RADIUS EXTensions
Internet-Draft
Obsoletes: 6614, 7360 (if approved)
Intended status: Standards Track
Expires: 3 April 2026

J.-F. Rieckers
DFN
S. Winter
RESTENA

30 September 2025

(Datagram) Transport Layer Security ((D)TLS) Encryption for RADIUS
draft-ietf-radext-radiusdtls-bis-09

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

Table of Contents

1. Introduction

The RADIUS protocol is a widely deployed authentication, authorization and accounting solution. It is defined in [RFC2865], [RFC2866], [RFC5176] and others. The deployment experience has shown several shortcomings, such 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 [RFC9765]

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)

The following list contains the most important changes from the previous specifications in [RFC6613] (RADIUS/TCP), [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, and therfore removes [RFC6613] as normative reference.

* 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" ([RFC6614], Section 2.3). This document changes this. For servers, TLS-X.509-PKIX (Section 4.2.1, equivalent to "certificates with PKIX" in RFC6614) and TLS-PSK (Section 4.2.4) is now mandated and clients must implement at least 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.

* [RFC6613] only included limited text around retransmissions, this document now gives more guidance on how to handle retransmissions, especially across different transports.

* The rules for verifying the peer certificate have been updated to follow guidance provided in [RFC9525]. Using the Common Name RDN for validation of server certificates is now forbidden.

* The response to unwanted packets has changed. Nodes should now reply with a Protocol-Error packet, which is connection-specific and should not be proxied.

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

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 of this document or [RFC9765] 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 4.5.

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

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.

RADIUS/(D)TLS implementations MUST 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.

RADIUS/(D)TLS clients MUST 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. RADIUS/(D)TLS servers MUST be able to answer to Status-Server requests. 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]. This is 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

As defined in Section 3.2, RADIUS/(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 [BCP195], especially in regards to recommended cipher suites and TLS session resumption. 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

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, which will be further specified in this section:

* TLS-X.509-PKIX
* TLS-X.509-FINGERPRINT
* TLS-RAW-PUBLIC-KEY
* TLS-PSK

Independent of the chosen mode of authentication, the mutual authentication MUST be performed during the initial handshake. Alternative methods, such as post-handshake certificate-based client authentication (see [RFC8446], Section 4.6.2) with TLS 1.3 or renegotiation with TLS 1.2, MUST NOT be used to achieve mutual authentication.

4.2.1. Authentication using X.509 certificates with PKIX trust model (TLS-X.509-PKIX)

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 trust anchor (i.e. a list of trusted Certificate Authorities (CAs)[RFC5280]) for new TLS sessions. This list SHOULD be application specific and not use a global system trust store.
* Certificate validation MUST include the verification rules as per [RFC5280].
* Implementations SHOULD indicate their trust anchors when opening or accepting TLS sessions. See [RFC5246], Section 7.4.4 and [RFC6066], Section 6 for TLS 1.2 and [RFC8446], Section 4.2.4 for TLS 1.3.

* When the configured trust base changes (e.g., removal of a CA from the trust anchor; issuance of a new CRL for a given CA), implementations SHOULD reassess all connected peer's continued validity of the certificate path. This can either be done by caching the peer's certificate for the duration of the connection and re-evaluating the cached certificate or by renegotiating the (D)TLS connection, either directly or by opening a new (D)TLS connection and closing the old one.

* Implementations SHOULD NOT keep a connection open for longer than the validity span of the peer certificate. At the time the peer certificate expires, the connection SHOULD be closed and re-opened.

RADIUS/(D)TLS peers SHOULD NOT be pre-configured with a list of trusted CAs by the vendor or manufacturer that are enabled by default. Instead, the peers SHOULD start off with an empty CA list as trust anchor. The addition of a CA SHOULD be done only when manually configured by the administrator. This does not preclude vendors or manufacturers including their trust list in their products, but the enabling of those lists should be a conscious decision by an administrator.

RADIUS/(D)TLS clients and servers MUST follow [RFC9525] when validating peer identities. Specific details are provided below:

* Certificates MAY use wildcards in the identifiers of DNS names and realm names, but only as the complete, left-most label.

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

  - If the expected RADIUS/(D)TLS server is associated with a specific NAI realm, e.g. by dynamic discovery [RFC7585] or static configuration, that realm is matched against the presented identifiers of any subjectAltName entry of type otherName whose name form is NAIRealm as defined in [RFC7585], Section 2.2.

