a maximum size of 4096 is received. Clients or proxies sending multiple requests over a single TCP connection without waiting for responses SHOULD implement capability discovery as discussed in Section 3.2.

By default, a server SHOULD not generate a response larger than 4096 octets. The Response-Length attribute MAY be included in a request to indicate that larger responses are desirable. Other attributes or configuration MAY be used as an indicator that large responses are likely to be acceptable.

A proxy that implements both this specification and RADIUS Fragmentation [I-D.ietf-radext-radius-fragmentation] SHOULD use RADIUS fragmentation when the following conditions are met:

1. A packet is being forwarded towards an endpoint whose configuration does not support a packet that large.
2. RADIUS Fragmentation can be used for the packet in question.

3.1. Rationale

The interoperability challenge appears at first significant. This specification proposes to introduce behavior where new implementations will fail to function with existing implementations.

However, these capabilities are introduced to support new use cases. If an implementation has 10000 octets of attributes to send, it cannot in general trim down the response to something that can be sent. Under this specification a large packet would be generated that will be silently discarded by an existing implementation. Without this specification, no packet is generated because the required attributes cannot be sent.

The biggest risk to interoperability would be if requests and responses are expanded to include additional information that is not strictly necessary. So, avoiding creating situations where large packets are sent to existing implementations is mostly an operational matter. Interoperability is most impacted when the size of packets in existing use cases is significantly increased and least impacted when large packets are used for new use cases where the deployment is likely to require updated RADIUS implementations.

There is a special challenge for proxies or clients with high request volume. When an RFC 6113 implementation receives a packet that is too large, it closes the connection and does not respond to any requests in process. Such a client would lose requests and might find distinguishing request-too-big situations from other failures difficult. In these cases, the discovery mechanism described in Section 3.2 can be used.

Also, RFC 6613 is an experiment. Part of running that experiment is to evaluate whether additional changes are required to RADIUS. A lower bar for interoperability should apply to changes to experimental protocols than standard protocols.

This specification provides good facilities to enable implementations to understand packet size when proxying to/from standards-track UDP RADIUS.

3.2. Discovery

As discussed in Section 2.1, a client MAY send a Status-Server message to discover whether an authentication or accounting server supports this specification. The client includes a Response-Length attribute; this signals the server to include a Response-Length attribute indicating the maximum packet size the server can process. In this one instance, Response-Length indicate the size of a request

that can be processed rather than a response.

4. Too Big Response

Define a new RADIUS code indicating that a server has received a request that is larger than can be processed. Include mandatory attribute indicating the maximum size that is permitted.

Clients will not typically be able to adjust and resend requests when this error is received. In some cases the client can fall back to RADIUS Fragmentation. In other cases this code will provide for better client error reporting and will avoid retransmitting requests guaranteed to fail.

5. Response Length Attribute

The following RADIUS attribute type values [RFC3575] are assigned. The assignment rules in section 10.3 of [RFC6929] are used.

Name	Attribute	Description
Response-Length	TBD	2-octet unsigned integer maximum response length

The Response-Length attribute MAY be included in any RADIUS request. In this context it indicates the maximum length of a response the client is prepared to receive. Values are between 4096 and 65535. The attribute MAY also be included in a response to a Status-Server message. In this case the attribute indicate the maximum size RADIUS request that is permitted.

6. IANA Considerations

Once this document is more complete it will define a new RADIUS code and attribute. Figure out if we have IANA policy lossage defining a code in an experimental document.

IANA is requested to assign the RADIUS type defined in section Section 5

7. Security Considerations

This specification updates RFC 6613 and will be used with [RFC6614]. When used over plain TCP, this specification creates new opportunities for an on-path attacker to impact availability. these attacks can be entirely mitigated by using TLS.

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.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.
- [RFC5997] DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", RFC 5997, August 2010.
- [RFC6613] DeKok, A., "RADIUS over TCP", RFC 6613, May 2012.
- [RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.
- [RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, April 2013.

8.2. References

- [I-D.ietf-abfab-aaa-saml]
Howlett, J. and S. Hartman, "A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for SAML", draft-ietf-abfab-aaa-saml-08 (work in progress), November 2013.
- [I-D.ietf-radext-radius-fragmentation]
Perez-Mendez, A., Lopez, R., Pereniguez-Garcia, F., Lopez-Millan, G., Lopez, D., and A. DeKok, "Support of fragmentation of RADIUS packets", draft-ietf-radext-radius-fragmentation-02 (work in progress), November 2013.

Author's Address

Sam Hartman
Painless Security

Email: hartmans-ietf@mit.edu

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.

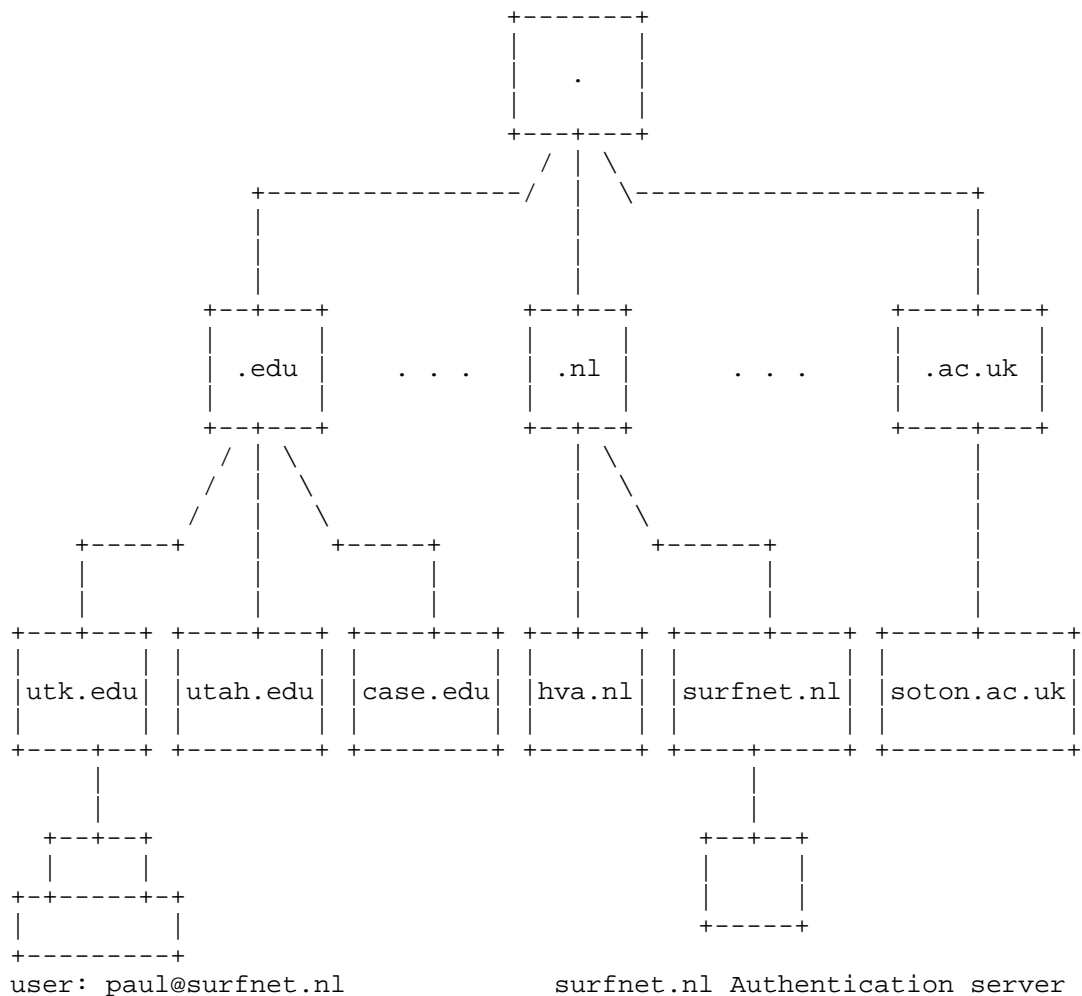


Figure 1: RADIUS hierarchy based on Top-Level Domain partitioning

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:

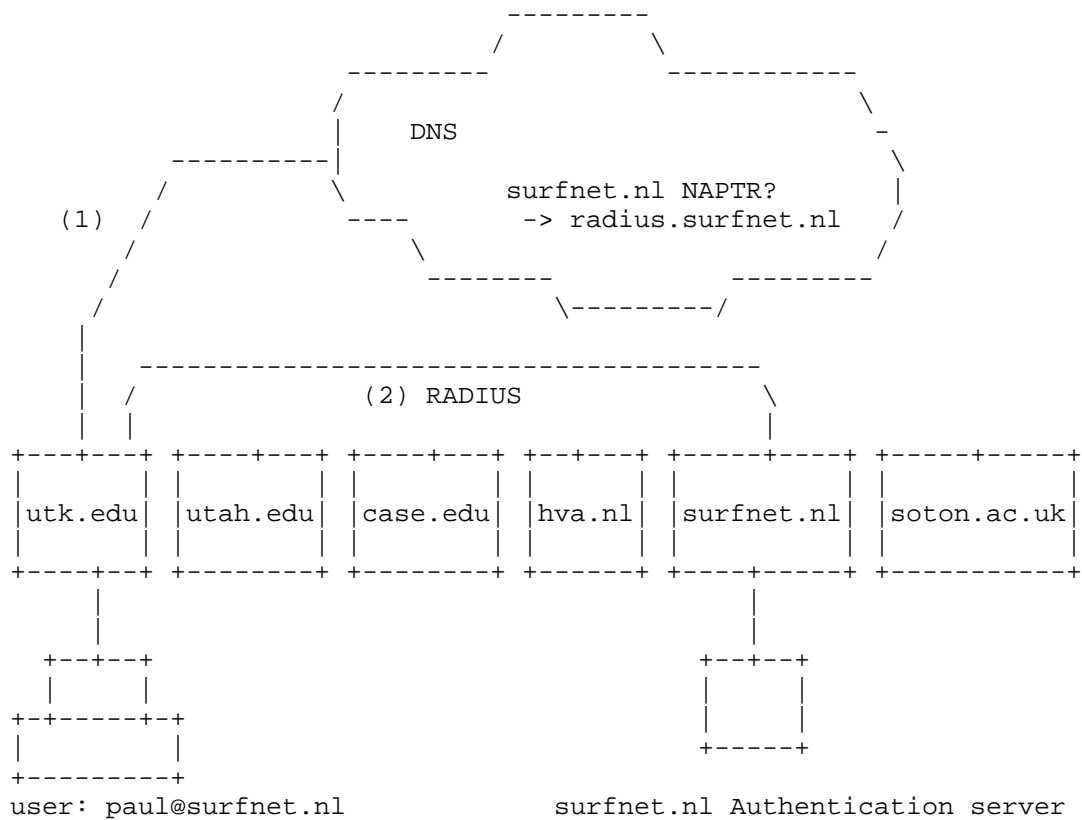


Figure 2: RADIUS hierarchy based on Top-Level Domain partitioning

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" "" roamserv.company.example.
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Examples:

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

Figure 6: Examples for NAI realm vs. certificate matching

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

3. DNS-based NAPTR/SRV Peer Discovery

3.1. Applicability

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

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

3.2. Configuration Variables

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

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

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

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

3.3. Terms

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

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

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

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

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

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

3.4. Realm to RADIUS server resolution algorithm

3.4.1. Input

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

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

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

- Usage of separators other than @.

- Encoding of User-Name in local encodings.

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

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

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

3.4.2. Output

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

3.4.3. Algorithm

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

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

11. $O' = (\text{set of } \{\text{hostname; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this hostname) } \} \text{ for all terminal lookup results}).$
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 wide-spread use in other protocols, this specification refrains from requesting a new registry "RADIUS/TLS SRV Protocol Registry" and continues to make use of these tags implicitly.

