Workgroup: RADIUS EXTensions
Internet-Draft:
draft-ietf-radext-radiusdtls-bis-01
Obsoletes: 6614, 7360 (if approved)
Published: 18 April 2024
Intended Status: Standards Track
Expires: 20 October 2024
Authors: J.-F. Rieckers S. Winter
DFN RESTENA

(Datagram) Transport Layer Security ((D)TLS Encryption for RADIUS

Abstract

This document specifies a transport profile for RADIUS using Transport Layer Security (TLS) over TCP or Datagram Transport Layer Security (DTLS) over UDP as the transport protocol. This enables encrypting the RADIUS traffic as well as dynamic trust relationships between RADIUS servers. The specification obsoletes the experimental specifications in RFC 6614 (RADIUS/TLS) and RFC 7360 (RADIUS/DTLS) and combines them in this specification.

About This Document

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

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 20 October 2024.

Copyright Notice

Table of Contents

1. Introduction

The RADIUS protocol as described in [RFC2865], [RFC2866], [RFC5176] and others is a widely deployed authentication, authorization and accounting solution. However, the deployment experience has shown several shortcomings, as its dependency on the unreliable transport protocol UDP and the lack of confidentiality for large parts of its packet payload. Additionally the confidentiality and integrity mechanisms rely on the MD5 algorithm, which has been proven to be insecure. Although RADIUS/(D)TLS does not remove the MD5-based mechanisms, it adds confidentiality and integrity protection through the TLS layer. For an updated version of RADIUS/(D)TLS without need for MD5 see [I-D.ietf-radext-radiusv11]

1.1. Purpose of RADIUS/(D)TLS

The main focus of RADIUS/TLS and RADIUS/DTLS is to provide means to secure communication between RADIUS peers using TLS or DTLS. The most important use of this specification lies in roaming environments where RADIUS packets need to be sent across insecure or untrusted networks. An example for a worldwide roaming environment that uses RADIUS over TLS to secure communication is eduroam as described in [RFC7593]

1.2. Changes from RFC6614 (RADIUS/TLS) and RFC7360 (RADIUS/DTLS)

- [RFC6614] referenced [RFC6613] for TCP-related specification, RFC6613 on the other hand had some specification for RADIUS/TLS. These specifications have been merged into this document.

- RFC6614 marked TLSv1.1 or later as mandatory, this specification requires TLSv1.2 as minimum and recommends usage of TLSv1.3

- RFC6614 allowed usage of TLS compression, this document forbids it.

- RFC6614 only requires support for the trust model "certificates with PKIX". This document changes this. For servers, "certificates with PKIX" and "TLS-PSK" is now mandated and clients must implement one of the two.

- The mandatory-to-implement cipher suites are not referenced directly, this is replaced by a pointer to the TLS BCP.

- The specification regarding steps for certificate verification has been updated

- [RFC6613] mandated the use of Status-Server as watchdog algorithm, [RFC7360] only recommended it. This specification mandates the use of Status-Server for both RADIUS/TLS and RADIUS/DTLS.

The rationales behind some of these changes are outlined in Section 8.

2. Conventions and Definitions

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

Within this document we will use the following terms:

RADIUS/(D)TLS node:
a RADIUS-over-(D)TLS client or server

RADIUS/(D)TLS client: a RADIUS-over-(D)TLS instance that initiates a new connection

RADIUS/(D)TLS server: a RADIUS-over-(D)TLS instance that listens on a RADIUS-over-(D)TLS port and accepts new connections

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

Whenever "(D)TLS" or "RADIUS/(D)TLS" is mentioned, the specification applies for both RADIUS/TLS and RADIUS/DTLS. Where "TLS" or "RADIUS/TLS" is mentioned, the specification only applies to RADIUS/TLS, where "DTLS" or "RADIUS/DTLS" is mentioned it only applies to RADIUS/DTLS.

Server implementations **MUST** support both RADIUS/TLS and RADIUS/DTLS. Client implementations **SHOULD** implement both, but **MUST** implement at least one of RADIUS/TLS or RADIUS/DTLS.

3. Changes to RADIUS

This section discusses the needed changes to the RADIUS packet format (Section 3.1), port usage and shared secrets (Section 3.2).

3.1. Packet format

Source: RFC6613, Section 2.1 with minimal changes: Removed paragraph about required ability to store shared secrets. Also added last paragraphs from RFC 7360, Section 2.1

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/(D)TLS:

- *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

*Calculation of dynamic attributes such as CHAP-Challenge, or Message-Authenticator

*Calculation of "encrypted" attributes such as Tunnel-Password.

The use of (D)TLS transport does not change the calculation of security-related fields (such as the Response-Authenticator) in RADIUS [RFC2865] or RADIUS Dynamic Authorization [RFC5176]. Calculation of attributes such as User-Password [RFC2865] or Message-Authenticator [RFC3579] also does not change.

The changes to RADIUS implementations required to implement this specification are largely limited to the portions that send and receive packets on the network and the establishment of the (D)TLS connection.

The requirement that RADIUS remain largely unchanged ensures the simplest possible implementation and widest interoperability of the specification. This includes the usage of the outdated security mechanisms in RADIUS that are based on shared secrets and MD5. This is not considered a security issue, since integrity and confidentiality are provided by the (D)TLS layer. See Section 7 or [I-D.ietf-radext-radiusv11] for more details.

We note that for RADIUS/DTLS the DTLS encapsulation of RADIUS means that RADIUS packets have an additional overhead due to DTLS. This is discussed further in Section 6

3.2. Default ports and shared secrets

IANA has reserved ports for RADIUS/TLS and RADIUS/DTLS. Since authentication of peers, confidentiality, and integrity protection is achieved on the (D)TLS layer, the shared secret for the RADIUS packets is set to a static string, depending on the method. The calculation of security-related fields such as Response-Authenticator, Message-Authenticator or encrypted attributes **MUST** be performed using this shared secret.