  - If the expected RADIUS/(D)TLS server was configured as a hostname, or the hostname was yielded by a dynamic discovery procedure, that name is matched against the presented identifiers of any subjectAltName entry of type dNSName [RFC5280]. Since a dynamic discovery might by itself not be secured, implementations MAY require the use of DNSSEC [RFC4033] to ensure the authenticity of the DNS result before considering this identity as valid.

- If the expected RADIUS/(D)TLS server was configured as an IP address, the configured IP address is matched against the presented identifier in any subjectAltName entry of type iPAddress [RFC5280].

- The Common Name RDN MUST NOT be used to identify a server.

- Clients MAY use other attributes of the certificate to validate the servers identity, but it MUST NOT accept any certificate without validation.

- Clients which also act as servers (i.e. proxies) may be susceptible to security issues when a ClientHello is mirrored back to themselves. More details on this issue are discussed in Section 7.

* 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 identifiers of any subjectAltName entry of type dNSName [RFC5280].

  - For clients configured by their source IP address, the configured IP address is matched against the presented identifiers of any subjectAltName entry of type iPAddress [RFC5280]. For clients configured by IP range, the certificate MUST be valid for the IP address the client is currently using.

  - Implementations MAY consider additional subjectAltName extensions to identify a client.

  - If configured by the administrator, the identity check MAY be omitted after a successful [RFC5280] trust chain check, e.g. if the client used dynamic lookup there is no configured client identity to verify. The clients authorization MUST then be validated using a certificate policy OID unless both peers are part of a trusted network.

* Implementations MAY allow 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 presented in subjectAltName entries of type uniformResourceIdentifier [RFC5280] or a set of allowed X.509v3 Certificate Policies).

4.2.2. Authentication using X.509 certificate fingerprints (TLS-X.509-FINGERPRINT)

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 (TLS-RAW-PUBLIC-KEYS)

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

4.2.4. Authentication using TLS-PSK (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 TLS-X.509-PKIX trust model.

Further guidance on the usage of TLS-PSK in RADIUS/(D)TLS is given in [RFC9813].

4.3. Connecting Client Identity

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 trust model used, the RADIUS/(D)TLS client identity can be determined differently.

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

With TLS-RAW-PUBLIC-KEY, a client is uniquely identified by the Raw public key.

With TLS-X.509-FINGERPRINT, a client is uniquely identified by the fingerprint of the presented client certificate.

With TLS-X.509-PKIX, a client is uniquely identified by the tuple of the serial number of the presented client certificate and the issuer.

In practice, identification of unique clients is not always necessary and could be based on the subject of the presented certificate or a subjectAltName entry. While this identification technique could match multiple distinct certificates and therefore distinct clients, it is often sufficient, e.g. for the purpose of applying policies.

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 addresses or ranges. One example of this can be to restrict clients configured by DNS name to only the IP address(es) that this DNS name resolves to.

A client connecting from outside the allowed 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. TLS Session Resumption

Session resumption lowers the time and effort required to start a (D)TLS session and increases network responsiveness. This is especially helpful when using short idle timeouts.

RADIUS/(D)TLS clients and server SHOULD implement session resumption. Implementations supporting session resumption MUST cache data during the initial full handshake, sufficient to allow authorization descisions to be made during resumption. For RADIUS/(D)TLS servers, this should preferably be done using stateless session resumption as specified in [RFC5077], to reduce the resource usage for cached sessions.

When resuming a (D)TLS session, both client and server MUST re-authorize the connection by using the original, cached data. In particular, this includes the X.509 certificate (when using a PKIX trust model) as well as any policies associated with that identity such as restrictions on source IP address. The re-authorization MUST give the same result as if a full handshake was performed at the time of resumption.

If cached data cannot be retrieved securely, resumption MUST NOT be done, by either immediately closing the connection or reverting to a full handshake. If a resumed session is closed immediately after being established, the RADIUS/(D)TLS client MUST NOT re-attempt session resumption but perform a full TLS handshake instead.

4.5. RADIUS Datagrams

The RADIUS/(D)TLS specification does not change the client/server architecture of RADIUS. RADIUS/(D)TLS clients transmit the same packet types on the connection they initiated as a RADIUS/UDP client would, and RADIUS/(D)TLS servers transmit the same packet types on the connections the server has accepted as a RADIUS/UDP server would. As noted in Section 3.2, RADIUS/(D)TLS uses the same port for Authentication and Accounting packets. As non-exhaustive example, a RADIUS/(D)TLS client can transmit packets of type Access-Request, Accounting-Request, Status-Server, Disconnect-ACK over the same connection, and a RADIUS/(D)TLS server can transmit packets of type Access-Accept, Access-Reject, Access-Challenge, Accounting-Response,

Disconnect-Request.

