Network Working Group
INTERNET-DRAFT
Category: Experimental
<draft-ietf-radext-dtls-13.txt>
Expires: January 4, 2015
3 July 2014

Alan DeKok
FreeRADIUS

DTLS as a Transport Layer for RADIUS
draft-ietf-radext-dtls-13

Abstract

The RADIUS protocol defined in RFC 2865 has limited support for authentication and encryption of RADIUS packets. The protocol transports data in the clear, although some parts of the packets can have obfuscated content. Packets may be replayed verbatim by an attacker, and client-server authentication is based on fixed shared secrets. This document specifies how the Datagram Transport Layer Security (DTLS) protocol may be used as a fix for these problems. It also describes how implementations of this proposal can co-exist with current RADIUS systems.

Table of Contents

1. Introduction

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

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

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

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

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

1.1. Terminology

This document uses the following terms:

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

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

implement RADIUS/DTLS.

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

RADIUS/UDP
RADIUS over UDP, as defined in [RFC2865].

RADIUS/TLS
RADIUS over TLS, as defined in [RFC6614].

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

1.2. Requirements Language

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

1.3. Document Status

This document is an Experimental RFC.

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

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

Another experimental characteristic of this specification is the question of key management between RADIUS/DTLS peers. RADIUS/UDP only allowed for manual key management, i.e., distribution of a shared secret between a client and a server. RADIUS/DTLS allows manual distribution of long-term proofs of peer identity, by using TLS-PSK ciphersuites. RADIUS/DTLS also allows the use of X.509 certificates in a PKIX infrastructure. It remains to be seen if one of these methods will prevail or if both will find their place in real-life deployments. The authors can imagine pre-shared keys (PSK) to be popular in small-scale deployments (Small Office, Home Office (SOHO) or isolated enterprise deployments) where scalability is not an issue and the deployment of a Certification Authority (CA) is considered too much of a hassle; however, the authors can also imagine large roaming consortia to make use of PKIX. Readers of this specification are encouraged to read the discussion of key management issues within [RFC6421] as well as [RFC4107].

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

2. Building on Existing Foundations

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

2.1. Changes to RADIUS

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

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

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

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

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

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

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

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

All other aspects of RADIUS are unchanged.

2.2. Similarities with RADIUS/TLS

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

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

* We replace references to "TCP" with "UDP"

* We replace references to "RADIUS/TLS" with "RADIUS/DTLS"

* We replace references to "TLS" with "DTLS"

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

2.2.1. Changes from RADIUS/TLS to RADIUS/DTLS

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

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

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

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

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

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

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

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

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

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

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

Section 6 applies to RADIUS/DTLS.

3. Interaction with RADIUS/UDP

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

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

This section describes how clients and servers should use

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

3.1. DTLS Port and Packet Types

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

3.2. Server Behavior

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

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

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

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

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

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

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

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

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

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

4. Client Behavior

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

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

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

- o TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- o TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

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

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

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

5. Session Management

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

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

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

5.1. Server Session Management

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

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

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

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

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

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

DTLS Data
: An implementation-specific variable which may contain information

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

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

5.1.1. Session Opening and Closing

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.2. Client Session Management

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

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

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

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

DTLS sessions MUST also be deleted when a RADIUS packet fails validation due to a packet being malformed, or when it has an invalid Message-Authenticator, or invalid Response Authenticator. There are other cases when the specifications require that a packet received via a DTLS session be "silently discarded". In those cases, implementations MAY delete the underlying DTLS session.

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

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

6. Implementation Guidelines

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

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

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

6.1. Client Implementations

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

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

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

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

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

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

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

6.2. Server Implementations

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

7. Diameter Considerations

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

8. IANA Considerations

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

- o Assignee: change "Mike McCauley" to "IESG".

- o Contact: change "Mike McCauley" to "IETF Chair"

- o Reference: Add this document as a reference

- o Assignment Notes: add the text "The UDP port 2083 was already previously assigned by IANA for "RadSec", an early implementation of RADIUS/TLS, prior to issuance of this RFC."