Protocol	Port	Shared Secret
RADIUS/TLS	2083/tcp	"radsec"
RADIUS/DTLS	2083/udp	"radius/dtls"

Table 1

RADIUS/(D)TLS does not use separate ports for authentication, accounting and dynamic authorization changes. The source port is

arbitrary. For considerations regarding the multi-purpose use of one port for authentication and accounting see Section 4.4.

RADIUS/TLS servers **MUST** immediately start the TLS negotiation when a new connection to the RADIUS/TLS port is opened. They **MUST** close the connection and discard any data sent if the connecting client does not start a TLS negotiation or if the TLS negotiation fails at any point.

RADIUS/DTLS servers **MUST** silently discard any packet they receive over the RADIUS/DTLS port that is not a new DTLS negotiation or a packet sent over a DTLS session established earlier.

RADIUS/(D)TLS peers **MUST NOT** use the old RADIUS/UDP or RADIUS/TCP ports for RADIUS/DTLS or RADIUS/TLS.

3.3. Detecting Live Servers

Source: RFC6613, Section 2.4 with minor modifications, Last paragraph: RFC6613 Section 2.6.5.

As RADIUS is a "hop-by-hop" protocol, a RADIUS proxy shields the client from any information about downstream servers. While the client may be able to deduce the operational state of the local server (i.e., proxy), it cannot make any determination about the operational state of the downstream servers.

Within RADIUS, proxies typically only forward traffic between the NAS and RADIUS servers, and they do not generate their own response. As a result, when a NAS does not receive a response to a request, this could be the result of packet loss between the NAS and proxy, a problem on the proxy, loss between the RADIUS proxy and server, or a problem with the server.

The absence of a reply can cause a client to deduce (incorrectly) that the proxy is unavailable. The client could then fail over to another server or conclude that no "live" servers are available (OKAY state in [RFC3539], Appendix A). This situation is made even worse when requests are sent through a proxy to multiple destinations. Failures in one destination may result in service outages for other destinations, if the client erroneously believes that the proxy is unresponsive.

It is **REQUIRED** that implementations utilize the existence of a TCP/DTLS connection along with the application-layer watchdog defined in [RFC3539], Section 3.4 to determine the liveliness of the server.

RADIUS/(D)TLS clients **MUST** mark a connection DOWN if one or more of the following conditions are met:

- *The administrator has marked the connection administrative DOWN.
- *The network stack indicates that the connection is no longer viable.
- *The application-layer watchdog algorithm has marked it DOWN.

If a RADIUS/(D)TLS client has multiple connection to a server, it **MUST NOT** decide to mark the whole server as DOWN until all connections to it have been marked DOWN.TODO: Explain what a server is. (Just the destination IP? include port?)Janfred

It is **REQUIRED** that RADIUS/(D)TLS clients implement the Status-Server extension as described in [RFC5997] as the application level watchdog to detect the liveliness of the peer in the absence of responses. Since RADIUS has a limitation of 256 simultaneous "in flight" packets due to the length of the ID field ([RFC3539], Section 2.4), it is **RECOMMENDED** that RADIUS/(D)TLS clients reserve ID zero (0) on each session for Status-Server packets. This value was picked arbitrary, as there is no reason to choose any other value over another for this use.

For RADIUS/TLS, the peers **MAY** send TCP keepalives as described in [RFC9293], Section 3.8.4, for RADIUS/DTLS connections, the peers **MAY** send periodic keepalives as defined in [RFC6520], as a way of proactively and rapidly triggering a connection DOWN notification from the network stack. These liveliness checks are essentially redundant in the presence of an application-layer watchdog, but may provide more rapid notifications of connectivity issues.

4. Packet / Connection Handling

This section defines the behaviour for RADIUS/(D)TLS peers for handling of incoming packets and establishment of a (D)TLS session

4.1. (D)TLS requirements

Source: Mainly RFC6614, Section 2.3, Items 1 and 2, but without peer authentication models (in next section) or unnecessary text (e.g. MTI cipher suites, we just rely on the TLS cipher suites. Maybe explicitly mention that the MTI ciphers from TLS are also mandatory for this?)

As defined in Section 3.2, RAIDUS/(D)TLS clients must establish a (D)TLS session immediately upon connecting to a new server.

RADIUS/(D)TLS has no notion of negotiating (D)TLS in an ongoing communication. As RADIUS has no provisions for capability signaling, there is also no way for a server to indicate to a client that it should transition to using TLS or DTLS. Servers and clients need to be preconfigured to use RADIUS/(D)TLS for a given endpoint. This action has to be taken by the administrators of the two systems.

Implementations **MUST** follow the recommendations given in [RFC9325].TODO: Add text which recommendations of RFC9325 must be followed and whyJanfred Additionally, the following requirements have to be met for the (D)TLS session:

- Support for TLS 1.2 [RFC5248] / DTLS 1.2 [RFC6347] is **REQUIRED**, support for TLS 1.3 [RFC8446] / DTLS 1.3 [RFC9147] or higher is **RECOMMENDED**.

- Negotiation of a cipher suite providing for confidentiality as well as integrity protection is **REQUIRED**.

- The peers **MUST NOT** negotiate compression.

- The session **MUST** be mutually authenticated (see Section 4.2)

4.2. Mutual authentication

Source: RFC6614, Section 2.3, Item 3 with modifications.

RADIUS/(D)TLS servers **MUST** authenticate clients, and RADIUS/(D)TLS clients **MUST** authenticate the server. RADIUS is designed to be used by mutually trusted systems. Allowing anonymous clients would ensure privacy for RADIUS/(D)TLS traffic, but would negate all other security aspects of the protocol, including security aspects of RADIUS itself, due to the fixed shared secret.