However, special considerations apply for mixing Authentication and Accounting packets over the same connection. Traditional RADIUS/UDP uses different ports for Authentication and Accounting, where RADIUS/(D)TLS uses the same connection for all RADIUS packets. Due to the use of one single port for all packet types, it is required that a RADIUS/(D)TLS server has a means to signal which types of packets are supported on the server to a connecting peer.

Since the number of outstanding RADIUS packets is limited, it is important to reply to packets of a packet type which the RADIUS/(D)TLS server does not process or, in a proxy setup, does not forward. Otherwise, these outstanding packets would impact the performance of the connection. The reply, however, must clearly indicate that the server did not process this packet to prevent the client from falsely assuming the server processed the packet.

For every unwanted packet, a RADIUS/(D)TLS server SHOULD respond with a Protocol-Error packet as defined in [RFC7930], Section 4. The Error-Cause attribute of this packet SHOULD be set to the value 406 ("Unsupported Extension"), if the server does not support the packet type, or the value 502 ("Request Not Routable (Proxy)"), if the request cannot be routed. Future specifications may recommend other Error-Cause attribute values for specific scenarios.

The RADIUS/(D)TLS client SHOULD NOT assume that the configured server is not able to handle all packets of the packet type based on the Protocol-Error response. In proxy scenarios, a RADIUS proxy may be unable to forward accounting packets for one realm, but able to forward them for another.

The previous specification of RADIUS/TLS in [RFC6614] recommended to send a different reply. For unwanted CoA-Requests or Disconnect-Requests, the servers should respond with a CoA-NAK or Disconnect-NAK, respectively. For unwanted Accounting-Requests, the servers should respond with an Accounting-Response containing an Error-Cause attribute with the value 406 ("Unsupported Extension"). It was also recommended that a RADIUS/TLS client observing this Accounting-Response should stop sending Accounting-Request packets to this server. This behavior, however, could lead to problems, especially in proxy fabrics, since the RADIUS client cannot determine whether the reply came from the correct server or a RADIUS proxy along the way. Other than the other responses (CoA-NAK, Disconnect-NAK and Accounting-Response), the Protocol-Error packet is explicitly only applicable to one RADIUS hop and must not be forwarded, which gives the RADIUS client the opportunity to re-route the unwanted packet to a different RADIUS server. This also is backwards compatible with existing implementations, since RADIUS clients must ignore any incoming RADIUS packets with an unknown packet type.

Since proxying of RADIUS packets is a general issue in RADIUS and not specific to RADIUS/(D)TLS, the details of handling the Protocol-Error reply on the client side are outside of the scope of this document.

4.6. Forwarding RADIUS packets between UDP and TCP based transports

When a RADIUS proxy forwards packets, it is possible that the incoming and outgoing links have substantially different properties. This issue is most notable in UDP to TCP proxying, but there are still possible issues even when the same transport is used on both incoming and outgoing links. [RFC2866], Section 1.2 noted this issue many years ago:

A forwarding server may either perform its forwarding function in a pass through manner, where it sends retransmissions on as soon as it gets them, or it may take responsibility for retransmissions, for example in cases where the network link between forwarding and remote server has very different characteristics than the link between NAS and forwarding server.

These differences are most notable in throughput, and in differing retransmission requirements.

4.6.1. Throughput Differences lead to Network Collapse

An incoming link to the proxy may have substantially different throughput than the outgoing link. Perhaps the network characteristics on the two links are different, or perhaps the home server is slow. In both situations, the proxy may be left with a difficult choice about what to do with the incoming packets, if the rate of incoming packets exceeds throughput on the outgoing link.

As RADIUS does not provide for connection-based congestion control, there is no way for the proxy to signal on the incoming link that the client should slow its rate of sending packets. As a result, the proxy must simply accept the packets, buffer them, and hope that they can be be sent outbound before the client gives up on the request.

4.6.2. Differing Retransmission Requirements

Due to the lossy nature of UDP, RADIUS/UDP and RADIUS/DTLS transports are required to perform retransmissions as per [RFC5080], Section 2.2.1. In contrast, RADIUS/TCP and RADIUS/TLS transports are reliable, and do not perform retransmissions. These requirements lead to an issue for proxies when they send packets across protocol boundaries with differing retransmission behaviors.

When a proxy receives packets on an unreliable transport, and forwards them across a reliable transport, it receives retransmissions from the client, but MUST NOT forward those retransmissions across the reliable transport. The proxy MAY log information about these retransmissions, but it does not perform any other action.