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

IANA is requested to assign the following identifiers:

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

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

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

8. References

8.1. Normative References

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

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

[I-D.ietf-radext-nai]

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

8.2. Informative References

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

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

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

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

[I-D.wierenga-ietf-eduroam]

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

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


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

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS

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

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

-- Service Name Object Identifier

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

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

-- Service Name

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

ub-naiRealm-length INTEGER ::= 255

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

END
```

Authors' Addresses

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

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

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

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

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

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

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

Abstract

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

Status of This Memo

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

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

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

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

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

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

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

1.1. Terminology

This document uses the following terms:

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

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

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

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

2.2. EAP-Key-Name

Description

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

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

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

Code

102 [RFC4072]

Length

>=3

String

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

2.3. EAP-Peer-Id

Description

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

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

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

```

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

```

Code

TBD2

Length

>=3

String

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

2.4. EAP-Server-Id

Description

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

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

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

```

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

```

Code

TBD3

Length

>=3

String

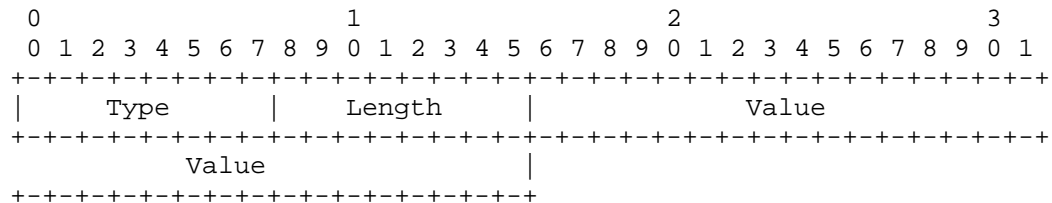
The String field is one or more octets, containing a EAP Server-Id exported by the EAP method. For details, see [RFC5247] Appendix A. A robust implementation SHOULD support the field as undistinguished octets.

2.5. Mobility-Domain-Id

Description

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

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



Code

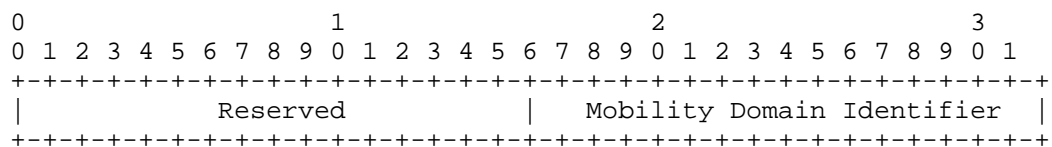
TBD4

Length

6

Value

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



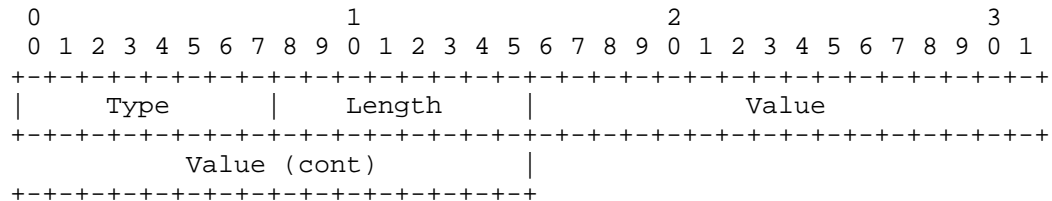
2.6. Preauth-Timeout

Description

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

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

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



Code

TBD5

Length

6

Value

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

2.7. Network-Id-Name

Description

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

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

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

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

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

```

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

```

Code

TBD6

Length

>=3

String

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

2.8. EAPoL-Announcement

Description

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

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

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

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

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

```

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

```

Code

TBD7

Length

>=3

String

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

2.9. WLAN-HESSID

Description

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

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

```

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

```

Code

TBD8

Length

19

String

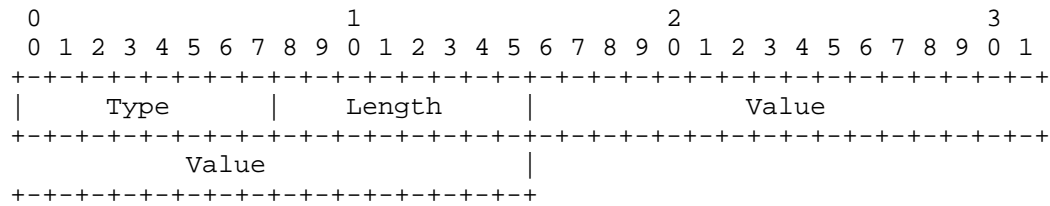
The String field is encoded in upper-case ASCII characters with the octet values separated by dash characters, as described in RFC 3580 [RFC3580]. Example: "00-10-A4-23-19-C0".

2.10. WLAN-Venue-Info

Description

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

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



Code

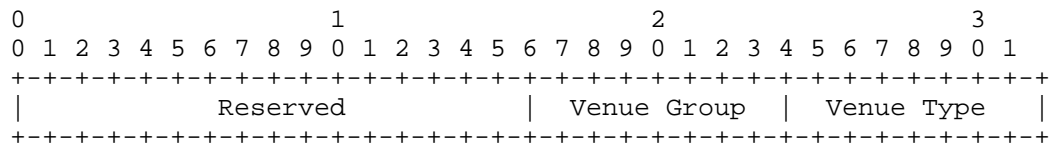
TBD9

Length

6

Value

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



Venue Group

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

Venue Type

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

2.11. WLAN-Venue-Language

Description

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

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

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

```

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

```

Code

TBD10

Length

4-5

String

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

2.12. WLAN-Venue-Name

Description

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

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

```

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

```

Code

TBD11

Length

>=3

String

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

2.13. WLAN-Reason-Code

Description

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

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

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

```

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

```

Code

TBD12

Length

6

Value

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

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

2.14. WLAN-Pairwise-Cipher

Description

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

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

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

Code

TBD13

Length

6

Value

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

```

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

```

2.15. WLAN-Group-Cipher

Description

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

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

```

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

```

Code

TBD14

Length

6

Value

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

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

```

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

```

2.16. WLAN-AKM-Suite

Description

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

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