9. Implementation Status

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

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

9.1. Radsecproxy

Organization: Radsecproxy

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

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

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

Licensing: Freely distributable with acknowledgement

Implementation experience: No comments from implementors.

9.2. jradius

Organization: Coova

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

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

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

Licensing: Freely distributable with requirement to redistribute source.

Implementation experience: No comments from implementors.

10. Security Considerations

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

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

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

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

DoS issues which result from tracking additional information.

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

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

10.1. Crypto-Agility

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

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

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

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

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

10.2. Legacy RADIUS Security

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

The existence of fast and cheap attacks on MD5 could result in a loss of all network security which depends on RADIUS. Attackers could obtain user passwords, and possibly gain complete network access. We cannot overstate the disastrous consequences of a successful attack on RADIUS.

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

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

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

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

10.3. Resource Exhaustion

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

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

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

10.4. Client-Server Authentication with DTLS

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

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

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

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

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

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

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

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

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

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

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

10.5. Network Address Translation

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

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

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

10.6. Wildcard Clients

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

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

10.7. Session Closing

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

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

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

10.8. Client Subsystems

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

As noted in Section 6.1, above, clients SHOULD use a local proxy which arbitrates all RADIUS traffic between the client and all servers. This proxy will encapsulate all knowledge about servers, including security policies, fail-over, and load-balancing. All client subsystems SHOULD communicate with this local proxy, ideally over a loopback address. The requirements on using strong shared secrets still apply.

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

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

11. References

11.1. Normative references

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

[RFC3539]
Aboba, B. et al., "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, June 2003.

[RFC5077]
Salowey, J, et al., "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008

[RFC5080]
Nelson, D. and DeKok, A, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.

[RFC5246]
Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5997]
DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", RFC 5997, August 2010.

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

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

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

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

11.2. Informative references

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

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

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

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

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

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

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

[MD5Attack]
Dobbertin, H., "The Status of MD5 After a Recent Attack", CryptoBytes Vol.2 No.2, Summer 1996.

[MD5Break]
Wang, Xiaoyun and Yu, Hongbo, "How to Break MD5 and Other Hash Functions", EUROCRYPT. ISBN 3-540-25910-4, 2005.

Acknowledgments

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

Authors' Addresses

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

Email: aland@freeradius.org

RADIUS Extensions Working Group S. Winter
Internet-Draft RESTENA
Intended status: Experimental M. McCauley
Expires: November 1, 2015 AirSpayce
April 30, 2015

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

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

```
                                    +-------+
                                    |       |
                                    |   .   |
                                    |       |
                                    +---+---+
                                      / | \
                    +----------------/  |  \---------------------+
                    |                   |                        |
                    |                   |                        |
                    |                   |                        |
                 +--+---+            +--+--+                +----+---+
                 |      |            |     |                |        |
                 | .edu |    . . .   | .nl |      . . .     | .ac.uk |
                 |      |            |     |                |        |
                 +--+---+            +--+--+                +----+---+
                  / | \                 | \                      |
                 /  |  \                |  \                     |
                /   |   \               |   \                    |
         +-----+    |    +-----+        |    +------+            |
         |          |          |        |           |            |
         |          |          |        |           |            |
     +---+---+ +----+---+ +----+---+ +--+---+ +-----+----+ +-----+-----+
     |       | |        | |        | |      | |          | |           |
     |utk.edu| |utah.edu| |case.edu| |hva.nl| |surfnet.nl| |soton.ac.uk|
     |       | |        | |        | |      | |          | |           |
     +----+--+ +--------+ +--------+ +------+ +----+-----+ +-----------+
          |                                        |
          |                                        |
       +--+--+                                  +--+--+
       |     |                                  |     |
     +-+-----+-+                                |     |
     |         |                                +-----+
     +---------+
     user: paul@surfnet.nl             surfnet.nl Authentication server
```