When a proxy receives packets on a reliable transport, and forwards them across an unreliable transport, the proxy MUST perform retransmissions across the unreliable transport as per [RFC5080], Section 2.2.1. That is, the proxy takes responsibility for the retransmissions. Implementations MUST take care to not completely decouple the two transports in this situation.

That is, if an incoming connection on a reliable transport is closed, there may be pending retransmissions on an outgoing unreliable transport. Those retransmissions MUST be stopped, as there is nowhere to send the reply. Similarly, if the proxy sees that the client has given up on a request (such as by re-using an Identifier before the proxy has sent a response), the proxy MUST stop all retransmissions of the old request and discard it.

The above requirements are a logical extension of the common practice where a client stops retransmission of a packet once it decides to "give up" on the packet and discard it. Whether this discard process is due to internal client decisions, or interaction with incoming connections is irrelevant. When the client cannot do anything with responses to a request, it MUST stop retransmitting that request.

4.6.3. Acct-Delay-Time and Event-Timestamp

In order to avoid congestive collapse, it is RECOMMENDED that RADIUS/(D)TLS clients which originate Accounting-Request packets (i.e. not proxies) do not include Acct-Delay-Time ([RFC2866], Section 5.2) in those packets. Instead, those clients SHOULD include Event-Timestamp ([RFC2869], Section 5.3), which is the time at which the original event occured. The Event-Timestamp MUST NOT be updated on any retransmissions, as that would both negate the meaning of Event-Timestamp, and create the same problem as with Acct-Delay-Time.

Not using Acct-Delay-Time allows for RADIUS packets to be retransmitted without change. In contrast, updating Acct-Delay-Time would require that the client create and send a new packet without signalling the server that the previous packet is no longer considered active. This process can occur repeatedly, which leads to multiple different packets containing effectively the same information (except for Acct-Delay-Time). This duplication contributes to congestive collapse of the network, if a RADIUS proxy performs retransmission to the next hop for each of those packets independently.

Additionally, the different properties of the RADIUS/TLS transport as well as cross-protocol proxying change the assumption of a negligible transmission time of the RADIUS packet, on which the value of Acct-Delay-Time is based. While a single UDP packet may have a negligible transmission time, application data sent via TLS could arrive at the sender with a significant delay due to the underlying TCP retransmission mechanism. If the packet is proxied from RADIUS/TLS to RADIUS/DTLS or RADIUS/UDP, the proxy has to retransmit on its own without changing the value of Acct-Delay-Time, which again introduces non-negligible transmission delays.

Using Event-Timestamp instead of Acct-Delay-Time also removes an ambiguity around retransmitted packets for RADIUS/TLS. Since there is no change to the packet contents when a retransmission timer expires, no new packet ID is allocated, and therefore no new packet is created.

Where RADIUS/(D)TLS clients do include Acct-Delay-Time in RADIUS packets, the client SHOULD use timers to detect packet loss, as described in Section 4.7.2. RADIUS/(D)TLS clients SHOULD NOT update the Acct-Delay-Time, and therefore create a new RADIUS packet with the same information, until the timer has determined that the original packet has in fact been completely lost. This ensures that there is no congestive collapse, since a new packet is only created if following hops have also given up on retransmission, while keeping the functionality of Acct-Delay-Time to determine how long ago the event occured. It only reduces the granularity of Acct-Delay-Time to the retransmission timeout, compared to the different approach of updating the Acct-Delay-Time on each retransmission.

4.7. Client Timers

RADIUS/(D)TLS clients may need to reconnect to a server that rejected their connection attempt and retransmit RADIUS packets which did not get an answer.

4.7.1. Reconnection attempts

In contrast to RADIUS/UDP, RADIUS/(D)TLS establishes a (D)TLS session before transmitting any RADIUS packets. Therefore, in addition to retransmission of RADIUS packets, RADIUS/(D)TLS clients also have to deal with connection retries.

RADIUS/(D)TLS clients MUST NOT immediately reconnect to a RADIUS/(D)TLS server after a failed connection attempt and MUST have a lower bound for the time between retries. The lower bound SHOULD be configurable. As only exception, a RADIUS/(D)TLS client MAY reconnect immediately iff the client attempted to resume a TLS session and the server closed the connection. In this case the new connection attempt MUST NOT use TLS session resumption.

It is RECOMMENDED that RADIUS/(D)TLS clients implement an algorithm for handling the timing of such reconnection attempts. Implementations MAY choose to use an algorithm similar to the retransmission algorithm defined in [RFC5080], Section 2.2.1. The algorithm used SHOULD include a configurable lower and upper bound for the time between retries, an (exponential) backoff, a configurable timeout after which the client gives up reconnecting and MAY add a jitter.