RADIUS/(D)TLS allows for the following different modes of mutual authentication.

4.2.1. Authentication using X.509 certificates with PKIX trust model

All RADIUS/(D)TLS server implementations **MUST** implement this model. RADIUS/(D)TLS client implementations **SHOULD** implement this model, but **MUST** implement either this or TLS-PSK

If implemented it **MUST** use the following rules:

- Implementations **MUST** allow the configuration of a list of trusted Certificate Authorities for new TLS sessions.

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

*Implementations **SHOULD** indicate their trusted Certification authorities (CAs). See [RFC5246], Section 7.4.4 and [RFC6066], Section 6 for TLS 1.2 and [RFC8446], Section 4.2.4 for TLS 1.3.

*RADIUS/(D)TLS clients validate the servers identity to match their local configuration:

  -If the expected RADIUS/(D)TLS server was configured as a hostname, the configured name is matched against the presented names from the subjectAltName:DNS extension; if no such exist, against the presented CN component of the certificate subject

  -If the expected RADIUS/(D)TLS server was configured as an IP address, the configured IP address is matched against the presented addresses in the subjectAltName:iPAddr extension; if no such exist, against the presented CN component of the certificate subject.

  -If the RADIUS/(D)TLS server was not configured but discovered as per [RFC7585], the client executes the following checks in this order, accepting the certificate on the first match:

    oThe realm which was used as input to the discovery is matched against the presented realm names from the subjectAltName:naiRealm extension.

    oIf the discovery process yielded a hostname, this hostname is matched against the presented names from the subjectAltName:DNS extension; if no such exist, against the presented CN component of the certificate subject. Implementations **MAY** require the use of DNSSEC [RFC4033] to ensure the authenticity of the DNS result before relying on this for trust checks.

    oIf the previous checks fail, the certificate **MAY** Be accepted without further name checks immediately after the [RFC5280] trust chain checks, if configured by the administrator.

*RADIUS/(D)TLS servers validate the certificate of the RADIUS/(D)TLS client against a local database of acceptable clients. The database may enumerate acceptable clients either by IP address or by a name component in the certificate

  -For clients configured by DNS name, the configured name is matched against the presented names from the subjectAltName:DNS extension; if no such exist, against the presented CN component in the certificate subject.

- For clients configured by their source IP address, the configured IP address is matched against the presented addresses in the subjectAltName:iPAddr extension; if no such exist, against the presented CN component of the certificate subject. For clients configured by IP range, the certificate **MUST** be valid for the IP address the client is currently using.

- It is possible for a RADIUS/(D)TLS server to not require additional name checks for incoming RADIUS/(D)TLS clients, i.e. if the client used dynamic lookup. In this case, the certificate is accepted immediately after the [RFC5280] trust chain checks. This **MUST NOT** be used outside of trusted network environments or without additional certificate attribute checks in place.

* Implementations **MAY** allow a configuration of a set of additional properties of the certificate to check for a peer's authorization to communicate (e.g. a set of allowed values in subjectAltName:URI or a set of allowed X.509v3 Certificate Policies).

* When the configured trust base changes (e.g., removal of a CA from the list of trusted CAs; issuance of a new CRL for a given CA), implementations **SHOULD** renegotiate the TLS session to reassess the connecting peer's continued authorization.Open discussion: RFC6614 says "may" here. I think this should be a "should". There are some discussions to change this to "must". Input from TLS/UTA experts is appreciated.Janfred

4.2.2. Authentication using X.509 certificate fingerprints

RADIUS/(D)TLS implementations **SHOULD** allow the configuration of a list of trusted certificates, identified via fingerprint of the DER encoded certificate bytes. When implementing this model, support for SHA-1 as hash algorithm for the fingerprint is **REQUIRED**, and support for the more contemporary hash function SHA-256 is **RECOMMENDED**.

4.2.3. Authentication using Raw Public Keys

RADIUS/(D)TLS implementations **SHOULD** support using Raw Public Keys [RFC7250] for mutual authentication.

4.2.4. Authentication using TLS-PSK

RADIUS/(D)TLS server implementations **MUST** support the use of TLS-PSK. RADIUS/(D)TLS client implementations **SHOULD** support the use of TLS-PSK, but **MUST** implement either this or the "Authentication using X.509 certificates with PKIX" trust model.

Further guidance on the usage of TLS-PSK in RADIUS/(D)TLS is given in [I-D.ietf-radext-tls-psk].

4.3. Connecting Client Identity

Source: RFC6614, Section 2.4 with small modifications

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

Depending on the operation mode, the RADIUS/(D)TLS client identity can be determined differently.

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

In Raw-Public-Key operation, a client is uniquely identified by the Raw public key.

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

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

Note well: having identified a connecting entity does not mean the server necessarily wants to communicate with that client. For example, if the Issuer is not in a trusted set of Issuers, the server may decline to perform RADIUS transactions with this client.

Additionally, a server **MAY** restrict individual or groups of clients to certain IP ranges. A client connecting from outside this range would be rejected, even if the mutual authentication otherwise would have been successful. To reduce server load and to prevent probing the validity of stolen credentials, the server **SHOULD** abort the (D)TLS negotiation immediately with a TLS alert access_denied(49) after the client transmitted identifying information, i.e. the client certificate or the PSK identifier, and the server recognizes that the client connects from outside the allowed IP range.

There are numerous trust models in PKIX environments, and it is beyond the scope of this document to define how a particular deployment determines whether a client is trustworthy. Implementations that want to support a wide variety of trust models should expose as many details of the presented certificate to the

administrator as possible so that the trust model can be implemented by the administrator. As a suggestion, at least the following parameters of the X.509 client certificate should be exposed:

- *Originating IP address
- *Certificate Fingerprint
- *Issuer
- *Subject
- *all X.509v3 Extended Key Usage
- *all X.509v3 Subject Alternative Name
- *all X.509v3 Certificate Policy

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

- *Originating IP address
- *TLS-PSK Identifier

4.4. RADIUS Datagrams

Source: RFC 6614, Section 2.5 with small modifications and without example list

RADIUS/(D)TLS clients transmit the same packet types on the connection they initiated as a RADIUS/UDP client would, RADIUS/(D)TLS servers transmit the same packet types on the connections they have accepted as a RADIUS/UDP server would.