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

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

```
                                    ---------
                                   /         \
                          ---------           ------------
                         /                                \
                         |    DNS                          -
               ----------|                                  \
              /          \          surfnet.nl NAPTR?       |
        (1)  /            ----       -> radius.surfnet.nl   /
            /                 \                            /
           /                   --------           ---------
          /                            \---------/
         |
         |   ---------------------------------------
         |  /              (2) RADIUS               \
         |  |                                       |
     +---+---+ +----+---+ +----+---+ +--+---+ +-----+----+ +-----+-----+
     |       | |        | |        | |      | |          | |           |
     |utk.edu| |utah.edu| |case.edu| |hva.nl| |surfnet.nl| |soton.ac.uk|
     |       | |        | |        | |      | |          | |           |
     +----+--+ +--------+ +--------+ +------+ +----+-----+ +-----------+
          |                                        |
          |                                        |
       +--+--+                                  +--+--+
       |     |                                  |     |
     +-+-----+-+                                |     |
     |         |                                +-----+
     +---------+
     user: paul@surfnet.nl             surfnet.nl Authentication server
```

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

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

1.1. Requirements Language

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

1.2. Terminology

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

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

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

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

1.3. Document Status

This document is an Experimental RFC.

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

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

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

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

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

- provisions for determining the authority of a server to act for users of a realm (declared out of scope for Diameter)

- much more in-depth guidance on DNS regarding timeouts, failure conditions, alteration of Time-To-Live (TTL) information than the Diameter counterpart

- a partially correct routing error detection during DNS lookups

2. Definitions

2.1. DNS Resource Record (RR) definition

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

2.1.1. S-NAPTR

2.1.1.1. Registration of Application Service and Protocol Tags

This specification defines three S-NAPTR service tags:

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

Figure 3: List of Service Tags

This specification defines two S-NAPTR protocol tags:

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

Figure 4: List of Protocol Tags

Note well:

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

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

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

2.1.1.2. Definition of Conditions for Retry/Failure

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

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

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

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

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

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

2.1.1.3. Server Identification and Handshake

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

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

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

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

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

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

Step 2 is to compare the value of algorithm’s variable "R" after the execution of step 3 of the discovery algorithm in Section 3.4.3 below (i.e. after a consortium name mangling, but before conversion to a form usable by the name resolution library) to all values of the contacted RADIUS/TLS server's X.509 certificate property "subjectAlternativeName:otherName:NAIRealm" as defined in Section 2.2.

2.1.1.3.2. Other mechanism: Trust Roots + policyOID

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

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

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

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

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

2.1.1.3.3. Other mechanism: DNSSEC / DANE

Where DNSSEC is used, the results of the algorithm can be trusted; i.e. the entity which executes the algorithm can be certain that the realm that triggered the discovery is actually served by the server that was discovered via DNS. However, this does not guarantee that the server is also authorized (i.e. a recognised member of the roaming consortium). The server still needs to present an X.509 certificate proving its authority to serve a particular realm.

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

Example:

Realm = "example.com"

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

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

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

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

2.1.1.3.4. Client Authentication and Authorisation

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

2.1.2. SRV

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

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

Figure 5: List of SRV Labels

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

2.1.3. Optional name mangling

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

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

Examples:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ub-naiRealm-length INTEGER ::= 255

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

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

The comparison of an NAIRealm to the NAI realm as derived from user input with this algorithm is a byte-by-byte comparison, except for the optional leftmost dot-separated part of the value whose content is a single "*" character; such labels match all strings in the same dot-separated part of the NAI realm. If at least one of the sAN:otherName:NAIRealm values matches the NAI realm, the server is considered authorized; if none matches, the server is considered unauthorized.

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

Examples:

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

Figure 6: Examples for NAI realm vs. certificate matching

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

3. DNS-based NAPTR/SRV Peer Discovery

3.1. Applicability

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

a. new user sessions, i.e. for the initial Access-Request. Subsequent messages concerning this session, for example Access-Challenges and Access-Accepts use the previously-established communication channel between client and server.

b. the first accounting ticket for a user session.

c. the first RADIUS DynAuth packet for a user session.