Where the connection to a RADIUS/(D)TLS server is established only when there is a RADIUS packet to be sent, adding a second RADIUS packet to be send SHOULD NOT trigger an immediate reconnection attempt. Instead, the algorithm SHOULD continue as it would have without the new packet, but the client MAY reset the timeout for giving up reconnecting.

Where the connection to a RADIUS/(D)TLS server is configured to be static and always kept open, the reconnect algorithm SHOULD have an upper limit for the time between retries (e.g. 60 seconds) and not give up trying to reconnect.

4.7.2. RADIUS packet retransmission

RADIUS/(D)TLS clients MUST implement retransmission timers for retransmitting RADIUS packets such as the ones defined in [RFC5080], Section 2.2.1. Other algorithms than the one defined in [RFC5080] are possible, but any timer implementations MUST have similar properties of including jitter, exponential backoff and a maximum retransmission count (MRC) or maximum retransmission duration (MRD).

As TLS is a reliable transport, RADIUS/TLS clients can only retransmit a packet if a connection closes without that packet receiving a reply, therefore the timers MUST NOT result in retransmission of any packet. Instead, the timers, MRC or MRD specifically, can be used to determine that a packet will most likely not receive an answer ever, for example because a packet loss has occured in a later RADIUS hop or the home server ignores the RADIUS packet.

See Section 5.2 for more discussion on retransmission behavior.

4.8. Session limits and timeout

While RADIUS/UDP could be implemented mostly stateless (except for the requests in flight), both TCP/TLS as well as DTLS require state tracking of the underlying TLS connection and are thus subject to potential resource exhaustion. This is aggravated by the fact that RADIUS client/servers are often statically configured and thus form long-running peer relationships with long-running connections.

Implementations SHOULD have configurable limits on the number of open connections. 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.

The close notification of (D)TLS or underlying connections are not fully reliable, or they might be unnecessarily kept alive by heartbeat or watchdog traffic, occupying resources. Therefore, both RADIUS/(D)TLS clients and servers MAY close connections after they have been idle for some time (no traffic except application layer watchdog). This idle timeout SHOULD be configurable within reasonable limits and SHOULD allow to disable idle timeout completely.

On the server side, this mostly helps avoid resource exhaustion. For clients, proactively closing sessions can also help mitigate situations where watchdog mechanisms are unavailable or fail to detect non-functional connections. Some scenarios or RADIUS protocol extensions could also require that a connection be kept open at all times, so clients MAY immediately re-open the connection. These scenarios could be related to monitoring the infrastructure or to allow the server to proactively send packets to the clients without a preceding request.

The value of the idle timeout to use depends on the exact deployment and is a trade-of between resource usage on clients/servers and the overhead of opening new connections. Very short timeouts that are at or below the timeouts used for application layer watchdogs, typically in the range of 30-60s can be considered unreasonable. In contrast, the upper limit is much more difficult to define but may be in the range of 10 to 15min, depending on the available resources, or never (disabling idle timeout) in scenarios where a permanently open connection is required.

4.9. Behavior on session closure of incoming sessions

If an incoming (D)TLS session or the underlying connection is closed or broken, then there is no way to send a RADIUS response message to the client. The RADIUS/(D)TLS server behavior then depends on the types of packets being processed, and on the role of the server.

A RADIUS/(D)TLS server acting as proxy MUST discard all requests associated with the closed connection. As no response can be sent over the now-closed (D)TLS connection, any further processing of requests is pointless. A discarded request may have a cached RADIUS response packet ([RFC5080], Section 2.2.2), in which case the cached response also MUST be discarded. If there is no cached response packet, then the request might still be processed by the home server. The RADIUS proxy MUST discard any response to these requests and SHOULD stop processing the requests.

A home server which receives Access-Request packets MUST behave as defined above for a proxy and discard those requests and stop processing them. Where a RADIUS packet is part of a multi-packet authentication session (e.g. EAP), the underlying authentication session could be continued, or the underlying authentication session data could be discarded. The server may be able to receive and process another packet for that session via a different incoming connection. It is difficult to make more recommendations for managing partially processed authentication sessions, as such recommendations depend strongly on the authentication method being used. As a result, further behavior is implementation defined and outside the scope of this specification.

A home server which receives other kinds of packets (for example Accounting-Request, CoA-Request, Disconnect-Request) MAY finish processing outstanding requests, and then discard any response. This behavior ensures that the desired action is still taken, even if the home server cannot inform the client of the result of that action.

4.10. Malformed Packets and Unknown clients

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, since a malformed RADIUS packet could impact the decoding of succeeding packets.