```

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

```

Code

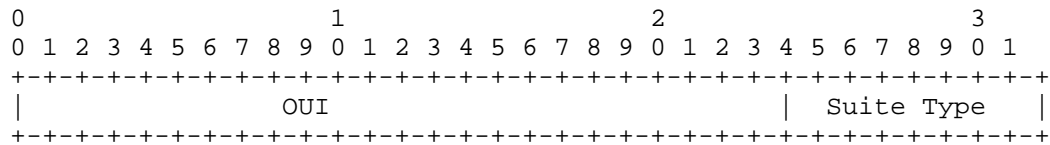
TBD15

Length

6

Value

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

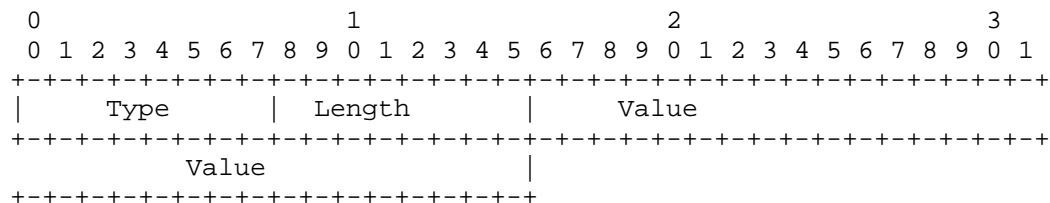


2.18. WLAN-RF-Band

Description

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

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



Code

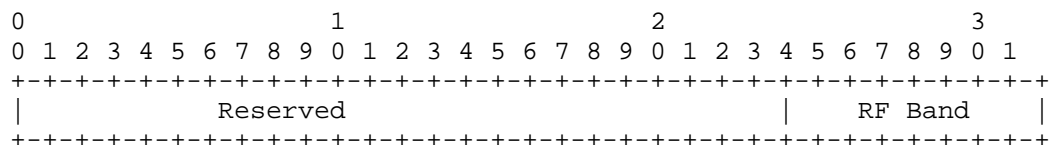
TBD17

Length

6

Value

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



3. Table of attributes

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

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

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

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

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

4. IANA Considerations

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

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

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

5. Security Considerations

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

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

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

6. References

6.1. Normative references

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

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

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

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

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

[ISO-639] ISO, "Codes for the Representation of Names of Languages".

[ISO-14962-1997]

ISO, "Space data and information transfer systems - ASCII encoded English", 1997.

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

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

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

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

6.2. Informative references

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

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

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

[RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G. and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, September 2003.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC5176] Chiba, M., Dommety, G., Eklund, M., Mitton, D. and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, January 2008.

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

November 2011.

Acknowledgments

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

Authors' Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: bernard_aboba@hotmail.com

Jouni Malinen
EMail: j@w1.fi

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

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

Joseph Salowey
Cisco Systems
EMail: jsalowey@cisco.com

Mark Jones
Azuca Systems
EMail: mark@azu.ca

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

DeKok, Alan
FreeRADIUS

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

Abstract

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

Status of this Memo

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

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

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

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

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

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

Copyright Notice

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

document authors. All rights reserved.

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

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

Table of Contents

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

1. Introduction

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

By "inter-domain authentication", this document refers to situations where a user has authentication credentials at one "home" domain, but is able to present them at a second "visited" domain to access certain services at the visited domain. The two domains generally have a pre-existing relationship, so that the credentials can be passed from the visited domain to the home domain for verification. The home domain typically responds with a permit / deny response, which may also include authorization parameters which the visited domain is expected to enforce on the user.

That is, the "roaming" scenario involves a user visiting, or "roaming" to a non-home domain, and requesting the use of services at that visited domain.

Interested parties have included the following:

- * Regional Internet Service Providers (ISPs) operating within a particular state or province, looking to combine their efforts with those of other regional providers to offer dialup service over a wider area.
- * Telecommunications companies who wish to combine their operations with those of one or more companies in another areas or nations, in order to offer more comprehensive network access service in areas where there is no native service. e.g. In another country.
- * Wireless LAN hotspots providing service to one or more ISPs.
- * Businesses desiring to offer their employees a comprehensive package of dialup services on a global basis. Those services may include Internet access as well as secure access to corporate intranets via a VPN, enabled by tunneling protocols such as the Point-to-Point Tunneling Protocol (PPTP) [RFC2637], the Layer 2 Forwarding (L2F) protocol [RFC2341], the Layer 2 Tunneling Protocol (L2TP) [RFC2661], and the IPsec tunnel mode [RFC4301].
- * Other protocols which are interested in leveraging the users credentials in order to take advantage of an existing authentication framework.

In order to enhance the interoperability of these services, it is necessary to have a standardized method for identifying users. This document defines syntax for the Network Access Identifier (NAI). Examples of implementations that use the NAI, and descriptions of its semantics, can be found in [RFC2194].

When the NAI was defined for network access, it had the side effect of defining an identifier which could be used in non-AAA systems. Some non-AAA systems defined identifiers which were compatible with the NAI, and deployments used the NAI. This process simplified the management of credentials, by re-using the same credential in multiple situations. Protocols that re-use the same credential or the same identifier format can benefit from this management simplicity. The alternative is to have protocol-specific credentials or identifier formats, which increases cost to both the user and the administrator.

There are privacy implications to using one identifier across multiple protocols. See Section 2.7 and Section 4 for further discussion of this topic.

The goal of this document is to define the format of an identifier which can be used in many protocols. A protocol may transport an encoded version of the NAI (e.g. '.' as %2E). However, the definition of the NAI is protocol independent. The goal of this document is to encourage the wide-spread adoption of the NAI format. This adoption will decrease work required to leverage identification and authentication in other protocols. It will also decrease the complexity of non-AAA systems for end users and administrators.

This document only suggests that the NAI format be used, but does not require such use. Many protocols already define their own identifier formats. Some of these are incompatible with the NAI, while others allow the NAI in addition to non-NAI identifiers. The definition of the NAI in this document has no requirements on protocol specifications, implementations, or deployments.

However, this document suggests that using one standard identifier format is preferable to using multiple incompatible identifier formats. Where identifiers need to be used in new protocols and/or specifications, it is RECOMMENDED that the format of the NAI be used. That is, the interpretation of the identifier is context-specific, while the format of the identifier remains the same. These issues are discussed in more detail in Section 2.8, below.

The recommendation for a standard identifier format is not a recommendation that each user have one universal identifier. In contrast, this document allows for the use of multiple identifiers,

and recommends the use of anonymous identifiers where those identifiers are publicly visible.

This document is a revised version of [RFC4282], which originally defined internationalized NAIs. Differences and enhancements compared to that document are listed in Appendix A.

1.1. Terminology

This document frequently uses the following terms:

"Local" or "localized" text

Text which is either in non-UTF-8, or in non-normalized form. The character set, encoding, and locale are (in general) unknown to Authentication, Authorization, and Accounting (AAA) network protocols. The client which "knows" the locale may have a different concept of this text than other AAA entities, which do not know the same locale.

Network Access Identifier

The Network Access Identifier (NAI) is a common format for user identifiers submitted by a client during authentication. The purpose of the NAI is to allow a user to be associated with an account name, as well as to assist in the routing of the authentication request across multiple domains. Please note that the NAI may not necessarily be the same as the user's email address or the user identifier submitted in an application layer authentication.

Network Access Server

The Network Access Server (NAS) is the device that clients connect to in order to get access to the network. In PPTP terminology, this is referred to as the PPTP Access Concentrator (PAC), and in L2TP terminology, it is referred to as the L2TP Access Concentrator (LAC). In IEEE 802.11, it is referred to as an Access Point.

Roaming Capability

Roaming capability can be loosely defined as the ability to use any one of multiple Internet Service Providers (ISPs), while maintaining a formal, customer-vendor relationship with only one. Examples of cases where roaming capability might be required include ISP "confederations" and ISP-provided corporate network access support.

Normalization or Canonicalization

These terms are defined in [RFC6365] Section 4. Those definitions are incorporated here by reference.

Locale

This term is defined in [RFC6365] Section 8. Those definitions are incorporated here by reference.

Tunneling Service

A tunneling service is any network service enabled by tunneling protocols such as PPTP, L2F, L2TP, and IPsec tunnel mode. One example of a tunneling service is secure access to corporate intranets via a Virtual Private Network (VPN).

1.2. Requirements Language

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

1.3. Purpose

As described in [RFC2194], there are a number of providers offering network access services, and essentially all Internet Service Providers are involved in roaming consortia.

In order to be able to offer roaming capability, one of the requirements is to be able to identify the user's home authentication server. For use in roaming, this function is accomplished via the Network Access Identifier (NAI) submitted by the user to the NAS in the initial network authentication. It is also expected that NASes will use the NAI as part of the process of opening a new tunnel, in order to determine the tunnel endpoint.

This document suggests that other protocols can take advantage of the NAI format. Many protocols include authentication capabilities, including defining their own identifier formats. These identifiers can then end up being transported in AAA protocols, so that the originating protocols can leverage AAA for user authentication. There is therefore a need for a definition of a user identifier which can be used in multiple protocols.

While the NAI is defined herein, it should be noted that existing protocols and deployments do not always use it. AAA systems **MUST** therefore be able to handle user identifiers which are not in the NAI format. The process by which that is done is outside of the scope of this document.

Non-AAA systems can accept user identifiers in forms other than the NAI. This specification does not forbid that practice. It only codifies the format and interpretation of the NAI. This document cannot change existing protocols or practices. It can, however, suggest that using a consistent form for a user identifier is of a benefit to the community.

This document does not make any protocol-specific definitions for an identifier format, and it does not make changes to any existing protocol. Instead, it defines a protocol-independent form for the NAI. It is hoped that the NAI is a user identifier which can be used in multiple protocols.

Using a common identifier format simplifies protocols requiring authentication, as they no longer need to specify protocol-specific format for user identifiers. It increases security, as multiple identifier formats allow attackers to make contradictory claims without being detected (see Section 4.2 for further discussion of this topic). It simplifies deployments, as a user can have one identifier in multiple contexts, which allows them to be uniquely

identified, so long as that identifier is itself protected against unauthorized access.

In short, having a standard is better than having no standard at all.

1.4. Motivation

The changes from [RFC4282] are listed in detail in Appendix A. However, some additional discussion is appropriate to motivate those changes.

The motivation to revise [RFC4282] began with internationalization concerns raised in the context of [EDUROAM]. Section 2.1 of [RFC4282] defines ABNF for realms which limits the realm grammar to English letters, digits, and the hyphen "-" character. The intent appears to have been to encode, compare, and transport realms with the Punycode [RFC3492] encoding form as described in [RFC5891]. There are a number of problems with this approach:

- * The [RFC4282] ABNF is not aligned with internationalization of DNS.
- * The requirement in [RFC4282] Section 2.1 that realms are ASCII conflicts with the Extensible Authentication Protocol (EAP) defined in [RFC3748], and RADIUS, which are both 8-bit clean, and which both recommend the use of UTF-8 for identifiers.
- * [RFC4282] Section 2.4 required mappings that are language-specific, and which are nearly impossible for intermediate nodes to perform correctly without information about that language.
- * [RFC4282] Section 2.4 requires normalization of user names, which may conflict with local system or administrative requirements.
- * The recommendations in RFC4282] Section 2.4 for treatment of bidirectional characters have proven to be unworkable.
- * The prohibition against use of unassigned code points in RFC4282] Section 2.4 effectively prohibits support for new scripts.
- * No Authentication, Authorization, and Accounting (AAA) client, proxy, or server has implemented any of the requirements in [RFC4282] Section 2.4, among other sections.

With international roaming growing in popularity, it is important for these issues to be corrected in order to provide robust and inter-

operable network services.

Furthermore, this document was motivated by a desire to codify existing practice related to the use of the NAI format and to encourage widespread use of the format.

2. NAI Definition

2.1. UTF-8 Syntax and Normalization

UTF-8 characters can be defined in terms of octets using the following ABNF [RFC5234], taken from [RFC3629]:

UTF8-xtra-char = UTF8-2 / UTF8-3 / UTF8-4

UTF8-2 = %xC2-DF UTF8-tail

UTF8-3 = %xE0 %xA0-BF UTF8-tail /
 %xE1-EC 2(UTF8-tail) /
 %xED %x80-9F UTF8-tail /
 %xEE-EF 2(UTF8-tail)

UTF8-4 = %xF0 %x90-BF 2(UTF8-tail) /
 %xF1-F3 3(UTF8-tail) /
 %xF4 %x80-8F 2(UTF8-tail)

UTF8-tail = %x80-BF

These are normatively defined in [RFC3629], but are repeated in this document for reasons of convenience.

See [RFC5198] and section 2.6 of this specification for a discussion of normalization. Strings which are not in Normal Form Composed (NFC) are not valid NAIs and SHOULD NOT be treated as such.

Implementations which expect to receive a NAI, but which instead receive non-normalised (but otherwise valid) UTF-8 strings instead SHOULD attempt to create a local version of the NAI, which is normalized from the input identifier. This local version can then be used for local processing. This local version of the identifier MUST NOT be used outside of the local context.

Where protocols carry identifiers which are expected to be transported over an AAA protocol, it is RECOMMENDED that the identifiers be in NAI format. Where the identifiers are not in the NAI format, it is up to the AAA systems to discover this, and to process them. This document does not suggest how that is done. However, existing practice indicates that it is possible.

As internationalized domain names become more widely used, existing practices are likely to become inadequate. This document therefore defines the NAI, which is a user identifier format that can correctly deal with internationalized identifiers.

2.2. Formal Syntax

The grammar for the NAI is given below, described in Augmented Backus-Naur Form (ABNF) as documented in [RFC5234].

```
nai           = utf8-username
nai           =/ "@" utf8-realm
nai           =/ utf8-username "@" utf8-realm

utf8-username = dot-string

dot-string    = string *("." string)
string        = 1*utf8-atext

utf8-atext    = ALPHA / DIGIT /
               "!" / "#" /
               "$" / "%" /
               "&" / "'" /
               "*" / "+" /
               "-" / "/" /
               "=" / "?" /
               "^" / "_" /
               "`" / "{" /
               "|" / "}" /
               "~" /
               UTF8-xtra-char