Due to the use of one single port for all packet types, it is required that a RADIUS/(D)TLS server signals which types of packets are supported on a server to a connecting peer.

- *When an unwanted packet of type 'CoA-Request' or 'Disconnect-Request' is received, a RADIUS/(D)TLS server needs to respond with a 'CoA-NAK' or 'Disconnect-AK', respectively. The NAK **SHOULD** contain an attribute Error-Cause with the value 406 ("Unsupported Extension"); see [RFC5176] for details.

- *When an unwanted packet of type 'Accounting-Request' is received, the RADIUS/(D)TLS server **SHOULD** reply with an Accounting-Response containing an Error-Cause attribute with value 406 "Unsupported Extensions" as defined in [RFC5176]. A RADIUS/(D)TLS accounting client receiving such an Accounting-Response **SHOULD** log the error

and stop sending Accounting-Request packets.TODO: Comment from Alan to send a Protocol Error packet instead.Janfred

4.5. Forwarding RADIUS packets between UDP and TCP based transports

Operating RADIUS proxies that use both UDP-based transports like RADIUS/UDP or RADIUS/DTLS and TCP-based transports like RADIUS/TLS requires different handing of packets. TCP based transports do not need retransmissions, since the reliable transport is provided by the TCP layer. Therefore, retransmission of RADIUS packets is forbidden over RADIUS/TLS. If a request is received over RADIUS/TLS and forwarded over RADIUS/UDP or RADIUS/DTLS, the proxy needs perform its own retransmissions for outstanding packets.

TODO: This section is currently a stub. Alan mentioned that we should have a section about handling this, especially around Accounting packets with Acct-Delay-Time. I need more text around this, help welcome.Janfred

5. RADIUS/TLS specific specifications

This section discusses all specifications that are only relevant for RADIUS/TLS.

5.1. Duplicates and Retransmissions

Source: RFC6613, Section 2.6.1, with small modifications

As TCP is a reliable transport, RADIUS/TLS peers **MUST NOT** retransmit RADIUS packets over a given TCP connection. Similarly, if there is no response to a RADIUS packet over one RADIUS/TLS connection, implementations **MUST NOT** retransmit that packet over a different connection to the same destination IP address and port, while the first connection is in the OKAY state ([RFC3539], Appendix A. TODO: Destination IP addr and port may be bad, but what is a server's identity?Janfred

However, if the TLS session or TCP connection is closed or broken, retransmissions over new connections are permissible. RADIUS request packets that have not yet received a response **MAY** be transmitted by a RADIUS/TLS client over a new connection. As this procedure involves using a new source port, the ID of the packet **MAY** change. If the ID changes, any security attributes such as Message-Authenticator **MUST** be recalculated.

If a TLS session or the underlying TCP connection is closed or broken, any cached RADIUS response packets ([RFC5080], Section 2.2.2) associated with that connection **MUST** be discarded. A RADIUS server **SHOULD** stop the processing of any requests associated with that TLS session. No response to these

requests can be sent over the TLS connection, so any further processing is pointless. This requirement applies not only to RADIUS servers, but also to proxies. When a client's connection to a proxy is closed, there may be responses from a home server that were supposed to be sent by the proxy back over that connection to the client. Since the client connection is closed, those responses from the home server to the proxy server **SHOULD** be silently discarded by the proxy.

Despite the above discussion, RADIUS servers **SHOULD** still perform duplicate detection on received packets, as described in [RFC5080], Section 2.2.2. This detection can prevent duplicate processing of packets from non-conforming clients.

RADIUS packets **SHOULD NOT** be retransmitted to the same destination IP an numerical port, but over a different transport protocol. There is no guarantee in RADIUS that the two ports are in any way related. This requirement does not, however, forbid the practice of putting multiple servers into a failover or load-balancing pool. In that situation, RADIUS requests **MAY** be retransmitted to another server that is known to be part of the same pool.

5.2. Malformed Packets and Unknown clients

Source: RFC 6613, Section 2.6.4 with small modifications.

The RADIUS specifications say that an implementation should "silently discard" a packet in a number of circumstances. This action has no further consequences for UDP based transports, as the "next" packet is completely independent of the previous one.

When TLS is used as transport, decoding the "next" packet on a connection depends on the proper decoding of the previous packet. As a result the behavior with respect to discarded packets has to change.

Implementations of this specification **SHOULD** tread the "silently discard" texts in the RADIUS specification referenced above as "silently discard and close the connection". That is, the implementation **SHOULD** send a TLS close notification and the underlying TCP connection **MUST** be closed if any of the following circumstances are seen:

- Connection from an unknown client
- Packet where the RADIUS "Length" field is less than the minimum RADIUS packet length
- Packet where the RADIUS "Length" field is more than the maximum RADIUS packet length

* Packet where an Attribute "Length" field has the value of zero or one (0 or 1)

* Packet where the attributes do not exactly fill the packet

* Packet where the Request Authenticator fails validation (where validation is required)

* Packet where the Response Authenticator fails validation (where validation is required)

* Packet where the Message-Authenticator attribute fails validation (when it occurs in a packet)

After applying the above rules, there are still two situations where the previous specifications allow a packet to be "silently discarded" upon receipt:

* Packet with an invalid code field

* Response packets that do not match any outstanding request

In these situations, the TCP connections **MAY** remain open, or they **MAY** be closed, as an implementation choice. However, the invalid packet **MUST** be silently discarded.