With DTLS, the "next" packet does not depend on proper decoding of the previous packet, since the RADIUS packets are sent in independent DTLS records. However, since both TLS and DTLS provide integrity protection and ensure that the packet was sent by the peer, a protocol violation at this stage implies that the peer is misbehaving.

Implementations of this specification SHOULD treat 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, in the case of RADIUS/TLS, 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 situations where the previous specifications allow a packet to be "silently discarded" upon receipt, but in which a connection MAY remain open:

* Packet with an invalid code field (see Section 4.5 for details)

* Response packets that do not match any outstanding request

* A server lacking the resources to process a request

For request packets that would have been silently discarded in the previous specifications, servers SHOULD reply with a Protocol-Error [RFC7930], Section 4 message to avoid request ID exhaustion, and servers SHOULD include an Error-Cause attribute indicating the type of failure. In any case, further processing of the original request MUST stop.

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

5. RADIUS/TLS specific specifications

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

5.1. Sending and receiving RADIUS traffic

The TLS layer of RADIUS/TLS provides a stream-based communication between the two peers instead of the traditional packet-based communication as with RADIUS/UDP. As a result, the way RADIUS packets are sent and received has to change.

Instead of relying on packet borders of the underlying transport protocol to indicate the start of a new packet, the RADIUS/TLS peers have to keep track of the packet borders by examining the header of the received RADIUS packets.

After the TLS session is established, a RADIUS/TLS peer MUST NOT send any data except for RADIUS packets over the connection. Since the RADIUS packet header contains a Length field, the end of the RADIUS packet can be deduced. The next RADIUS packet MUST be sent directly after the RADIUS packet before, that is, the peers MUST NOT add padding before, between, or after RADIUS packets.

When receiving RADIUS packets, a RADIUS/TLS node MUST determine the borders of RADIUS packet based on the Length field in the RADIUS header. Note that, due to the stream architecture of TLS, it is possible that a RADIUS packet is first recieved only partially, and the remainder of the packet is contained in following fragments. Therefore, RADIUS/TLS peers MUST NOT assume that the packet length is invalid solely based on the currenlty available bytes in the stream.

As an implementation note, it is RECOMMENDED that RADIUS/TLS implementations do not pass a single RADIUS packet to the TLS library in multiple fragments and instead assemble the RADIUS packet and pass it as one unit, in order to avoid unnecessary overhead when sending or receiving (especially if every new write generates a new TLS record) and wait times on the other peer.

5.2. Duplicates and Retransmissions

As TCP is a reliable transport, RADIUS/TLS peers MUST NOT retransmit RADIUS packets over a given TCP connection. 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 session, the ID of the packet MAY change. If the ID changes, any security attributes such as Message-Authenticator MUST be recalculated.

Despite the above discussion, RADIUS/TLS 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 address and 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.

5.3. TCP Applications Are Not UDP Applications

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

Additionally, differences in the transport like Head of Line (HoL) blocking and the possibility of increased transmission times 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. Additionally, due to the architecture of TCP as reliable stream transport, TCP retransmissions can occur significantly later, even multiple seconds, after the original data was passed to the network stack by the application. In contrast, RADIUS/UDP packets are usually received either quickly, or not at all, in which case the RADIUS/UDP stack triggers a retransmission of the packet on the application layer.

6. RADIUS/DTLS specific specifications

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

6.1. RADIUS packet handling

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 UDP packet to be larger than the Path MTU (PMTU), which causes the packet to be fragmented. Implemententors and operators should be aware of the possibility of fragmented UDP packets.

RADIUS/DTLS nodes MUST send exactly one RADIUS packet per DTLS record. This ensures that the RADIUS packets do not get fragmented at a point where a re-ordering of UDP packets would result in decoding failures. The DTLS specification mandates that a DTLS record must not span multiple UDP packets. We note that a single UDP datagram may, however, contain multiple DTLS records. RADIUS/DTLS nodes MAY use this behavior to send multiple RADIUS packets in one UDP packet.

For the receiving RADIUS/DTLS node, the length checks defined in [RFC2865], Section 3 still apply. That is, a receiving RADIUS/DTLS node MUST perform all the length checks, but MUST use the length of the decrypted payload of the DTLS record instead of the UDP packet length. Exactly one RADIUS packet is encapsulated in a DTLS record, and any data outside the range of the RADIUS length field within the decrypted payload of a single DTLS record MUST be treated as padding, as it would be with a RADIUS/UDP packet, and be ignored. For DTLS messages containing multiple DTLS records, each DTLS record MUST be parsed individually.