utf8-realm    = 1*( label "." ) label

label         = utf8-rtext *(ldh-str)
ldh-str       = *( utf8-rtext / "-" ) utf8-rtext
utf8-rtext    = ALPHA / DIGIT / UTF8-xtra-char
```

2.3. NAI Length Considerations

Devices handling NAIs MUST support an NAI length of at least 72 octets. Devices SHOULD support an NAI length of 253 octets. However, the following implementation issues should be considered:

- * NAI octet length constraints may impose a more severe constraint on the number of UTF-8 characters.
- * NAIs are often transported in the User-Name attribute of the

Remote Authentication Dial-In User Service (RADIUS) protocol. Unfortunately, RFC 2865 [RFC2865], Section 5.1, states that "the ability to handle at least 63 octets is recommended." As a result, it may not be possible to transfer NAIs beyond 63 octets through all devices. In addition, since only a single User-Name attribute may be included in a RADIUS message and the maximum attribute length is 253 octets, RADIUS is unable to support NAI lengths beyond 253 octets.

- * NAIs can also be transported in the User-Name attribute of Diameter [RFC6733], which supports content lengths up to $2^{24} - 9$ octets. As a result, NAIs processed only by Diameter nodes can be very long. However, an NAI transported over Diameter may eventually be translated to RADIUS, in which case the above limitations will apply.

- * NAIs may be transported in other protocols. Each protocol can have its own limitations on maximum NAI length. The above criteria should permit the widest use, and widest possible inter-operability of the NAI.

2.4. Support for Username Privacy

Interpretation of the username part of the NAI depends on the realm in question. Therefore, the utf8-username portion SHOULD be treated as opaque data when processed by nodes that are not a part of the home domain for that realm.

That is, the only domain which is capable of interpreting the meaning of the utf8-username portion of the NAI is the home domain. Any third-party domains cannot form any conclusions about the utf8-username, and cannot decode it into sub-fields. For example, it may be used as "firstname.lastname", or it may be entirely digits, or it may be a random hex identifier. There is simply no way (and no reason) for any other domain to interpret the utf8-username field as having any meaning whatsoever.

In some situations, NAIs are used together with a separate authentication method that can transfer the username part in a more secure manner to increase privacy. In this case, NAIs MAY be provided in an abbreviated form by omitting the username part. Omitting the username part is RECOMMENDED over using a fixed username part, such as "anonymous", since including a fixed username part is ambiguous as to whether or not the NAI refers to a single user. However, current practice is to use the username "anonymous" instead of omitting the username part. This behavior is also permitted.

The most common use-case of omitting or obfuscating the username part

is with TLS-based EAP methods such as TTLS [RFC5281]. Those methods allow for an "outer" identifier, which is typically an anonymous "@realm". This outer identifier allows the authentication request to be routed from a visited domain to a home domain. At the same time, the username part is kept confidential from the visited network. The protocol provides for an "inner" authentication exchange, in which a full identifier is used to authenticate a user.

That scenario offers the best of both worlds. An anonymous NAI can be used to route authentication to the home domain, and the home domain has sufficient information to identify and authenticate users.

However, some protocols do not support authenticate methods which allow for "inner" and "outer" exchanges. Those protocols are limited to using an identifier which is publicly visible. It is therefore RECOMMENDED that such protocols use ephemeral identifiers. We recognize that this practice is not currently used, and will likely be difficult to implement.

Similarly to the anonymous user, there may be situations where portions of the realm are sensitive. For those situations, it is RECOMMENDED that the sensitive portion of the realm also be omitted. e.g. To use "@example.com" instead of "@sensitive.example.com", or "anonymous@sensitive.example.com". The home domain is authoritative for users in all subdomains, and can (if necessary) route the authentication request to the appropriate subsystem within the home domain.

For roaming purposes, it is typically necessary to locate the appropriate backend authentication server for the given NAI before the authentication conversation can proceed. As a result, authentication routing is impossible unless the realm portion is available, and in a well-known format.

2.5. International Character Sets

This specification allows both international usernames and realms. International usernames are based on the use of Unicode characters, encoded as UTF-8. Internationalization of the username portion of the NAI is based on the "Internationalized Email Headers" [RFC6532] extensions to the "local-part" portion of email addresses [RFC5322].

In order to ensure a canonical representation, characters of the realm portion in an NAI MUST match the ABNF in this specification as well as the requirements specified in [RFC5891]. In practice, these requirements consist of the following item:

- * Realms MUST be of the form that can be registered as a

Fully Qualified Domain Name (FQDN) within the DNS.

This list is significantly shorter and simpler than the list in Section 2.4 of [RFC4282]. The form suggested in [RFC4282] depended on intermediate nodes performing canonicalizations based on insufficient information, which meant that the form was not canonical.

Specifying the realm requirement as above means that the requirements depend on specifications that are referenced here, rather than copied here. This allows the realm definition to be updated when the referenced documents change, without requiring a revision of this specification.

One caveat on the above recommendation is the issues noted in [RFC6912]. That document notes that there are additional restrictions around DNS registration which forbid some code points from being valid in a DNS U-label. These restrictions cannot be expressed algorithmically.

For this specification, that caveat means the following. Realms not matching the above ABNF are not valid NAIs. However, some realms which do match the ABNF are still invalid NAIs. That is, matching the ABNF is a necessary, but not sufficient, requirement for an NAI.

In general, the above requirement means following the requirements specified in [RFC5891].

2.6. The Normalization Process

Conversion to Unicode as well as normalization SHOULD be performed by edge systems (e.g. laptops, desktops, smart phones, etc.) that take "local" text as input. These edge systems are best suited to determine the users intent, and can best convert from "local" text to a normalized form.

Other AAA systems such as proxies do not have access to locale and character set information that is available to edge systems. Therefore, they may not always be able to convert local input to Unicode.

That is, all processing of NAIs from "local" character sets and locales to UTF-8 SHOULD be performed by edge systems, prior to the NAIs entering the AAA system. Inside of an AAA system, NAIs are sent over the wire in their canonical form, and this canonical form is used for all NAI and/or realm comparisons.

Copying of localized text into fields that can subsequently be placed

into the RADIUS User-Name attribute is problematic. This practice can result in a AAA proxy encountering non-UTF8 characters within what it expects to be an NAI. An example of this requirement is [RFC3579] Section 2.1, which states:

the NAS MUST copy the contents of the Type-Data field of the EAP-Response/Identity received from the peer into the User-Name attribute

As a result, AAA proxies expect the contents of the EAP-Response/Identity sent by an EAP supplicant to consist of UTF-8 characters, not localized text. Using localized text in AAA username or identity fields means that realm routing becomes difficult or impossible.

In contrast to [RFC4282] Section 2.4, AAA systems are now expected to perform NAI comparisons, matching, and AAA routing based on the NAI as it is received. This specification provides a canonical representation, ensures that intermediate AAA systems such as proxies are not required to perform translations, and can be expected to work through AAA systems that are unaware of international character sets.

In an ideal world, the following requirements would be widely implemented:

- * Edge systems using "localized" text SHOULD normalize the NAI prior to it being used as an identifier in an authentication protocol.
- * AAA systems SHOULD NOT normalize the NAI, as they may not have sufficient information to perform the normalization.

There are issues with this approach, however.

2.6.1. Issues with the Normalization Process

The requirements in the preceding section are not implemented today. For example, most EAP implementations use a user identifier which is passed to them from some other local system. This identifier is treated as an opaque blob, and is placed as-is into the EAP Identity field. Any subsequent system which receives that identifier is assumed to be able to understand and process it.

This opaque blob unfortunately can contain localized text, which means that the AAA systems have to process that text.

These limitations have the following theoretical and practical implications.

- * edge systems used today generally do not normalize the NAI

- * Therefore AAA systems SHOULD attempt to normalize the NAI

The suggestion in the above sentence contradicts the suggestion in the previous section. This is the reality of imperfect protocols.

Where the user identifier can be normalized, or determined to be in normal form, the normal form MUST be used as the NAI. In all other circumstances, the user identifier MUST NOT be treated as an NAI. That data is still, however, a user identifier. AAA systems MUST NOT fail authentication simply because the user identifier is not an NAI.

That is, when the realm portion of the NAI is not recognized by an AAA server, it SHOULD try to normalize the NAI into NFC form. That normalized form can then be used to see if the realm matches a known realm. If no match is found, the original form of the NAI SHOULD be used in all subsequent processing.

The AAA server may also convert realms to punycode, and perform all realm comparisons on the resulting punycode strings. This conversion follows the recommendations above, but may have different operational effects and failure modes.

2.7. Use in Other Protocols

As noted earlier, the NAI format can be used in other, non-AAA protocols. It is RECOMMENDED that the definition given here be used unchanged. Using other definitions for user identifiers may hinder interoperability, along with the users ability to authenticate successfully. It is RECOMMENDED that protocols requiring the use of a user identifier use the NAI format.

This document cannot require other protocols to use the NAI format for user identifiers. Their needs are unknown, and at this time unknowable. This document suggests that interoperability and inter-domain authentication is useful, and should be encouraged.

Where a protocol is 8-bit clean, it can likely transport the NAI as-is, without further modification.

Where a protocol is not 8-bit clean, it cannot transport the NAI as-is. Instead, this document presumes that a protocol-specific transport layer takes care of encoding the NAI on input to the protocol, and decoding it when the NAI exits the protocol. The encoded or escaped version of the NAI is not a valid NAI, and MUST NOT be presented to the AAA system.

For example, HTTP carries user identifiers, but escapes the '.' character as "%2E" (among others). When HTTP is used to transport the NAI "fred@example.com", the data as transported will be in the form "fred@example%2Ecom". That data exists only within HTTP, and has no relevance to any AAA system.

Any comparison, validation, or use of the NAI MUST be done on its un-escaped (i.e. utf8-clean) form.

2.8. Using the NAI format for other identifiers

As discussed in Section 1, above, is RECOMMENDED that the NAI format be used as the standard format for user identifiers. This section discusses that use in more detail.

It is often useful to create new identifiers for use in specific contexts. These identifiers may have a number of different properties, most of which are unimportant to this document. The goal of this document is to create identifiers which are to be in a well-known format, and to have namespaces. The NAI format fits these requirements.

One example of such use is the "private user identity", which is an identifier defined by the 3rd-Generation Partnership Project (3GPP). That identifier is used to uniquely identify the user to the network. The identifier is used for authorization, authentication, accounting, administration, etc. The "private user identity" is globally unique, and is defined by the home network operator. The format of the identifier is explicitly the NAI, as stated by Section 13.3 of [3GPP]:

The private user identity shall take the form of an NAI, and shall have the form username@realm as specified in clause 2.1 of IETF RFC 4282

For 3GPP, the "username" portion is a unique identifier which is derived from device-specific information. The "realm" portion is composed of information about the home network, followed by the base string "3gppnetwork.org". e.g.
234150999999999@ims.mnc015.mcc234.3gppnetwork.org.

This format as defined by 3GPP ensures that the identifier is globally unique, as it is based off of the "3gppnetwork.org" domain. It ensures that the "realm" portion is specific to a particular home network (or organization), via the "ims.mnc015.mcc234" prefix to the realm. Finally, it ensures that the "username" portion follows a well-known format.

This document suggests that the NAI format be used for all new specifications and/or protocols where a user identifier is required. Where the username portions need to be created with subfields, a well-known and documented method as has been done with 3GPP is preferred to ad-hoc methods.

3. Routing inside of AAA Systems

Many AAA systems use the "utf8-realm" portion of the NAI to route requests within a AAA proxy network. The semantics of this operation involves a logical AAA routing table, where the "utf8-realm" portion acts as a key, and the values stored in the table are one or more "next hop" AAA servers.

Intermediate nodes **MUST** use the "utf8-realm" portion of the NAI without modification to perform this lookup. As noted earlier, intermediate nodes may not have access to the same locale information as the system which injected the NAI into the AAA routing systems. Therefore, almost all "case insensitive" comparisons can be wrong. Where the "utf8-realm" is entirely ASCII, current AAA systems sometimes perform case-insensitive matching on realms. This method **MAY** be continued, as it has been shown to work in practice.

Many existing non-AAA systems have user identifiers which are similar in format to the NAI, but which are not compliant with this specification. For example, they may use non-NFC form, or they may have multiple "@" characters in the user identifier. Intermediate nodes **SHOULD** normalize non-NFC identifiers to NFC, prior to looking up the "utf8-realm" in the logical routing table. Intermediate nodes **MUST NOT** modify the identifiers that they forward. The data as entered by the user is inviolate.

The "utf8-realm" provisioned in the logical AAA routing table **SHOULD** be provisioned to the proxy prior to it receiving any AAA traffic. The "utf8-realm" **SHOULD** be supplied by the "next hop" or "home" system that also supplies the routing information necessary for packets to reach the next hop.

This "next hop" information may be any of, or all of, the following information: IP address; port; RADIUS shared secret; TLS certificate; DNS host name; or instruction to use dynamic DNS discovery (i.e. look up a record in the "utf8-realm" domain). This list is not exhaustive, and may be extended by future specifications.

It is **RECOMMENDED** to use the entirety of the "utf8-realm" for the routing decisions. However, AAA systems **MAY** use a portion of the "utf8-realm" portion, so long as that portion is a valid "utf8-realm", and that portion is handled as above. For example,

routing "fred@example.com" to a "com" destination is forbidden, because "com" is not a valid "utf8-realm". However, routing "fred@sales.example.com" to the "example.com" destination is permissible.

Another reason to forbid the use of a single label (e.g. "fred@sales") is that many non-AAA systems treat a single label as being a local identifier within their realm. That is, a user logging in as "fred@sales" to a domain "example.com", would be treated as if the NAI was instead "fred@sales.example.com". Permitting the use of a single label would mean changing the interpretation and meaning of a single label, which cannot be done.

3.1. Compatibility with Email Usernames

As proposed in this document, the Network Access Identifier is of the form "user@realm". Please note that while the user portion of the NAI is based on the "Internet Message Format" [RFC5322] "local-part" portion of an email address as extended by "Internationalized Email Headers" [RFC6532], it has been modified for the purposes of Section 2.2. It does not permit quoted text along with "folding" or "non-folding" whitespace that is commonly used in email addresses. As such, the NAI is not necessarily equivalent to usernames used in e-mail.

However, it is a common practice to use email addresses as user identifiers in AAA systems. The ABNF in Section 2.2 is defined to be close to the "addr-spec" portion of [RFC5322] as extended by [RFC6532], while still being compatible with [RFC4282].

In contrast to [RFC4282] Section 2.5, this document states that the internationalization requirements for NAIs and email addresses are substantially similar. The NAI and email identifiers may be the same, and both need to be entered by the user and/or the operator supplying network access to that user. There is therefore good reason for the internationalization requirements to be similar.

3.2. Compatibility with DNS

The "utf8-realm" portion of the NAI is intended to be compatible with Internationalized Domain Names (IDNs) [RFC5890]. As defined above, the "utf8-realm" portion as transported within an 8-bit clean protocol such as RADIUS and EAP can contain any valid UTF8 character. There is therefore no reason for a NAS to convert the "utf8-realm" portion of an NAI into Punycode encoding form [RFC3492] prior to placing the NAI into a RADIUS User-Name attribute.

The NAI does not make a distinction between A-labels and U-labels, as

those are terms specific to DNS. It is instead an IDNA-valid label, as per the first item in Section 2.3.2.1 of [RFC5890]. As noted in that section, the term "IDNA-valid label" encompasses both of the terms A-label and U-label.

When the realm portion of the NAI is used as the basis for name resolution, it may be necessary to convert internationalized realm names to Punycode [RFC3492] encoding form as described in [RFC5891]. As noted in [RFC6055] Section 2, resolver Application Programming Interfaces (APIs) are not necessarily DNS-specific, so conversion to Punycode needs to be done carefully:

Applications which convert an IDN to A-label form before calling (for example) `getaddrinfo()` will result in name resolution failures if the Punycode name is directly used in such protocols. Having libraries or protocols to convert from A-labels to the encoding scheme defined by the protocol (e.g., UTF-8) would require changes to APIs and/or servers, which IDNA was intended to avoid.

As a result, applications SHOULD NOT assume that non-ASCII names are resolvable using the public DNS and blindly convert them to A-labels without knowledge of what protocol will be selected by the name resolution library.

3.3. Realm Construction

The home realm usually appears in the "utf8-realm" portion of the NAI, but in some cases a different realm can be used. This may be useful, for instance, when the home realm is reachable only via intermediate proxies.

Such usage may prevent interoperability unless the parties involved have a mutual agreement that the usage is allowed. In particular, NAIs MUST NOT use a different realm than the home realm unless the sender has explicit knowledge that (a) the specified other realm is available and (b) the other realm supports such usage. The sender may determine the fulfillment of these conditions through a database, dynamic discovery, or other means not specified here. Note that the first condition is affected by roaming, as the availability of the other realm may depend on the user's location or the desired application.

The use of the home realm MUST be the default unless otherwise configured.

3.3.1. Historical Practices

Some AAA systems have historically used NAI modifications with multiple "prefix" and "suffix" decorations to perform explicit routing through multiple proxies inside of a AAA network.

In RADIUS based environment, the use of decorated NAI is NOT RECOMMENDED for the following reasons:

- * Using explicit routing paths is fragile, and is unresponsive to changes in the network due to servers going up or down, or to changing business relationships.
- * There is no RADIUS routing protocol, meaning that routing paths have to be communicated "out of band" to all intermediate AAA nodes, and also to all edge systems (e.g. supplicants) expecting to obtain network access.
- * Using explicit routing paths requires thousands, if not millions of edge systems to be updated with new path information when a AAA routing path changes. This adds huge expense for updates that would be better done at only a few AAA systems in the network.
- * Manual updates to RADIUS paths are expensive, time-consuming, and prone to error.
- * Creating compatible formats for the NAI is difficult when locally-defined "prefixes" and "suffixes" conflict with similar practices elsewhere in the network. These conflicts mean that connecting two networks may be impossible in some cases, as there is no way for packets to be routed properly in a way that meets all requirements at all intermediate proxies.
- * Leveraging the DNS name system for realm names establishes a globally unique name space for realms.

In summary, network practices and capabilities have changed significantly since NAIs were first overloaded to define AAA routes through a network. While manually managed explicit path routing was once useful, the time has come for better methods to be used.

Notwithstanding the above recommendations, the above practice is widely used for Diameter routing [RFC5729]. The routes described there are managed automatically, for both credential provisioning and routing updates. Those routes also exist within a particular framework (typically 3G), where membership is controlled and system behavior is standardized. There are no known issues with using

explicit routing in such an environment.

However, if decorated identifiers are used, such as:

```
homerealm.example.org!user@otherrealm.example.net
```

Then the part before the (non-escaped) '!' MUST be a "utf8-realm" as defined in the ABNF in Section 2.2. When receiving such an identifier, the "otherrealm.example.net" system MUST convert the identifier to "user@homerealm.example.org" before forwarding the request. The forwarding system MUST then apply normal AAA routing for the transaction, based on the updated identifier.

3.4. Examples

Examples of valid Network Access Identifiers include the following:

```
bob
joe@example.com
fred@foo-9.example.com
jack@3rd.depts.example.com
fred.smith@example.com
fred_smith@example.com
fred$@example.com
fred=?#&*+~/^smith@example.com
nancy@eng.example.net
eng.example.net!nancy@example.net
eng%nancy@example.net
@privatecorp.example.net
\(user\)@example.net
```

An additional valid NAI is the following, given as a hex string, as this document can only contain ASCII characters.

```
626f 6240 ceb4 cebf ceba ceb9 cebc ceae 2e63 6f6d
```

Examples of invalid Network Access Identifiers include the following:

```
fred@example
fred@example_9.com
fred@example.net@example.net
fred.@example.net
eng:nancy@example.net
eng;nancy@example.net
(user)@example.net
<nancy>@example.net
```

One example given in [RFC4282] is still permitted by the ABNF, but it

is NOT RECOMMENDED because of the use of the Punycode [RFC3492] encoding form for what is now a valid UTF-8 string.

alice@xn--tmonesimerkki-bfbb.example.net

4. Security Considerations

Since an NAI reveals the home affiliation of a user, it may assist an attacker in further probing the username space. Typically, this problem is of most concern in protocols that transmit the username in clear-text across the Internet, such as in RADIUS, described in [RFC2865] and [RFC2866]. In order to prevent snooping of the username, protocols may use confidentiality services provided by protocols transporting them, such as RADIUS protected by IPsec [RFC3579] or Diameter protected by TLS [RFC6733].

This specification adds the possibility of hiding the username part in the NAI, by omitting it. As discussed in Section 2.4, this is possible only when NAIs are used together with a separate authentication method that can transfer the username in a secure manner. In some cases, application-specific privacy mechanism have also been used with NAIs. For instance, some EAP methods apply method-specific pseudonyms in the username part of the NAI [RFC3748]. While neither of these approaches can protect the realm part, their advantage over transport protection is that privacy of the username is protected, even through intermediate nodes such as NASes.

4.1. Correlation of Identities over Time and Protocols

The recommendations in Section 2.7 and Section 2.8 for using the NAI in other protocols has implications for privacy. Any attacker who is capable of observing traffic containing the NAI can track the user, and correlate his activity across time and across multiple protocols. The authentication credentials therefore SHOULD be transported over channels which permit private communications, or multiple identifiers SHOULD be used, so that user tracking is impossible.

It is RECOMMENDED that user privacy be enhanced by configuring multiple identifiers for one user. These identifiers can be changed over time, in order to make user tracking more difficult for a malicious observer. However, provisioning and management of the identifiers may be difficult in to do in practice, which is likely why multiple identifiers are rarely used today.

4.2. Multiple Identifiers

Section 1.3 states that multiple identifier formats allow attackers to make contradictory claims without being detected. This statement

deserves further discussion.

Section 2.4 discussed "inner" and "outer" identifiers in the context of TTLS [RFC5281]. A close reading of that specification shows there is no requirement that the inner and outer identifiers be in any way related. That is, it is perfectly valid to use "@example.com" for an outer identifier, and "user@example.org" as an inner identifier. The authentication request will then be routed to "example.com", which will likely be unable to authenticate "user@example.org".

Even worse, a misconfiguration of "example.com" means that it may in turn proxy the inner authentication request to the "example.org" domain. Such cross-domain authentication is highly problematic, and there are few good reasons to allow it.

It is therefore RECOMMENDED that systems which permit anonymous "outer" identifiers require that the "inner" domain be the same as, or a sub-domain of the "outer" domain. An authentication request using disparate realms is a security violation, and the request SHOULD be rejected.

The situation gets worse when multiple protocols are involved. The TTLS protocol permits MS-CHAP [RFC2433] to be carried inside of the TLS tunnel. MS-CHAP defines its own identifier which is encapsulated inside of the MS-CHAP exchange. That identifier is not required to be any particular format, is not required to be in UTF-8, and in practice, can be one of many unknown character sets. There is no way in practice to determine which character set was used for that identifier.

The result is that the "outer" EAP Identity carried by TTLS is likely to not even share the same character set as the "inner" identifier used by MS-CHAP. The two identifiers are entirely independent, and fundamentally incomparable.

Such protocol design is NOT RECOMMENDED.

5. Administration of Names

In order to avoid creating any new administrative procedures, administration of the NAI realm namespace piggybacks on the administration of the DNS namespace.

NAI realm names are required to be unique, and the rights to use a given NAI realm for roaming purposes are obtained coincident with acquiring the rights to use a particular Fully Qualified Domain Name (FQDN). Those wishing to use an NAI realm name should first acquire the rights to use the corresponding FQDN. Administrators MUST NOT

publicly use an NAI realm without first owning the corresponding FQDN. Private use of unowned NAI realms within an administrative domain is allowed, though it is RECOMMENDED that example names be used, such as "example.com".

Note that the use of an FQDN as the realm name does not require use of the DNS for location of the authentication server. While Diameter [RFC6733] supports the use of DNS for location of authentication servers, existing RADIUS implementations typically use proxy configuration files in order to locate authentication servers within a domain and perform authentication routing. The implementations described in [RFC2194] did not use DNS for location of the authentication server within a domain. Similarly, existing implementations have not found a need for dynamic routing protocols or propagation of global routing information. Note also that there is no requirement that the NAI represent a valid email address.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

[RFC2119]

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

[RFC3629]

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5198]

Klensin J., and Padlipsky M., "Unicode Format for Network Interchange", RFC 5198, March 2008

[RFC5234]

Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008.

[RFC5890]

Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 5890, August 2010

[RFC5891]

Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010

[RFC6365]

Hoffman, P., and Klensin, J., "Terminology Used in Internationalization in the IETF", RFC 6365, September 2011

7.2. Informative References

[RFC2194]

Aboba, B., Lu, J., Alsop, J., Ding, J., and W. Wang, "Review of Roaming Implementations", RFC 2194, September 1997.

[RFC2341]

Valencia, A., Littlewood, M., and T. Kolar, "Cisco Layer Two Forwarding (Protocol) "L2F"", RFC 2341, May 1998.

[RFC2433]

Zorn G., and Cobb, S. "Microsoft PPP CHAP Extensions", RFC 2433, October 1998.

[RFC2637]

Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol", RFC 2637, July 1999.

[RFC2661]

Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, August 1999.

[RFC2865]

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

[RFC2866]

Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

[RFC3492]

Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

[RFC3579]

Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RFC3748]

Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

- [RFC4282]
Aboba, B. et al., "The Network Access Identifier", RFC 4282, December 2005.
- [RFC4301]
Kent, S. and S. Keo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC5281]
Funk, P., and Blake-Wilson, S., "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.
- [RFC5322]
Resnick, P. (Ed), "Internet Message Format", RFC 5322, October 2008.
- [RFC5335]
Y. Abel, Ed., "Internationalized Email Headers", RFC 5335, September 2008.
- [RFC5729]
Korhonen, J. (Ed) et. al., "Clarifications on the Routing of Diameter Requests Based on the Username and the Realm", RFC 5729, December 2009
- [RFC6055]
Thaler, D., et al, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.
- [RFC6532]
Yang, A., et al, "Internationalized Email Headers", RFC 6532, February 2012.
- [RFC6733]
V. Fajardo, Ed., et al, "Diameter Base Protocol", RFC 6733, October 2012.
- [RFC6912]
Sullivan, A., et al, "Principles for Unicode Code Point Inclusion in Labels in the DNS", RFC 6912, April 2013.
- [EDUROAM]
<http://eduroam.org>, "eduroam (EDUcational ROAMing)"
- [3GPP]
3GPP, "TS 23.003 Numbering, addressing, and Identification (Release 12)", July 2014,

ftp://ftp.3gpp.org/Specs/archive/23_series/23.003/.

Acknowledgments

The initial text for this document was [RFC4282], which was then heavily edited. The original authors of [RFC4282] were Bernard Aboba, Mark A. Beadles, Jari Arkko, and Pasi Eronen.

The ABNF validator at <http://www.apps.ietf.org/abnf.html> was used to verify the syntactic correctness of the ABNF in Section 2.

Appendix A - Changes from RFC4282

This document contains the following updates with respect to the previous NAI definition in RFC 4282 [RFC4282]:

- * The formal syntax in Section 2.1 has been updated to forbid non-UTF8 characters. e.g. characters with the "high bit" set.
- * The formal syntax in Section 2.1 has been updated to allow UTF-8 in the "realm" portion of the NAI.
- * The formal syntax in [RFC4282] Section 2.1 applied to the NAI after it was "internationalized" via the ToAscii function. The contents of the NAI before it was "internationalized" were left indeterminate. This document updates the formal syntax to define an internationalized form of the NAI, and forbids the use of the ToAscii function for NAI "internationalization".
- * The grammar for the user and realm portion is based on a combination of the "nai" defined in [RFC4282] Section 2.1, and the "utf8-addr-spec" defined in [RFC5335] Section 4.4.
- * All use of the ToAscii function has been moved to normal requirements on DNS implementations when realms are used as the basis for DNS lookups. This involves no changes to the existing DNS infrastructure.
- * The discussions on internationalized character sets in Section 2.4 have been updated. The suggestion to use the ToAscii function for realm comparisons has been removed. No AAA system has implemented these suggestions, so this change should have no operational impact.
- * The section "Routing inside of AAA Systems" section is new in this document. The concept of a "local AAA routing table" is also new, although it accurately describes the functionality of wide-spread implementations.
- * The "Compatibility with EMail Usernames" and "Compatibility with DNS" sections have been revised and updated. The Punycode transformation is suggested to be used only when a realm name is used for DNS lookups, and even then the function is only used by a resolving API on the local system, and even then it is recommended that only the home network perform this conversion.
- * The "Realm Construction" section has been updated to note that editing of the NAI is NOT RECOMMENDED.

- * The "Examples" section has been updated to remove the instance of the IDN being converted to ASCII. This behavior is now forbidden.

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org

RADIUS EXTensions Working Group
Internet-Draft
Intended status: Experimental
Expires: July 30, 2015

A. Perez-Mendez
R. Marin-Lopez
F. Pereniguez-Garcia
G. Lopez-Millan
University of Murcia
D. Lopez
Telefonica I+D
A. DeKok
Network RADIUS
January 26, 2015

Support of fragmentation of RADIUS packets
draft-ietf-radext-radius-fragmentation-12

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 RADIUS 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 July 30, 2015.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Requirements Language	6
2. Status of this document	6
3. Scope of this document	7
4. Overview	10
5. Fragmentation of packets	12
5.1. Pre-authorization	13
5.2. Post-authorization	17
6. Chunk size	20
7. Allowed large packet size	21
8. Handling special attributes	22
8.1. Proxy-State attribute	22
8.2. State attribute	23
8.3. Service-Type attribute	24
8.4. Rebuilding the original large packet	24
9. New flag T field for the Long Extended Type attribute definition	24
10. New attribute definition	25
10.1. Frag-Status attribute	25
10.2. Proxy-State-Length attribute	26
10.3. Table of attributes	27
11. Operation with proxies	27
11.1. Legacy proxies	28
11.2. Updated proxies	28
12. General considerations	30
12.1. Flag T	30
12.2. Violation of RFC2865	31
12.3. Proxying based on User-Name	31
12.4. Transport behaviour	31
13. Security Considerations	32
14. IANA Considerations	32
15. Acknowledgements	33
16. References	34
16.1. Normative References	34
16.2. Informative References	34

Authors' Addresses	35
------------------------------	----

1. Introduction