These requirements reduce the possibility for a misbehaving client or server to wreak havoc on the network.

5.3. TCP Applications Are Not UDP Applications

Source: RFC6613, Section 2.6.7 (TCP != UDP) and Section 2.6.2 (HoL-Blocking) with small modifications

Implementors should be aware that programming a robust TCP-based application can be very different from programming a robust UDP-based application.

Implementations **SHOULD** have configurable connection limits, configurable limits on connection lifetime and idle timeouts and a configurable rate limit on new connections. Allowing an unbounded number or rate of TCP/TLS connections may result in resource exhaustion.

Additionally, differences in the transport like Head of Line (HoL) blocking should be considered.

When using RADIUS/UDP or RADIUS/DTLS, there is no ordering of packets. If a packet sent by a peer is lost, that loss has no effect on subsequent packets sent by that peer.

Unlike UDP, TCP is subject to issues related to Head of Line blocking. This occurs when a TCP segment is lost and a subsequent TCP segment arrives out of order. While the RADIUS peers can process RADIUS packets out of order, the semantics of TCP makes this impossible. This limitation can lower the maximum packet processing rate of RADIUS/TLS.

6. RADIUS/DTLS specific specifications

This section discusses all specifications that are only relevant for RADIUS/DTLS.

6.1. RADIUS packet lengths

Source: RFC7360, Section 2.1, last paragraphs

The DTLS encryption adds an additional overhead to each packet sent. RADIUS/DTLS implementations **MUST** support sending and receiving RADIUS packets of 4096 bytes 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.

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 bytes outside the range of the length field as padding and ignore it on reception.

6.2. Server behavior

Source: RFC7360, Section 3.2 with small modifications

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

Some servers maintain a list of allowed clients per destination port. Others maintain a global list of clients that 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 the RADIUS/DTLS port. When packets are received fom a client with the "DTLS Required" flag set on non-DTLS ports, 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 the 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".

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

6.3. Client behavior

Source: RFC7360, Section 4

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

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.

Clients may implement "pools" of servers for fail-over or load-balancing. These pools **SHOULD NOT** mix RADIUS/UDP and RADIUS/DTLS servers.This paragraph should probably be moved, as it also applies to RADIUS/TLS. Mixing secure transports with insecure ones is bad practice, regardless of UDP or TCP.Janfred

6.4. Session Management

Source; RFC7360, Section 5

Where RADIUS/TLS can rely on the TCP state machine to perform session tracking, RADIUS/DTLS cannot. As a result, implementations of RADIUS/DTLS may need to perform session management of the DTLS session in the application layer. This subsection 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.

There are some specs (e.g. watchdog, stateless session resumption, closing session if malformed packet or security checks fail) which are valid for both server and client. It might be worth to just move them here instead of having them in both the client and the server spec.Janfred

6.4.1. Server Session Management

Source: RFC7360, Section 5.1

A RADIUS/DTLS server **MUST** track ongoing DTLS sessions for each client, based on 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 contains the following information:

DTLS Session:
Any information required to maintain and manage the DTLS session.

Last Traffic: A variable containing a timestamp that 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 that may contain information about the active DTLS session. This variable may be empty or nonexistent.

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

6.4.1.1. Session Opening and Closing

Source: RFC7360, Section 5.1.1 with small modifications

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 DTLS sessions that are partially set up. 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.TODO: Suggestion from Alan: "if closed for any reason", but not sure if this is what we mean.Jan fred When a session is deleted due to it failing security requirements, the DTLS session **MUST** be closed, 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 non-RADIUS packet is received over the DTLS connection, 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. A session **SHOULD NOT** be deleted when a well-formed, but "unexpected", RADIUS packet is received over it.

These requirements ensure the security while maintaining flexibility. Any security-related issue causes the connection to be

closed. After security restrictions have been applied, any unexpected traffic may be safely ignored, as it cannot cause a security issue. This allows for future extensions to the RADIUS/DTLS specifications.

As UDP does not guarantee delivery of messages, RADIUS/DTLS servers **MUST** 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, such as when packets are "silently discarded".

When a session has not received a packet for a period of time, it is labeled "idle". The server **SHOULD** delete idle DTLS sessions after an "idle timeout".RFC 7360 adds a paragraph about that the idle timeout should not be exposed to the admin as configurable parameter and references a mechanism to determine this value from the application-layer watchdog, but I didn't find the specification anywhere.Janfred

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 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 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 reusing 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 reuse 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.In RFC7360 there is a final paragraph about mitigation of the 4-tuple problem by using a local proxy. I'm not sure if this is the

right place here, i'd rather move that to a general "Implementation Guidelines" paragraph.Janfred

6.4.2. Client Session Management

Source: RFC7360, Section 5.2 with modifications

RADIUS/DTLS clients **SHOULD** use PMTU discovery [RFC6520] to determine the PMTU between the client and server, prior to sending any RADIUS traffic.

RADIUS/DTLS clients **SHOULD** proactively close sessions when they have been idle for a period of time. Clients **SHOULD** close a session when no traffic other than watchdog packet 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.

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.Maybe modify this text to be more similar to the TLS specific text here.Janfred

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.

7. Security Considerations

As this specification relies on the existing TLS and DTLS specifications, all security considerations for these protocols also apply to the (D)TLS portions of RADIUS/(D)TLS.

For RADIUS however, many security considerations raised in the RADIUS documents are related to RADIUS encryption and authorization. Those issues are largely mitigated when (D)TLS is used as a transport method, since encryption and authorization is achieved on the (D)TLS layer. The issues that are not mitigated by this

specification are related to the RADIUS packet format and handling, which is unchanged in this specification.

A few remaining security considerations and notes to administrators deploying RADIUS/(D)TLS are listed below.