3.2. Configuration Variables

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

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

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

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

3.3. Terms

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

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

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

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

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

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

3.4. Realm to RADIUS server resolution algorithm

3.4.1. Input

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

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

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

Usage of separators other than @.

Encoding of User-Name in local encodings.

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

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

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

3.4.2. Output

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

3.4.3. Algorithm

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

1. Determine P = (position of last "@" character) in I.

2. generate R = (substring from P+1 to end of I)

3. modify R according to agreed consortium procedures if applicable

4. convert R to a representation usable by the name resolution library if needed

5. Initialize TIMER = 0; start TIMER. If TIMER reaches DNS_TIMEOUT, continue at step 20.

6. Using the host’s name resolution library, perform a NAPTR query for R (see "Delay considerations" below). If the result is a negative DNS response, O-2 = Effective TTL (TTL value of the SOA record) and continue at step 13. If name resolution returns with error, O-1 = { empty set }, O-2 = BACKOFF_TIME and terminate.

7. Extract NAPTR records with service tag "aaa+auth", "aaa+acct", "aaa+dynauth" as appropriate. Keep note of the protocol tag and remaining TTL of each of the discovered NAPTR records.

8. If no records found, continue at step 13.

9. For the extracted NAPTRs, perform successive resolution as defined in [RFC3958], section 2.2. An implementation MAY use greedy result evaluation according to the NAPTR order/preference fields (i.e. can execute the subsequent steps of this algorithm for the highest-order entry in the set of results, and only lookup the remainder of the set if necessary).

10. If the set of hostnames is empty, O-1 = { empty set }, O-2 = BACKOFF_TIME and terminate.

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

12. Proceed with step 18.

13. Generate R’ = (prefix R with "_radiustls._tcp." and/or "_radiustls._udp.")

14. Using the host’s name resolution library, perform SRV lookup with R’ as label (see "Delay considerations" below).

15. If name resolution returns with error, O-1 = { empty set }, O-2 = BACKOFF_TIME and terminate.

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

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

18. Generate O-1 by resolving hostnames in O’ into corresponding A and/or AAAA addresses: O-1 = (set of {IP address; port; protocol; order/preference; Effective TTL (all DNS TTLs that led to this result) } for all hostnames), O-2 = 0.

19. For each element in O-1, test if the original request which triggered dynamic discovery was received on {IP address; port}. If yes, O-1 = { empty set }, O-2 = BACKOFF_TIME, log error, Terminate (see next section for a rationale). If no, O is the result of dynamic discovery. Terminate.

20. O-1 = { empty set }, O-2 = BACKOFF_TIME, log error, Terminate.

3.4.4. Validity of results

The dynamic discovery algorithm is used by servers which do not have sufficient configuration information to process an incoming request on their own. If the discovery algorithm result contains the server’s own listening address (IP address and port), then there is a potential for an endless forwarding loop. If the listening address is the DNS result with the highest priorty, the server will enter a tight loop (the server would forward the request to itself, triggering dynamic discovery again in a perpetual loop). If the address has a lower priority in the set of results, there is a potential loop with intermediate hops in between (the server could forward to another host with a higher priority, which might use DNS itself and forward the packet back to the first server). The underlying reason that enables these loops is that the server executing the discovery algorithm is seriously misconfigured in that it does not recognise the request as one that is to be processed by itself. RADIUS has no built-in loop detection, so any such loops would remain undetected. So, if step 18 of the algorithm discovers such a possible-loop situation, the algorithm should be aborted and an error logged. Note that this safeguard does not provide perfect protection against routing loops. One reason which might introduce a loop include the possiblity that a subsequent hop has a statically configured next-hop which leads to an earlier host in the loop. Another reason for occuring loops is if the algorithm was executed with greedy result evaluation, and the own address was in a lower-priority branch of the result set which was not retrieved from DNS at all, and thus can’t be detected.