The RADIUS [RFC2865] protocol carries authentication, authorization, and accounting information between a RADIUS Client and an RADIUS Server. Information is exchanged between them 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 intention of the above packet size limitation was to avoid as much as possible UDP fragmentation. Back then, 4096 seemed large enough for any purpose. Now, new scenarios are emerging that require the exchange of authorization information exceeding this 4096 limit. For instance, the Application Bridging for Federated Access Beyond web (ABFAB) IETF WG defines the transport of Security Assertion Markup Language (SAML) sentences from the RADIUS server to the RADIUS client [I-D.ietf-abfab-aaa-saml]. This assertion is likely to be larger than 4096 octets.

This means that peers desiring to send large amounts of data must fragment it across multiple packets. For example, RADIUS-EAP [RFC3579] defines how an Extensible Authentication Protocol (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 RADIUS Server is large and highly dynamic. In other cases, the RADIUS Client needs to send large amounts of authorization data to the RADIUS 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 RADIUS Client and Server, with no accompanying solution.

[RFC6158] section 3.1 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 (utilization of a sequence of packets) does not talk about large amounts of data sent from the RADIUS Client to a RADIUS 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 not entirely solve the problem of sending large amounts of data from a RADIUS Server to a RADIUS Client, as many current RADIUS attributes are not permitted in an Access-Challenge packets.

- o The second approach (utilization of names rather than values) 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 statement 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 setup of an additional trust infrastructure (e.g. Public Key Infrastructure - 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 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 [RFC6929]). It is also transparent to existing RADIUS proxies, which do not implement this specification. The only systems needing to implement this RFC 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.

A different approach to deal with RADIUS packets above the 4096 octet limit is described in [I-D.ietf-radext-bigger-packets], which proposes to extend RADIUS over TCP by allowing the length field in the RADIUS header to take values up to 65535 octets. This provides a simpler operation, but it has the drawback of requiring every RADIUS proxy in the path between the RADIUS client and the RADIUS server to implement the extension as well.

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]. When these words appear in lower case, they have their natural language meaning.

2. Status of this document

This document is an Experimental RFC. It defines a proposal to allow sending and receiving data exceeding the 4096 octet limit in RADIUS packets imposed by [RFC2865], without requiring the modification of intermediary proxies.

The experiment consists in verifying whether the approach is usable on a large scale environment, by observing the uptake, usability, and operational behavior it shows in large-scale, real-life deployments. In that sense, so far the main use case for this specification is the transportation of large SAML sentences defined within the ABFAB architecture [I-D.ietf-abfab-arch]. Hence, it can be tested wherever an ABFAB deployment is being piloted.

Besides, this proposal defines some experimental features that will need to be tested and verified before the document can be considered for Standards Track. The first one of them is the requirement of updating [RFC2865] in order to relax the sentence defined in Section 4.1 and stating that "An Access-Request MUST contain either a User-Password or a CHAP- Password or a State". This specification might generate Access-Request packets without any of these attributes. Although all known implementations have chosen the philosophy of "be liberal in what you accept", we need to gain more operational experience to verify that unmodified proxies do not drop this kind of packets. More details on this aspect can be found in Section 12.2.

Another experimental feature of this specification is that it requires proxies to base their routing decisions on the value of the RADIUS User-Name attribute. Our experience is that this is the common behaviour, thus no issues are expected. However, it needs to be confirmed after using different implementations of intermediate proxies. More details on this aspect can be found in Section 12.3.

Moreover, this document requires two minor updates to Standards Track documents. First, it modifies the definition of the "Reserved" field of the "Long Extended Type" attribute [RFC6929], by allocating an additional flag "T". No issues are expected with this update, although some proxies might drop packets that does not have the

"Reserved" field set to 0. More details on this aspect can be found in Section 12.1.

The other Standards Track document that requires a minor update is [RFC6158]. It states that "attribute designers SHOULD NOT assume that a RADIUS implementation can successfully process RADIUS packets larger than 4096 octets", something no longer true if this document advances.

A proper "Updates" clause will be included for these modifications when/if the experiment is successful and this document is re-issued as a Standards Track document.

3. Scope of this document

This specification describes how a RADIUS Client and a RADIUS Server can exchange data exceeding the 4096 octet limit imposed by one packet. However, the mechanism described in this specification SHOULD NOT be used to exchange more than 100 kilo-octets of data. Any more than this may turn RADIUS into a generic transport protocol, such as TCP or SCTP, which is undesired. Experience shows that attempts to transport bulk data across the Internet with UDP will inevitably fail, unless they re-implement all of the behavior of TCP. The underlying design of RADIUS lacks the proper retransmission policies or congestion control mechanisms which would make it a competitor to TCP.

Therefore, RADIUS/UDP transport is by design unable to transport bulk data. It is both undesired and impossible to change the protocol at this point in time. This specification is intended to allow the transport of more than 4096 octets of data through existing RADIUS/UDP proxies. Other solutions such as RADIUS/TCP MUST be used when a "green field" deployment requires the transport of bulk data.

Section 7, below, describes with further details the reasoning for this limitation, and recommends administrators to adjust it according to the specific capabilities of their existing systems in terms of memory and processing power.

Moreover, 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, it is 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 4 kilooctets 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 RADIUS 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 Change of Authorization (CoA) [RFC5176] packets. Instead, according to [RFC5176] the CoA client will 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.

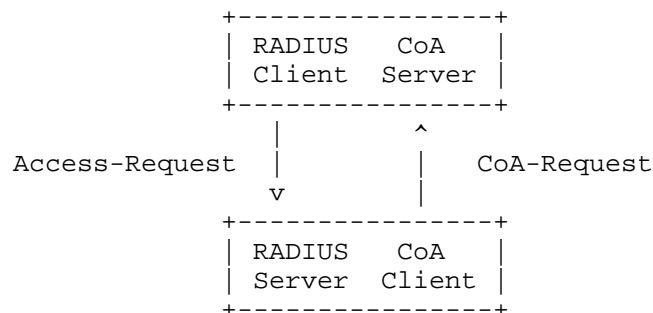
The above requirement does not assume that the CoA client and the RADIUS Server are co-located. They may, in fact be run on separate parts of the infrastructure, or even by separate administrators. There is, however, a requirement that the two communicate. We can see that the CoA client needs to send session identification attributes in order to send CoA packets. These attributes cannot be known a priori by the CoA client, and can only come from the RADIUS Server. Therefore, even when the two systems are not co-located, they must be able to communicate in order to operate in unison. The alternative is for the two systems to have differing views of the users authorization parameters, which is a security disaster.

This specification does not allow for fragmentation of CoA packets. Allowing for fragmented CoA packets would involve changing multiple parts of the RADIUS protocol, with the corresponding possibility for implementation issues, mistakes, etc.

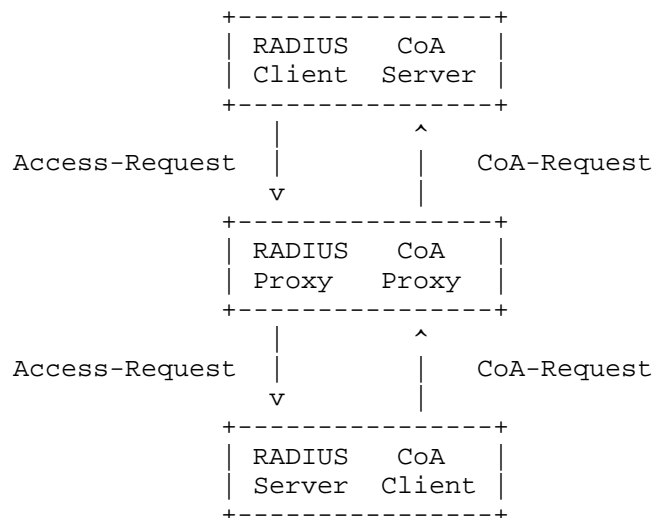
Where CoA clients (i.e. RADIUS Servers) need to send large amounts of authorization data to a CoA server (i.e. RADIUS Client), they need only send a minimal CoA-Request packet, containing Service-Type of Authorize-Only, as per [RFC5176], along with session identification attributes. This CoA packet serves as a signal to the RADIUS Client that the users' session requires re-authorization. When the RADIUS Client re-authorizes the user via Access-Request, the RADIUS Server can perform fragmentation, and send large amounts of authorization data to the RADIUS Client.

The assumption in the above scenario is that the CoA client and RADIUS Server are co-located, or at least strongly coupled. That is, the path from CoA client to CoA server SHOULD be the exact reverse of

the path from RADIUS Client to RADIUS Server. The following diagram will hopefully clarify the roles:



Where there is a proxy involved:



That is, the RADIUS and CoA subsystems at each hop are strongly connected. Where they are not strongly connected, it will be impossible to use CoA-Request packets to transport large amounts of authorization data.

This design is more complicated than allowing for fragmented CoA packets. However, the CoA client and the RADIUS Server must communicate even when not using this specification. We believe that standardizing that communication, and using one method for exchange of large data is preferred to unspecified communication methods and multiple ways of achieving the same result. If we were to allow fragmentation of data over CoA packets, the size and complexity of

this specification would increase significantly.

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.

4. 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-authorization" as the exchange of authorization information before the end user authentication has started (from the RADIUS Client to the RADIUS Server), whereas the term "post-authorization" is used to refer to an authorization exchange happening after this authentication process (from the RADIUS Server to the RADIUS Client).

In this specification we refer to the "size limit" as the practical limit on RADIUS packet sizes. This limit is the minimum between 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 RADIUS Client 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 the packet to be proxied if necessary (see Section 11.1). 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 RADIUS Client in a later Access-Request. Each Access-Accept contains a Service-Type attribute with the "Additional-Authorization" value. This indicates that the service being provided is part of a fragmented exchange, and that the Access-Accept should not be interpreted as providing network access to the end user.

When a RADIUS Client or RADIUS 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 of the same type, 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 the RADIUS Client and RADIUS Server, and the overhead for fragmentation signalling attributes. Specific details are given in Section 6. A new attribute called Frag-Status (Section 10.1) 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.

The reconstruction process is performed by simply appending all of the chunks together. Unlike IPv4 fragmentation, there is no "fragment offset" field. The chunks in this specification are explicitly ordered, as RADIUS is a lock-step protocol, as noted in Section Section 12.4. That is, chunk N+1 cannot be sent until all of the chunks up to and including N have been received and acknowledged.

When multiple chunks are sent, a special situation may occur for Extended Type attributes as defined in [RFC6929]. 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 using the newly defined flag "T" in the Reserved field of the "Long Extended Type" attribute format (see Section 9 for further details on this flag).

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".

5. Fragmentation of packets

When the RADIUS Client or the RADIUS Server 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 delivery of a large fragmented RADIUS packet with authorization data can happen before or after the end user has been authenticated by the RADIUS Server. We can distinguish two phases, which can be omitted if there is no authorization data to be sent:

1. Pre-authorization. In this phase, the RADIUS Client MAY send a large packet with authorization information to the RADIUS Server before the end user is authenticated. Only the RADIUS Client is allowed to send authorization data during this phase.
2. Post-authorization. In this phase, the RADIUS Server MAY send a large packet with authorization data to the RADIUS Client after the end user has been authenticated. Only the RADIUS Server is allowed to send authorization data during this phase.

The following subsections describe how to perform fragmentation for packets for these two phases, pre-authorization and post-authorization. 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.

5.1. Pre-authorization

When the RADIUS Client needs to send a large amount of data to the RADIUS Server, the data to be sent is split into chunks and sent to the RADIUS Server via multiple Access-Request / Access-Accept exchanges. The example below shows this exchange.

The following is an Access-Request which the RADIUS Client intends to send to a RADIUS 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
  NAS-Identifier
  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
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
```