If a RADIUS packet should be re-transmitted, either as retransmission due to a missing response by the client or as retransmission of a cached response by the server, the RADIUS/DTLS peers MUST re-process the RADIUS packet through DTLS. That is, for the purpose of retransmissions, RADIUS/DTLS peers cache the RADIUS packet, as a RADIUS/UDP peer would, and not the DTLS record that contains the RADIUS packet.

6.2. Server behavior

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 from 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

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.

6.4. Session Management

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. When implementations do not use the 5-tuple described below, note that IP address based policies MUST still be applied for all incoming packets, similar to the mandated behavior for TLS Session Resumption in Section 4.4.

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.

6.4.1. Server Session Management

A RADIUS/DTLS server MUST track ongoing DTLS sessions for each client, based on the following 5-tuple:

* source IP address
* source port
* destination IP address
* destination port
* protocol (fixed to UDP)

Note that this 5-tuple is independent of IP address version (IPv4 or IPv6).

Each 5-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.

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

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 5-tuple and entry) MUST be deleted when the DTLS session is closed for any reason. 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.

Since UDP is stateless, the potential exists for the client to initiate a new DTLS session using a particular 5-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 5-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 5-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 5-tuple. Non-compliant, or unexpected packets will be ignored by the DTLS layer.

6.4.2. Client Session Management

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 MAY use PMTU discovery [RFC6520] to determine the PMTU between the client and server, prior to sending any RADIUS traffic. While a RADIUS client has limited to no possibilities to reduce the size of an outgoing RADIUS packet without unwanted side effects, it gives the RADIUS client the possibility to determine whether or not the RADIUS packet can even be sent over the connection. IP fragmentation may not be functioning, so by determining the PMTU, the RADIUS client can preemptively select a different RADIUS server to send the RADIUS packet to. Further discussion of this problem is deemed outside of the scope of this document.

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.

One possible way to reduce the attack surface is to reduce the number of proxies in the overall proxy chain. For this, dynamic discovery as defined in [RFC7585] can be used.

7.1.1. Loopback-Attack on Peers acting as Server and Client

RADIUS/(D)TLS nodes that are configured to act both as client and server, typically in a proxy configuration, may be vulnerable to attacks where an attacker mirrors back all traffic to the node. Therefore, nodes that are capable of acting as both client and server SHOULD implement mitigations to avoid accepting connections from itself. One example of a potentially vulnerable configuration is a setup where the RADIUS/(D)TLS server is accepting incoming

connections from any address (or a wide address range). Since the server may not be able to verify the certificate subject or subject alternate names, the trust is based on the certificate issuer or certificate OID. However, in this case, the client certificate which the RADIUS/(D)TLS node uses for outgoing connections on the client side might also satisfy the trust check of the server side. Other scenarios where the identification of an outgoing connection satisfies the trust check of an incoming one are possible, but are not enumerated here.

Either through misconfiguration, erroneous or spoofed dynamic discovery, or an attacker rerouting TLS packets, a proxy might thus open a connection to itself, creating a loop. Such attacks have been described for TLS-PSK [RFC9257], dubbed a selfie-attack, but are much broader in the RADIUS/(D)TLS case. In particular, as described above, they also apply to certificate based authentication.

Implementations SHOULD therefore detect connections from itself, and reject them. There is currently no detection method that works universally for all use-cases and TLS implementations. Some possible detection methods are listed below:

* Comparing client or server random used in the TLS handshake. While this is a very effective method, it requires access to values which are normally private to the TLS implementation.

* Sending a custom random number in an extension in the TLS client hello. Again, this is very effective, but requires extension of the TLS implementation.

* Comparing the incoming server certificate to all server certificates configured on the proxy. While in some scenarios this can be a valid detection method, using the same server certificate on multiple servers would keep these servers from connecting with each other, even when this connection is legitimate.

The application layer RADIUS protocol also offers some loop detection, e.g. using a Proxy-State attribute. However, these methods are not capable of reliably detecting and suppressing these attacks in every case and are outside the scope of this document.

7.2. Usage of null encryption cipher suites for debugging

For debugging purposes, some TLS implementations 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 party eavesdropping on the conversation. Following the recommendations in [RFC9325], Section 4.1, this specification forbids the usage of NULL encryption cipher suites for RADIUS/(D)TLS.

For cases where administrators need access to the decrypted RADIUS/(D)TLS traffic, we suggest using different approaches, like exporting the key material from TLS libraries according to [I-D.ietf-tls-keylogfile].

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 bogus RADIUS packet would fail the cryptographic checks and the server would silently discard the bogus 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 bogus 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 SHOULD have a configurable limit of the number of sessions they can track. 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/(D)TLS servers MUST limit the number of partially open (D)TLS sessions and SHOULD expose this limit as configurable option to the administrator.