7.1. RADIUS Proxies

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

Additionally, when RADIUS proxies are used, the RADIUS client has no way of ensuring that the complete path of the RADIUS packet is protected, since RADIUS routing is done hop-by-hop and any intermediate proxy may be configured, after receiving a RADIUS packet via RADIUS/(D)TLS from one peer, to forward this packet to a different peer using the RADIUS/UDP transport profile. There is no technical solution to this problem with the current specification. Where the confidentiality of the contents of the RADIUS packet across the whole path is required, organizational solutions need to be in place, that ensure that every intermediate RADIUS proxy is configured to forward the RADIUS packets using RADIUS/(D)TLS as transport.

TODO: Mabe add a reference to handling dynamic discovery (RFC7585) here too, and (as per Alans comments) that this issue is best resolved by limiting use of proxies.Janfred

7.2. Usage of null encryption cipher suites for debugging

For debugging purposes, some TLS implementation offer cipher suites with NULL encryption, to allow inspection of the plaintext with packet sniffing tools. Since with RADIUS/(D)TLS the RADIUS shared secret is set to a static string ("radsec" for RADIUS/TLS, "radius/dtls" for RADIUS/DTLS), using a NULL encryption cipher suite will also result in complete disclosure of the whole RADIUS packet, including the encrypted RADIUS attributes, to any intermediate IP node eavesdropping on the conversation. To prevent this, while keeping a NULL encryption cipher suite active, the only option is to set a different shared secret for RADIUS. In this case, the security considerations for confidentiality of RADIUS/UDP packets apply. Following the recommendations in [RFC9325], Section 4.1, this

specification forbids the usage of NULL encryption cipher suites for RADIUS/(D)TLS.

7.3. Possibility of Denial-of-Service attacks

Both RADIUS/TLS and RADIUS/DTLS have a considerable higher amount of data that the server needs to store in comparison to RADIUS/UDP. Therefore, an attacker could try to exhaust server resources.

With RADIUS/UDP, any bogous RADIUS packet would fail the cryptographic checks and the server would silently discard the bogous packet. For RADIUS/(D)TLS, the server needs to perform at least a partial TLS handshake to determine whether or not the client is authorized. Performing a (D)TLS handshake is more complex than the cryptographic check of a RADIUS packet. An attacker could try to trigger a high number of (D)TLS handshakes at the same time, resulting in a high server load and potentially a Denial-of-Service. To prevent this attack, a RADIUS/(D)TLS server **SHOULD** have configurable limits on new connection attempts.

Both TLS and DTLS need to store session information for each open (D)TLS session. Especially with DTLS, a bogous or misbehaving client could open an excessive number of DTLS sessions. This session tracking could lead to a resource exhaustion on the server side, triggering a Denial-of-Service. Therefore, RADIUS/(D)TLS servers **MUST** limit the absolute number of sessions they can track and **SHOULD** expose this limit as configurable option to the administrator. When the total number of sessions tracked is going to exceed the configured limit, servers **MAY** free up resources by closing the session that has been idle for the longest time. Doing so may free up idle resources that then allow the server to accept a new session.

RADIUS/DTLS servers **MUST** limit the number of partially open DTLS sessions and **SHOULD** expose this limit as configurable option to the administrator.

To prevent resource exhaustion by partially opening a large number of DTLS sessions, RADIUS/DTLS servers **SHOULD** have a timeout on partially open DTLS sessions. We recommend a limit of a few seconds, implementations **SHOULD** expose this timeout as configurable option to the administrator. If a DTLS session is not established within this timeframe, it is likely that this is a bogous connection. In contrast, an established session might not send packets for longer periods of time, but since the peers are mutually authenticated this does not pose a problem other than the problems mentioned before.

A different means of prevention is IP filtering. If the IP range that the server expects clients to connect from is restricted, then

the server can simply reject or drop all connection attempts from outside those ranges. If every RADIUS/(D)TLS client is configured with an IP range, then the server does not even have to perform a partial TLS handshake if the connection attempt comes from outside every allowed range, but can instead immediately drop the connection. To perform this lookup efficiently, RADIUS/(D)TLS servers **SHOULD** keep a list of the cummulated permitted IP ranges, individually for each transport.

7.4. Session Closing

If malformed RADIUS packets are received or the packets fail the authenticator checks, this specification requires that the (D)TLS session be closed. 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 potentially security risk. Similarly, any RADIUS traffic failing authentication vector or Message-Authenticator validation means that two parties do not have a common shared secret. Since the shared secret is static, this again means the other party is misbehaving.

We wish to avoid the situation where a third party can send well-formed RADIUS packets to a RADIUS proxy that cause a (D)TLS session to close. Therefore, in other situations, the session SOULD remain open in the face of non-conforming packets. Any malformed RADIUS packets sent by a third party will go through the security checks of the RADIUS proxy upon reception and will not be forwarded. Well-formed RADIUS packets with portions that the proxy does not understand do not pose a security risk to the security properties of the RADIUS/(D)TLS session and can be forwarded. This ensures forward compatibility with future RADIUS extensions.

7.5. Migrating from RADIUS/UDP to RADIUS/(D)TLS

Since RADIUS/UDP security relies on the MD5 algorithm, which is considered insecure, using RADIUS/UDP over insecure networks is risky. We therefore recommend to migrate from RADIUS/UDP to RADIUS/(D)TLS. Within this migration process, however, there are a few items that need to be considered by administrators.

Firstly, administrators may be tempted to simply migrate from RADIUS/UDP to RADIUS/(D)TLS with (D)TLS-PSK and reuse the RADIUS shared secret as (D)TLS-PSK. While this may seem like an easy way to upgrade RADIUS/UDP to RADIUS/(D)TLS, the cryptographic problems with the RADIUS/UDP shared secret render the shared secret potentially compromised. Using a potentially compromised shared secret as TLS-PSK compromises the whole TLS connection. Therefore, any shared

secret used with RADIUS/UDP before **MUST NOT** be used with RADIUS/(D)TLS and (D)TLS-PSK. Implementers **MUST NOT** reuse the configuration option for the RADIUS/UDP shared secret for the (D)TLS-PSK to prevent accidental reuse.