Figure 1: Desired Access-Request

The RADIUS Client 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 RADIUS Client wishes to send more data in a subsequent Access-Request. The RADIUS Client 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
  NAS-Identifier
  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 RADIUS Servers (i.e. servers implementing fragmentation) receiving this packet will see the Frag-Status attribute, and postpone all authorization and authentication handling until all of the chunks have been received. This postponement also affects to the verification that the Access-Request packet contains some kind of authentication attribute (e.g. User-Password, CHAP-Password, State or other future attribute), as required by [RFC2865] (see Section 12.2 for more information on this).

Non-compliant RADIUS Servers (i.e. servers not implementing fragmentation) should also see the Service-Type requesting provisioning for an unknown service, and return Access-Reject. Other non-compliant RADIUS Servers may return an Access-Reject, Access-Challenge, or an Access-Accept with a particular Service-Type other than Additional-Authorization. Compliant RADIUS Client implementations MUST treat these responses as if they had received Access-Reject instead.

Compliant RADIUS 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 temporally unique to the RADIUS Server.

```
Access-Accept (ID = 23)
  Frag-Status = More-Data-Request
  Service-Type = Additional-Authorization
  State = 0xabc00001
  Message-Authenticator
```

Figure 3: Access-Accept (chunk 1)

The RADIUS Client 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 RADIUS Client 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 (see Section 11.1). 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-Pending
  Service-Type = Additional-Authorization
  State = 0xabc000001
  Message-Authenticator
```

Figure 4: Access-Request (chunk 2)

Compliant RADIUS 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 RADIUS Server will associate the attributes with the previous chunk. Since the Frag-Status attribute has value More-Data-Request, the RADIUS 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 RADIUS Client will see this response, and use the RADIUS Identifier field to associate it with an ongoing chunking session. The RADIUS Client 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 RADIUS Client 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 RADIUS Server matches it with an ongoing session via the State attribute, 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 8.4 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 RADIUS 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 (see Section 12.2 for some considerations about this), 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 RADIUS Client 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 RADIUS Client 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 RADIUS Client 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.

5.2. Post-authorization

When the RADIUS Server wants to send a large amount of authorization data to the RADIUS Client after authentication, the operation is very similar to the pre-authorization one. The presence of Service-Type = Additional-Authorization attribute ensures that a RADIUS Client 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. It is RECOMMENDED that RADIUS Servers include a State attribute on their original Access-Accept packets, even if fragmentation is not taking place, to allow the RADIUS Client to send additional authorization data in subsequent exchanges. This State attribute would be included in the last transmitted chunk, to avoid confusion with the ones used for fragmentation signalling.

Client supporting this specification MUST include a Frag-Status = Fragmentation-Supported attribute in the first Access-Request sent to the RADIUS 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 RADIUS Server intends to send to a RADIUS 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
  Example-Long-2 [M]
  Example-Long-2 [M]
  Example-Long-2
  State = 0xcba00003
```

Figure 8: Desired Access-Accept

The RADIUS Server therefore must send the attributes listed above in a series of chunks. The first chunk contains seven (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 RADIUS Server wishes to send more data in a subsequent Access-Accept. The RADIUS Server 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 RADIUS Clients receiving this packet will see the Frag-Status attribute, and suspend all authorization handling until all of the chunks have been received. Non-compliant RADIUS Clients should also see the Service-Type indicating the provisioning for an unknown service, and will treat it as an Access-Reject.

RADIUS 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 (Section 11.1).

```
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 RADIUS Server 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 RADIUS Server 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 RADIUS Server includes the original State attribute to allow the RADIUS

Client to send additional authorization data. The original Service-Type attribute is included as well.