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

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

3.4.5. Delay considerations

The host’s name resolution library may need to contact outside entities to perform the name resolution (e.g. authoritative name servers for a domain), and since the NAI discovery algorithm is based on uncontrollable user input, the destination of the lookups is out of control of the server that performs NAI discovery. If such outside entities are misconfigured or unreachable, the algorithm above may need an unacceptably long time to terminate. Many RADIUS implementations time out after five seconds of delay between Request and Response. It is not useful to wait until the host name resolution library signals a timeout of its name resolution algorithms. The algorithm therefore controls execution time with TIMER. Execution of the NAI discovery algorithm SHOULD be non-blocking (i.e. allow other requests to be processed in parallel to the execution of the algorithm).

3.4.6. Example

Assume

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

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

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

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

May the configuration variables be

DNS_TIMEOUT = 3 seconds

MIN_EFF_TTL = 60 seconds

BACKOFF_TIME = 3600 seconds

If DNS contains the following records:

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

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

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

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

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

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

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

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

1. P = 7

2. R = "tu-m[U+00FC]nchen.example"

3. NOOP

4. name resolution library converts R to xn--tu-mnchen-t9a.example

5. TIMER starts.

6. Result:

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

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

7. Result:

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

8. NOOP

9. Successive resolution performs SRV query for label _myradius._tcp.xn--tu-mnchen-t9a.example, which results in

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

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

10. NOOP

11. O’ = {

    (radsec.xn--tu-mnchen-t9a.example.; 2083; RADIUS/TLS; 10; 60),

    (backup.xn--tu-mnchen-t9a.example.; 2083; RADIUS/TLS; 20; 60)

    } // minimum TTL is 47, up’ed to MIN_EFF_TTL

12. Continuing at 18.

13. (not executed)

14. (not executed)

15. (not executed)

16. (not executed)

17. (not executed)

18. O-1 = {

    (2001:0DB8::202:44ff:fe0a:f704; 2083; RADIUS/TLS; 10; 60),

    (192.0.2.7; 2083; RADIUS/TLS; 20; 60)

    }; O-2 = 0

19. No match with own listening address; terminate with tuple (O-1, O-2) from previous step.

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

4. Operations and Manageability Considerations

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

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

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

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

5. Security Considerations

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

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

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

With Dynamic Discovery being enabled for a RADIUS Server, and depending on the deployment scenario, the server may need to open up its target IP address and port for the entire internet, because arbitrary clients may discover it as a target for their authentication requests. If such clients are not part of the roaming consortium, the RADIUS/TLS connection setup phase will fail (which is intended) but the computational cost for the connection attempt is significant. With the port for a TLS-based service open, the RADIUS server shares all the typical attack vectors for services based on TLS (such as HTTPS, SMTPS, ...). Deployments of RADIUS/TLS with Dynamic Discovery should consider these attack vectors and take appropriate counter-measures (e.g. blacklisting known-bad IPs on a firewall, rate-limiting new connection attempts, etc.).

6. Privacy Considerations

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

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

- o All intermediate IP hops can inspect most of the packet payload in clear text, including the User-Name and Calling-Station-Id attributes, and can observe which client sent the packet to which clearinghouse. This allows the creation of mobility profiles for any passive observer on the IP path.

- o The existence of a central clearinghouse creates an opportunity for the clearinghouse to trivially create the same mobility profiles. The clearinghouse may or may not be trusted not to do this, e.g. by sufficiently threatening contractual obligations.

- o In addition to that, with the clearinghouse being a RADIUS intermediate in possession of a valid shared secret, the clearinghouse can observe and record even the security-critical RADIUS attributes such as User-Password. This risk may be mitigated by choosing authentication payloads which are cryptographically secured and do not use the attribute User-Password - such as certain EAP types.

- o There is no additional information disclosure to parties outside the IP path between the RADIUS client and server (in particular, no DNS servers learn about realms of current ongoing authentications).