When upgrading from RADIUS/UDP to RADIUS/(D)TLS, there may be a period of time, where the connection between client and server is configured for both transport profiles. If the old RADIUS/UDP configuration is left configured, but not used in normal operation, e.g. due to a fail-over configuration that prefers RADIUS/(D)TLS, an attacker could disrupt the RADIUS/(D)TLS communication and force a downgrade to RADIUS/UDP. To prevent this it is **RECOMMENDED** that, when the migration to RADIUS/(D)TLS is completed, the RADIUS/UDP configuration is removed. RADIUS/(D)TLS clients **MUST NOT** fall back to RADIUS/UDP if the RADIUS/(D)TLS communication fails, unless explicitly configured this way.

7.6. 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. With (D)TLS enabled, these problems are expected to get worse.

We therefore recommend in these situations to use a local proxy that 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 benefit of this configuration is that there is one place in the client that arbitrates all RADIUS traffic. Subsystems that do not implement RADIUS/(D)TLS can remain unaware of (D)TLS. (D)TLS sessions opened by the proxy can remain open for a long period of time, even when client subsystems are restarted. The proxy can do RADIUS/UDP to some servers and RADIUS/(D)TLS to others.

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

8. Design Decisions

Many of the design decisions of RADIUS/TLS and RADIUS/DTLS can be found in [RFC6614] and [RFC7360]. This section will discuss the

rationale behind significant changes from the experimental specification.

8.1. Mandatory-to-implement transports

With the merging of RADIUS/TLS and RADIUS/DTLS the question of mandatory-to-implement transports arose. In order to avoid incompatibilities, there were two possibilities: Either mandate one of the transports for all implementations or mandate the implementation of both transports for either the server or the client. As of the time writing, RADIUS/TLS is widely adapted for some use cases (see Appendix A). However, TLS has some serious drawbacks when used for RADIUS transport. Especially the sequential nature of the connection and the connected issues like Head-of-Line blocking could create problems.

Therefore, the decision was made that RADIUS servers must implement both transports. For RADIUS clients, that may run on more constrained nodes, the implementations can choose to implement only the transport, that is better suited for their needs.

8.2. Mandatory-to-implement trust profiles

[RFC6614] mandates the implementation of the trust profile "certificate with PKIX trust model" for both clients and servers. The experience of the deployment of RADIUS/TLS as specified in [RFC6614] has shown that most actors still rely on RADIUS/UDP. Since dealing with certificates can create a lot of issues, both for implementers and administrators, for the re-specification we wanted to create an alternative to insecure RADIUS transports like RADIUS/UDP that can be deployed easily without much additional administrative overhead.

As with the supported transports, the assumption is that RADIUS servers are generally believed to be less constrained that RADIUS clients. Since some client implementations already support using certificates for mutual authentication and there are several use cases, where Pre-shared keys are not usable (e.g. a dynamic federation with changing members), the decision was made that, analog to the supported transports, RADIUS servers must implement both certificates with PKIX trust model and TLS-PSK as means of mutual authentication. RADIUS clients again can choose which method is better suited for them, but must, for compatibility reasons, implement at least one of the two.

8.3. Changes in application of TLS

The original specification of RADIUS/TLS does not forbid the usage of compression in the TLS layer. As per [RFC9325], Section 3.3, compression should not be used due to the possibility of

compression-related attacks, unless the application protocol is proven to be not open to such attacks. Since some attributes of the RADIUS packets within the TLS tunnel contain values that an attacker could at least partially choose (i.e. username, MAC address or EAP message), there is a possibility for compression-related attacks, that could potentially reveal data in other RADIUS attributes through length of the TLS record. To circumvent this attack, this specification forbids the usage of TLS compression.

9. IANA Considerations

Upon approval, IANA should update the Reference to radsec in the Service Name and Transport Protocol Port Number Registry:

*Service Name: radsec

*Port Number: 2083

*Transport Protocol: tcp/udp

*Description: Secure RADIUS Service

*Assignment notes: The TCP port 2083 was already previously assigned by IANA for "RadSec", an early implementation of RADIUS/TLS, prior to issuance of the experimental RFC 6614. [This document] updates RFC 6614 (RADIUS/TLS) and RFC 7360 (RADIUS/DTLS), while maintaining backward compatibility, if configured. For further details see RFC 6614, Appendix A or [This document] Appendix C.

10. References

10.1. Normative References

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

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

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

[RFC3539] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, DOI

10.17487/RFC3539, June 2003, <https://www.rfc-editor.org/rfc/rfc3539>.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, DOI 10.17487/RFC3579, September 2003, <https://www.rfc-editor.org/rfc/rfc3579>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/rfc/rfc4033>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/rfc/rfc5077>.

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/rfc/rfc5246>.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <https://www.rfc-editor.org/rfc/rfc5247>.

[RFC5248] Hansen, T. and J. Klensin, "A Registry for SMTP Enhanced Mail System Status Codes", BCP 138, RFC 5248, DOI 10.17487/RFC5248, June 2008, <https://www.rfc-editor.org/rfc/rfc5248>.

[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, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

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

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/rfc/rfc6066>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/rfc/rfc6347>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <https://www.rfc-editor.org/rfc/rfc6520>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/rfc/rfc7250>.

[RFC7585] Winter, S. and M. McCauley, "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)", RFC 7585, DOI 10.17487/RFC7585, October 2015, <https://www.rfc-editor.org/rfc/rfc7585>.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/rfc/rfc9293>.

[RFC9325]
Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/rfc/rfc9325>.