To prevent resource exhaustion by partially opening a large number of (D)TLS sessions, RADIUS/(D)TLS servers SHOULD have a timeout on partially open (D)TLS sessions. We recommend a limit of a few seconds, implementations SHOULD expose this timeout as configurable option to the administrator. If a (D)TLS session is not established within this timeframe, it is likely that this is a bogus 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 accumulated permitted IP address ranges, individually for each transport.

7.4. TLS Session Lifetime and Key Rotation

The underlying TLS sessions of RADIUS/(D)TLS connections may have a long lifetime. Especially when dealing with high volume of RADIUS traffic, the encryption keys have to be rotated regularly, depending on both the amout of data which was tansferred, and on the encryption method. See [RFC8446], Section 5.5 and [I-D.irtf-cfrg-aead-limits] for more information.

Implementers SHOULD be aware of this issue and determine whether the underlying TLS library automatically rotates encryption keys or not. If the underlying TLS library does not perform the rotation automatically, RADIUS/(D)TLS implementations SHOULD perform this rotation manually, either by a key update of the existing TLS connection or by closing the TLS connection and opening a new one.

7.5. 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 potentially a 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 SHOULD 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.

7.6. 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.

Special considerations apply for clients behind a NAT, where some clients use RADIUS/UDP and others use RADIUS/(D)TLS. A RADIUS server might not be able to detect if a RADIUS/(D)TLS client falls back to RADIUS/UDP, they will appear with the same source IP address to the server and use the same shared secret. It is therefore RECOMMENDED to not use RADIUS/UDP and RADIUS/(D)TLS clients behind a NAT at the same time.

7.7. 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, implementers 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 than 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/(D)TLS servers must implement both certificates with PKIX trust model and TLS-PSK as means of mutual authentication. RADIUS/(D)TLS 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. [RFCXXXX] updates RFC 6614 (RADIUS/TLS) and RFC 7360 (RADIUS/DTLS).
* Reference: [RFCXXXX] (this document)

10. References

10.1. Normative References

[BCP195] Best Current Practice 195, <https://www.rfc-editor.org/info/bcp195>. At the time of writing, this BCP comprises the following:

Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/info/rfc8996>.

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/info/rfc9325>.

[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>.

[RFC7930] Hartman, S., "Larger Packets for RADIUS over TCP", RFC 7930, DOI 10.17487/RFC7930, August 2016, <https://www.rfc-editor.org/rfc/rfc7930>.

[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>.

[RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <https://www.rfc-editor.org/rfc/rfc9525>.

10.2. Informative References

[I-D.ietf-tls-keylogfile]
Thomson, M., Rosomakho, Y., and H. Tschofenig, "The SSLKEYLOGFILE Format for TLS", Work in Progress, Internet-Draft, draft-ietf-tls-keylogfile-05, 9 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-keylogfile-05>.

[I-D.irtf-cfrg-aead-limits]
G端nther, F., Thomson, M., and C. A. Wood, "Usage Limits on AEAD Algorithms", Work in Progress, Internet-Draft, draft-

irtf-cfrg-aead-limits-10, 8 April 2025,
<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-aead-limits-10>.

[RFC2869] Rigney, C., Willats, W., and P. Calhoun, "RADIUS Extensions", RFC 2869, DOI 10.17487/RFC2869, June 2000, <https://www.rfc-editor.org/rfc/rfc2869>.

[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>.

[RFC9257] Housley, R., Hoyland, J., Sethi, M., and C. A. Wood, "Guidance for External Pre-Shared Key (PSK) Usage in TLS", RFC 9257, DOI 10.17487/RFC9257, July 2022, <https://www.rfc-editor.org/rfc/rfc9257>.

[RFC9765] DeKok, A., "RADIUS/1.1: Leveraging Application-Layer Protocol Negotiation (ALPN) to Remove MD5", RFC 9765, DOI 10.17487/RFC9765, April 2025, <https://www.rfc-editor.org/rfc/rfc9765>.

[RFC9813] DeKok, A., "Operational Considerations for Using TLS Pre-Shared Keys (TLS-PSKs) with RADIUS", BCP 243, RFC 9813, DOI 10.17487/RFC9813, July 2025, <https://www.rfc-editor.org/rfc/rfc9813>.

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

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 and his many text contributions, like text around forwarding issues between TCP and UDP based transports.

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'Universit迪
L-4365 Esch-sur-Alzette
Luxembourg
Email: stefan.winter@restena.lu
URI: www.restena.lu