With RADIUS and dynamic discovery,

- This protocol allows for RADIUS clients to identify and directly connect to the RADIUS home server. This can eliminate the use of clearinghouses to do forwarding of requests, and it also eliminates the ability of the clearinghouse to then aggregate the user information that flows through it. However, there exist reasons why clearinghouses might still be used. One reason to keep a clearinghouse is to act as a gateway for multiple backends in a company; another reason may be a requirement to sanitise RADIUS datagrams (filter attributes, tag requests with new attributes, ...).

- Even where intermediate proxies continue to be used for reasons unrelated to dynamic discovery, the number of such intermediates may be reduced by removing those proxies which are only deployed for pure request routing reasons. This reduces the number of entities which can inspect the RADIUS traffic.

- RADIUS clients which make use of dynamic discovery will need to query the Domain Name System, and use a user’s realm name as the query label. A passive observer on the IP path between the RADIUS client and the DNS server(s) being queried can learn that a user of that specific realm was trying to authenticate at that RADIUS client at a certain point in time. This may or may not be sufficient for the passive observer to create a mobility profile. During the recursive DNS resolution, a fair number of DNS servers and the IP hops in between those get to learn that information. Not every single authentication triggers DNS lookups, so there is no one-to-one relation of leaked realm information and the number of authentications for that realm.

- Since dynamic discovery operates on a RADIUS hop-by-hop basis, there is no guarantee that the RADIUS payload is not transmitted between RADIUS systems which do not make use of this algorithm, and possibly using other transports such as RADIUS/UDP. On such hops, the enhanced privacy is jeopardized.

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

7. IANA Considerations

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

- S-NAPTR Application Service Tags registry

   * aaa+auth

   * aaa+acct

   * aaa+dynauth

- S-NAPTR Application Protocol Tags registry

   * radius.tls.tcp

   * radius.dtls.udp

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

Service Name: radiustls; radiusdtls

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

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

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

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

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

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

IANA is requested to assign the following identifiers:

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

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

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

8. References

8.1. Normative References

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

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

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

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

[RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5580] Tschofenig, H., Adrangi, F., Jones, M., Lior, A., and B. Aboba, "Carrying Location Objects in RADIUS and Diameter", RFC 5580, August 2009.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