10.2. Informative References

[I-D.ietf-radext-radiusv11]
DeKok, A., "RADIUS ALPN and removing MD5", Work in Progress, Internet-Draft, draft-ietf-radext-radiusv11-04, 26 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-radext-radiusv11-04>.

[I-D.ietf-radext-tls-psk]
DeKok, A., "RADIUS and TLS-PSK", Work in Progress, Internet-Draft, draft-ietf-radext-tls-psk-09, 29 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-radext-tls-psk-09>.

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

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

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

[RFC7593] Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam Architecture for Network Roaming", RFC 7593, DOI 10.17487/RFC7593, September 2015, <https://www.rfc-editor.org/rfc/rfc7593>.

Appendix A. Lessons learned from deployments of the Experimental [RFC6614]

There are at least two major (world-scale) deployments of [RFC6614]. This section will discuss lessens learned from these deployments, that influenced this document.

A.1. eduroam

eduroam is a globally operating Wi-Fi roaming consortium exclusively for persons in Research and Education. For an extensive background on eduroam and its authentication fabric architecture, refer to [RFC7593].

Over time, more than a dozen out of 100+ national branches of eduroam used RADIUS/TLS in production to secure their country-to-country RADIUS proxy connections. This number is big enough to attest that the protocol does work, and scales. The number is also low enough to wonder why RADIUS/UDP continued to be used by a majority of country deployments despite its significant security issues.

Operational experience reveals that the main reason is related to the choice of PKIX certificates for securing the proxy interconnections. Compared to shared secrets, certificates are more complex to handle in multiple dimensions:

- Lifetime: PKIX certificates have an expiry date, and need administrator attention and expertise for their renewal

- Validation: The validation of a certificate (both client and server) requires contacting a third party to verify the revocation status. This either takes time during session setup (OCSP checks) or requires the presence of a fresh CRL on the server - this in turn requires regular update of that CRL.

- Issuance: PKIX certificates carry properties in the Subject and extensions that need to be vetted. Depending on the CA policy, a certificate request may need significant human intervention to be verified. In particular, the authorisation of a requester to operate a server for a particular NAI realm needs to be verified. This rules out public "browser-trusted" CAs; eduroam is operating a special-purpose CA for eduroam RADIUS/TLS purposes.

- Automatic failure over time: CRL refresh and certificate renewal must be attended to regularly. Failure to do so leads to failure of the authentication service. Among other reasons, employee churn with incorrectly transferred or forgotten responsibilities is a risk factor.

It appears that these complexities often outweigh the argument of improved security; and a fallback to RADIUS/UDP is seen as the more appealing option.

It can be considered an important result of the experiment in [RFC6614] that providing less complex ways of operating RADIUS/TLS are required. The more thoroughly specified provisions in the

current document towards TLS-PSK and raw public keys are a response to this insight.

On the other hand, using RADIUS/TLS in combination with Dynamic Discovery as per [RFC7585] necessitates the use of PKIX certificates. So, the continued ability to operate with PKIX certificates is also important and cannot be discontinued without sacrificing vital functionality of large roaming consortia.

A.2. Wireless Broadband Alliance's OpenRoaming

OpenRoaming is a globally operating Wi-Fi roaming consortium for the general public, operated by the Wireless Broadband Alliance (WBA). With its (optional) settled usage of hotspots, the consortium requires both RADIUS authentication as well as RADIUS accounting.

The consortium operational procedures were defined in the late 2010s when [RFC6614] and [RFC7585] were long available. The consortium decided to fully base itself on these two RFCs.

In this architecture, using PSKs or raw public keys is not an option. The complexities around PKIX certificates as discussed in the previous section are believed to be controllable: the consortium operates its own special-purpose CA and can rely on a reliable source of truth for operator authorisation (becoming an operator requires a paid membership in WBA); expiry and revocation topics can be expected to be dealt with as high-priority because of the monetary implications in case of infrastructure failure during settled operation.

A.3. Participating in more than one roaming consortium

It is possible for a RADIUS/TLS (home) server to participate in more than one roaming consortium, i.e. to authenticate its users to multiple clients from distinct consortia, which present client certificates from their respective consortium's CA; and which expect the server to present a certificate from the matching CA.

The eduroam consortium has chosen to cooperate with (the settlement-free parts of) OpenRoaming to allow eduroam users to log in to (settlement-free) OpenRoaming hotspots.

eduroam RADIUS/TLS servers thus may be contacted by OpenRoaming clients expecting an OpenRoaming server certificate, and by eduroam clients expecting an eduroam server certificate.

It is therefore necessary to decide on the certificate to present during TLS session establishment. To make that decision, the availability of Trusted CA Indication in the client TLS message is important.

It can be considered an important result of the experiment in [RFC6614] that Trusted CA Indication is an important asset for inter-connectivity of multiple roaming consortia.

Appendix B. Interoperable Implementations

Appendix C. Backward compatibility

TODO describe necessary steps to configure common servers for compatibility with this version. Hopefully the differences to [RFC6614] are small enough that almost no config change is necessary.

Acknowledgments

Thanks to the original authors of RFC 6613, RFC 6614 and RFC 7360: Alan DeKok, Stefan Winter, Mike McCauley, Stig Venaas and Klaas Vierenga.

Thanks to Arran Curdbard-Bell for text around keepalives and the Status-Server watchdog algorithm.

Thanks to Alan DeKok for his constant review of this document over its whole process.

Authors' Addresses

Jan-Frederik Rieckers
Deutsches Forschungsnetz | German National Research and Education Network
Alexanderplatz 1
10178 Berlin
Germany

Email: rieckers@dfn.de
URI: www.dfn.de

Stefan Winter
Fondation Restena | Restena Foundation
2, avenue de l'Université
L-4365 Esch-sur-Alzette
Luxembourg

Email: stefan.winter@restena.lu
URI: www.restena.lu