```
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 = 0xfda000003
  Message-Authenticator
```

Figure 11: Access-Accept (chunk 2)

On reception of this last chunk, the RADIUS 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 8.4 for further details of this process.

6. 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 RADIUS 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 RADIUS 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 8.1.

7. 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 end users are permitted to trigger the exchange of large amounts of fragmented data between the RADIUS Client and the RADIUS Server, having the potential to allow Denial of Service (DoS) attacks. An attacker could initiate a large number of connections, each of which requests the RADIUS Server to store a large amount of data. This data could cause memory exhaustion on the RADIUS 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. Its default value SHOULD be 100 kilo-octets. Any more than this may turn RADIUS into a generic transport protocol, which is undesired. This limit SHOULD be configurable, 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. Its default value SHOULD be to 25. Any more than this may indicate an implementation error, misconfiguration, or a denial of service (DoS) attack. This limit SHOULD be configurable, so that it can be changed if necessary.

For instance, let's imagine the RADIUS Server wants to transport an SAML assertion which is 15000 octets long, to the RADIUS Client. In this hypothetical scenario, we assume there are 3 intermediate proxies, each one inserting a Proxy-State attribute of 20 octets. Also we assume the State attributes generated by the RADIUS Server have a size of 6 octets, and the User-Name attribute take 50 octets. Therefore, the amount of free space in a chunk for the transport of the SAML assertion attributes is: Total (4096) - RADIUS header (20) - User-Name (50 octets) - Frag-Status (7 octets) - Service-Type (6 octets) - State (6 octets) - Proxy-State (20 octets) - Proxy-State (20) - Proxy-State (20) - Message-Authenticator (18 octets), resulting in a total of 3929 octets, that is, 15 attributes of 255 bytes.

According to [RFC6929], a Long-Extended-Type provides a payload of 251 octets. Therefore, the SAML assertion described above would result into 60 attributes, requiring of 4 round-trips to be completely transmitted.

8. Handling special attributes

8.1. Proxy-State attribute

RADIUS proxies may introduce Proxy-State attributes into any Access-Request packet they forward. If they are unable to 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 RADIUS 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 much 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, RADIUS 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 much 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 RADIUS 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 RADIUS Client in the next reply packet, encoded into a new attribute called Proxy-State-Length. The RADIUS Server MAY artificially increase this quantity in order to handle with situations where proxies behave inconsistently (e.g. they generate Proxy-State attributes with a different size for each packet), or for situations where intermediary proxies remove Proxy-State attributes generated by other proxies. Increasing this value would make the RADIUS Client to leave some free space for these situations.
3. The RADIUS Client SHOULD react upon the reception of this attribute by adjusting the maximum size for the next chunk accordingly. However, as the Proxy-State-Length offers just an estimation of the space required by the proxies, the RADIUS Client MAY select a smaller amount in environments known to be problematic.

8.2. 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 5 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 RADIUS

Client MUST assure that this original State is included into the first chunk sent to the RADIUS Server (as this one never contains any State attribute due to fragmentation).

8.3. 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.

8.4. 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-Length

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 Service-Type = Additional-Authorization
- o Frag-Status
- o Proxy-State (except the ones in the last chunk)
- o User-Name (except the one in the first chunk)

9. New flag T field for the Long Extended Type attribute definition

This document defines a new field in 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). Proxies implementing [RFC6929] will see these attributes as invalid (they will not be able to reconstruct them), but they will still forward them as [RFC6929] section 5.2 indicates they SHOULD forward unknown attributes anyway.

As a consequence of this addition, the Reserved field is now 6 bits long (see Section 12.1 for some considerations). The following figure represents the new attribute format.

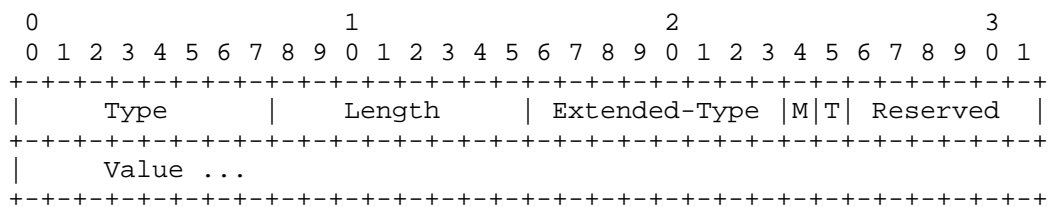


Figure 12: Updated Long Extended Type attribute format

10. New attribute definition

This document proposes the definition of two new extended type attributes, called Frag-Status and Proxy-State-Length. The format of these attributes follows the indications for an Extended Type attribute defined in [RFC6929].

10.1. 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.

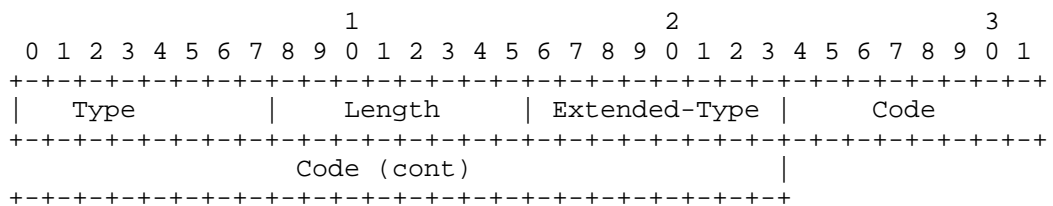


Figure 13: Frag-Status format

Type

241 (To be confirmed by IANA)

Length

7

Extended-Type

TBD1

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. RADIUS Clients supporting this specification MUST include a Frag-Status = Fragmentation-Supported attribute in the first Access-Request sent to the RADIUS Server, in order to indicate they would accept fragmented data from the sever.

10.2. Proxy-State-Length 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-Length attribute is the following:

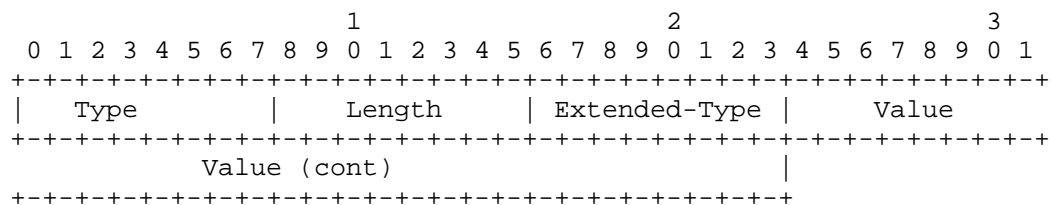


Figure 14: Proxy-State-Length format

Type

241 (To be confirmed by IANA)

Length

7

Extended-Type

TBD2

Value

4 octets. Total length (in octets) of received Proxy-State attributes (including headers). As the RADIUS "length" field cannot take values over 4096 octets, values of Proxy-State-Length MUST be less than that maximum length.

This attribute MAY be present in Access-Challenge and Access-Accept packets. It MUST NOT be included in Access-Request or Access-Reject packets.

10.3. 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-Length	0	0-1	0	0-1

11. 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.

11.1. 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 RADIUS 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.

11.2. Updated proxies

Updated proxies can interact with RADIUS Clients and Servers in order to obtain the complete large packet before starting 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 RADIUS Client to obtain a large Access-Request packet, modify an attribute resulting into an even more large packet, and interacts with the RADIUS Server to complete the transmission of the modified packet.

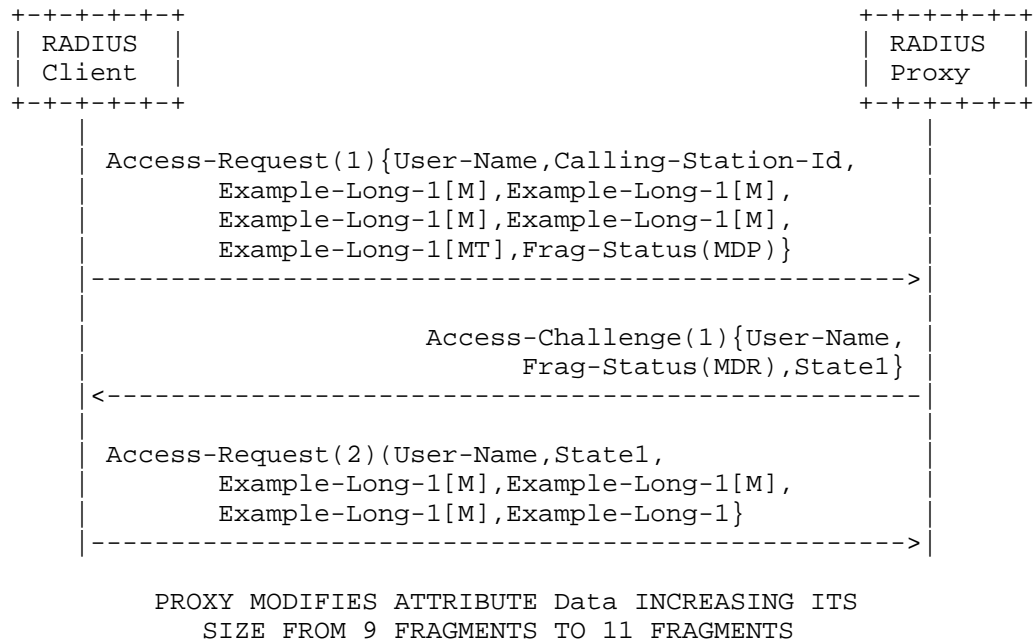


Figure 15: Updated proxy interacts with RADIUS Client

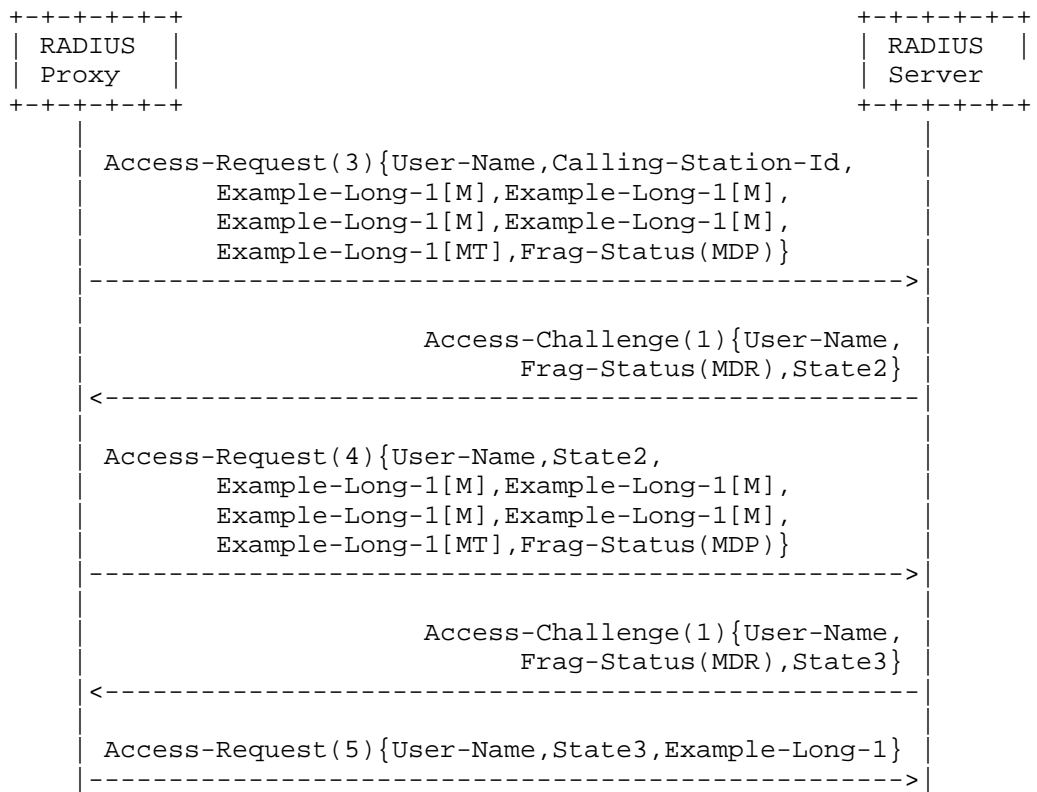


Figure 16: Updated proxy interacts with RADIUS Server

12. General considerations

12.1. Flag T

As described in Section 9, this document modifies the definition of the "Reserved" field of the "Long Extended Type" attribute [RFC6929], by allocating an additional flag "T". The meaning and position of this flag is defined in this document, and nowhere else. This might generate an issue if subsequent specifications want to allocate a new flag as well, as there would be no direct way for them to know which parts of the "Reserved" field have already been defined.

An immediate and reasonable solution for this issue would be declaring that this RFC updates [RFC6929]. In this way, [RFC6929] would include an "Updated by" clause that will point readers to this document. Another alternative would be creating an IANA registry for the "Reserved" field. However, the working group thinks that would

be overkill, as not such a great number of specifications extending that field are expected.

In the end, the proposed solution is that this experimental RFC should not update RFC 6929. Instead, we rely on the collective mind of the WG to recall that this T flag is used. When/if the experiment will be successful, the T flag will be properly assigned.

12.2. Violation of RFC2865

Section 5.1 indicates that all authorization and authentication handling will be postponed until all the chunks have been received. This postponement also affects to the verification that the Access-Request packet contains some kind of authentication attribute (e.g. User-Password, CHAP-Password, State or other future attribute), as required by [RFC2865]. This checking will therefore be delayed until the original large packet has been rebuilt, as some of the chunks may not contain any of them.

The authors acknowledge that this specification violates the "MUST" requirement of [RFC2865] Section 4.1 that states that "An Access-Request MUST contain either a User-Password or a CHAP-Password or a State". We note that a proxy which enforces that requirement would be unable to support future RADIUS authentication extensions. Extensions to the protocol would therefore be impossible to deploy. All known implementations have chosen the philosophy of "be liberal in what you accept". That is, they accept traffic which violates the requirement of [RFC2865] Section 4.1. We therefore expect to see no operational issues with this specification. After we gain more operational experience with this specification, it can be re-issued as a standards track document, and update [RFC2865].

12.3. Proxying based on User-Name

This proposal assumes legacy proxies to base their routing decisions on the value of the User-Name attribute. For this reason, every packet sent from the RADIUS Client to the RADIUS Server (either chunks or requests for more chunks) MUST contain a User-Name attribute.

12.4. Transport behaviour

This proposal does not modify the way RADIUS interacts with the underlying transport (UDP). That is, RADIUS keeps following a lock-step behaviour, that requires receiving an explicit acknowledge for each chunk sent. Hence, bursts of traffic which could congest links between peers are not an issue.

Another benefit of the lock-step nature of RADIUS, is that there are no security issues with overlapping fragments. Each chunk simply has a length, with no "fragment offset" field as with IPv4. The order of the fragments is determined by the order in which they are received. There is no ambiguity about the size or placement of each chunk, and therefore no security issues associated with overlapping chunks.

13. 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 authenticated 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. RADIUS 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 7 gives some indications on what could be 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 RADIUS Servers today already store large amounts of accounting data.

14. 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>.

In particular, this document defines two new RADIUS attributes, entitled "Frag-Status" and "Proxy-State-Length" (see Section 10), with assigned values of 241.TBD1 and 241.TBD2 from the Short Extended Space of [RFC6929]:

Type	Name	Length	Meaning
----	----	-----	-----
241.TBD1	Frag-Status	7	Signals fragmentation
241.TBD2	Proxy-State-Length	7	Indicates the length of the received Proxy-State attributes

The Frag-Status attribute also defines a 8-bit "Code" field, for which the IANA is to create and maintain a new sub-registry entitled "Code values" under the RADIUS "Frag-Status" attribute. Initial values for the RADIUS Frag-Status "Code" registry are given below; future assignments are to be made through "RFC required" [RFC5226]. Assignments consist of a Frag-Status "Code" name and its associated value.

Value	Frag-Status Code Name	Definition
----	-----	-----
0	Reserved	See Section 10.1
1	Fragmentation-Supported	See Section 10.1
2	More-Data-Pending	See Section 10.1
3	More-Data-Request	See Section 10.1
4-255	Unassigned	

Additionally, allocation of a new Service-Type value for "Additional-Authorization" is requested.

Value	Service Type Value	Definition
----	-----	-----
TBA	Additional-Authorization	See Section 5.1

15. Acknowledgements

The authors would like to thank the members of the RADEXT working group who have contributed to the development of this specification, either by participating on the discussions on the mailing lists or by sending comments about our RFC.

The authors also thank David Cuenca (University of Murcia) for implementing a proof of concept implementation of this RFC that has been useful to improve the quality of the specification.

This work has been partly funded by the GEANT GN3+ SA5 and CLASSe (<http://sec.cs.kent.ac.uk/CLASSe/>) projects.

16. References

16.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.
- [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.
- [RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, April 2013.

16.2. Informative References

- [I-D.ietf-abfab-aaa-saml]
Howlett, J. and S. Hartman, "A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for SAML", draft-ietf-abfab-aaa-saml-09 (work in progress), February 2014.
- [I-D.ietf-abfab-arch]
Howlett, J., Hartman, S., Tschofenig, H., Lear, E., and J. Schaad, "Application Bridging for Federated Access Beyond Web (ABFAB) Architecture", draft-ietf-abfab-arch-13 (work in progress), July 2014.
- [I-D.ietf-radext-bigger-packets]
Hartman, S., "Larger Packets for RADIUS over TCP", draft-ietf-radext-bigger-packets-01 (work in progress), July 2014.
- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

- [RFC4849] Congdon, P., Sanchez, M., and B. Aboba, "RADIUS Filter Rule Attribute", RFC 4849, April 2007.
- [RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.
- [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.

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>