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

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

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

8.2. Informative References

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

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

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

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

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

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

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

 DEFINITIONS EXPLICIT TAGS ::=

 BEGIN

 -- EXPORTS ALL --

 IMPORTS

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

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


 -- Service Name Object Identifier

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

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

 -- Service Name

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

 ub-naiRealm-length INTEGER ::= 255

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

 END
```

Authors' Addresses

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

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

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

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

RADIUS Extensions Working Group   S. Winter
Internet-Draft   RESTENA
Intended status: Experimental   M. McCauley
Expires: August 17, 2012   OSC
                                                               S. Venaas
                                                             K. Wierenga
                                                                   Cisco
                                                       February 14, 2012

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

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

1.1. Requirements Language

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

1.2. Terminology

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

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

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

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

1.3. Document Status

This document is an Experimental RFC.

It is one out of several approaches to address known cryptographic weaknesses of the RADIUS protocol (see also Section 4). The specification does not fulfill all recommendations on a AAA transport profile as per [RFC3539]; in particular, by being based on TCP as a transport layer, it does not prevent head-of-line blocking issues.

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

Another experimental characteristic of this specification is the question of key management between RADIUS/TLS peers. RADIUS/UDP only allowed for manual key management, i.e. distribution of a shared secret between a client and a server. RADIUS/TLS allows manual distribution of long-term proofs of peer identity as well (by using TLS-PSK cipher suites, or identifying clients by a certificate fingerprint), but as a new feature enables use of X.509 certificates in a PKIX infrastructure. It remains to be seen if one of these methods prevail, or if both will find their place in real-life deployments. The authors can imagine pre-shared keys to be popular in small-scale deployments (SOHO or isolated enterprise deployments) where scalability is not an issue and the deployment of a CA is considered too much a hassle; but can also imagine large roaming consortia to make use of PKIX. Readers of this specification are encouraged to read the discussion of key management issus within [RFC6421] as well as [RFC4107].

It has yet to be decided whether this approach is to be chosen for standards track. One key aspect to judge whether the approach is usable at large scale is by observing the uptake, usability and operational behaviour of the protocol in large-scale, real-life deployments.

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

2. Normative: Transport Layer Security for RADIUS/TCP

2.1. TCP port and packet types

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

2.2. TLS negotiation

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

2.3. Connection Setup

RADIUS/TLS nodes

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

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

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

   * Support for certificate-based mutual authentication is REQUIRED.

   * Negotiation of mutual authentication is REQUIRED.

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

Section 6 for details.

* Support for and negotiation of compression is OPTIONAL.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.4. Connecting Client Identity

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

RADIUS/TLS supports multiple operation modes.

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

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

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

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

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

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

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

- Originating IP address
- TLS Identifier

2.5. RADIUS Datagrams

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

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

- Access-Request
- Accounting-Request

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

and they receive

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

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

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

and they receive

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

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

Section 3.4 for a discussion of signaling.

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

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

3. Informative: Design Decisions

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

3.1. Implications of Dynamic Peer Discovery

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

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

3.2. X.509 Certificate Considerations

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

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

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

3.3. Ciphersuites and Compression Negotiation Considerations

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

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

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

3.4. RADIUS Datagram Considerations

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

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

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

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

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

4. Compatibility with other RADIUS transports

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

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

If a given IP device is able to receive RADIUS payloads on multiple transports, this may or may not be the same instance of software, and it may or may not serve the same purposes. It is not safe to assume that both ports are interchangeable. In particular, it can not be assumed that state is maintained for the packet payloads between the transports. Two such instances MUST be considered separate RADIUS server entities.

5. Diameter Compatibility

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

6. Security Considerations

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

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

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

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

- o Client side: de-configure the failover setup, leaving RADIUS/TLS as the only communication option

- o Server side: de-configure the RADIUS/UDP client from the list of valid RADIUS clients

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

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

7. IANA Considerations

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

- o Reference: list the RFC number of this document as the reference

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

8. Notes to the RFC Editor

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

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

9. Acknowledgements

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

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

10. References

10.1. Normative References

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

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

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

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

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

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

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

10.2. Informative References

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

[I-D.ietf-radext-dynamic-discovery] Winter, S. and M. McCauley, "NAI-based Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS", draft-ietf-radext-dynamic-discovery-03 (work in progress), July 2011.

[RFC3539] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, June 2003.

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

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

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

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

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

[MD5-attacks] Black, J., Cochran, M., and T. Highland, "A Study of the MD5 Attacks: Insights and Improvements", October 2006, <http://www.springerlink.com/content/40867l85727r7084/>.

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

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

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

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

Appendix A. Implementation Overview: Radiator

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

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

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

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

<AuthBy RADSEC>

* Protocol tcp

* UseTLS

* TLS_CertificateFile

* Secret radsec

<ServerRADSEC>

* Protocol tcp

* UseTLS

* TLS_RequireClientCert

* Secret radsec

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

Appendix B. Implementation Overview: radsecproxy

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

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

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

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

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

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

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

Appendix C. Assessment of Crypto-Agility Requirements

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

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

(M) negotiation of cryptographic algorithms for integrity and auth

(M) negotiation of cryptographic algorithms for encryption

(M) replay protection

(M) define mandatory-to-implement cryptographic algorithms

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

(R) support for Perfect Forward Secrecy in session keys

(R) support X.509 certificate based operation

(R) support Pre-Shared keys

(R) support for confidentiality of the entire packet

(M/R) support Automated Key Management

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

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

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

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

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

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

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

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

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

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

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

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

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

Authors' Addresses

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

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

Mike McCauley
Open Systems Consultants
9 Bulbul Place
Currumbin Waters QLD 4223
AUSTRALIA

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

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

EMail: stig@cisco.com

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

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