Network Working Group Alan DeKok
INTERNET-DRAFT FreeRADIUS
Category: Experimental
<draft-ietf-radext-dtls-06.txt>
Expires: January 12, 2014
12 July 2013

DTLS as a Transport Layer for RADIUS
draft-ietf-radext-dtls-06

Abstract

The RADIUS protocol [RFC2865] 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 a RADIUS 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 is not 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. This continuity ensures that existing network-layer infrastructure (firewall rules, etc.) does not need to be changed when RADIUS clients and servers are upgraded to support RADIUS over DTLS.

This specification does not, however, solve all of the problems associated with RADIUS. 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.

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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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 encapsulated RADIUS packets of 4096 in length, with a corresponding increase in the maximum size of the encapsulated DTLS packets.

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.

(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 identies can be determined from TLS 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 item (3) applies to RADIUS/DTLS when the new port is used. When DTLS is used over the existing RADIUS/UDP ports, the relationship between RADIUS packet codes and UDP ports in RADIUS/DTLS is unchanged from RADIUS.

Section 3.4 item (4) applies to RADIUS/DTLS when the new port is used. When DTLS is used over the existing RADIUS/UDP ports, the use of negative ICMP responses is unchanged from RADIUS.

Section 3.4 item (5) applies to RADIUS/DTLS when the new port is used. When DTLS is used over the existing RADIUS/UDP ports, the use of negative ICMP responses is unchanged from RADIUS.

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

2.2.2. Reinforcement of RADIUS/TLS

We re-iterate that much of [RFC6614] applies to this document. Specifically, Section 4 and Section 6 of that document are applicable 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. That approach 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, long-term use of RADIUS/UDP is NOT RECOMMENDED.

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 Section 2.2.1 describes issues surrounding the use of one port for multiple packet types, by referencing [RFC6614] Section 3.4.

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.

As RADIUS has no provisions for capability signalling, there is no way for a RADIUS 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.

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".

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.

RADIUS/DTLS clients SHOULD NOT probe servers to see if they support DTLS transport. Doing so could cause problems for servers which do not implement DTLS. Instead, clients SHOULD use DTLS as a transport layer only when administratively configured.

RADIUS clients often had multiple independent RADIUS implementations, or processes that originate packets. This practice was simple to implement, but means that each independent subsystem must independently discover network issues or server failures. It is therefore RECOMMENDED that clients 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. Connection Management

Where [RFC6614] can rely on the TCP state machine to perform connection tracking, this specification cannot. As a result, implementations of this specification may need to perform connection 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 connection 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 Connection Management

A RADIUS/DTLS server MUST track ongoing DTLS client connections based on a key composed of the following 4-tuple:

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

Note that this key is independent of IP address version (IPv4 or IPv6).

Each entry associated with a key contains the following information:

DTLS Data
An implementation-specific variable containing information about the active DTLS connection.

Last Taffic
A variable containing a timestamp which indicates when this connection last received valid traffic.

Each entry may contain other information, such as idle timeouts, connection lifetimes, and other implementation-specific data.

5.1.1. Session Management

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. The requirement to accept RADIUS/UDP and RADIUS/DTLS on the same port makes this recommendation difficult to implement in practice. Server implementation SHOULD therefore have a way of tracking partially setup DTLS connections. Servers SHOULD limit both the number and impact on resources of partial connections.

Sessions (both key and entry) MUST 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 failed security, 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.

The above paragraph can be rephrased more generically. 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 connection. 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.

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 connection 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 timestamp SHOULD be updated on reception of a valid RADIUS/DTLS packet, or a DTLS heartbeat. 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 sessions they are tracking. They SHOULD stop the creating of new sessions when a large number are already being tracked. This "maximum sessions" number SHOULD be exposed to administrators as a configurable setting.

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 it has received secure notification from the client that the session is closed. Or, when the server has decided for itself that the session is closed. Taking any other action would permit unauthenticated clients to perform a DoS attack, by closing active 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 Connection Management

Clients SHOULD use Path MTU (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. Alternatively, RADIUS/DTLS clients may 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.

A client may choose to avoid DTLS heartbeats and watchdog packets entirely. However, DTLS provides no signal that a 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 RADIUS/DTLS clients implement DTLS heartbeats and/or watchdog packets for all DTLS sessions.

DTLS sessions MUST also be deleted when a RADIUS packet fails validation due to a packet being malformed, or when it has an invalid Message-Authenticator, or invalid Response Authenticator. 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 hex strings.

It is RECOMMENDED that keys be derived from a cryptographically secure pseudo-random number generator (CSPRNG). 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. It is a good idea to leverage existing functionality.

RADIUS/DTLS clients SHOULD use one source 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 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.

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. Implementation Experience

Two implementations of RADIUS/DTLS exist, Radsecproxy, and jradius (http://www.coova.org/JRadius). Some experimental tests have been performed, but there are at this time no production implementations using RADIUS/DTLS.

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 leveraging DTLS.

8. 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.

9. IANA Considerations

No new RADIUS attributes or packet codes are defined. IANA is requested to update the already-assigned UDP 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 UDP port 2083 was already previously assigned by IANA for "RadSec", an early implementation

of RADIUS/TLS, prior to issuance of this RFC."

10. Security Considerations

This entire 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. All 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.

The main portion of the specification that could have security implications is a servers ability to accept both RADIUS and DTLS packets on the same port. The filter that disambiguates the two protocols is simple, and is just a check for the value of one octet. We do not expect this check to have any security issues.

We also note that nothing prevents malicious clients from sending DTLS packets to existing RADIUS implementations, or RADIUS packets to existing DTLS implementations. There should therefore be no issue with clients sending RADIUS/DTLS packets to legacy servers that do not support the protocol. These packets will be silently discarded, and will not change the security profile of the server.

This specification also suggests that implementations use a connection 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.

10.1. Legacy RADIUS Security

We reiterate here the poor security of the legacy RADIUS protocol. It is RECOMMENDED that all RADIUS clients and servers implement this specification. 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.

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 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 that are minor variations of words is NOT RECOMMENDED, as it would negate all of the security of DTLS.

10.2. 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.

10.3. 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.

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 a client IP/port combination, followed by the reception of a RADIUS/UDP packet from that same client IP/port combination. If this behavior is allowed, it would permit a downgrade attack to occur, and would negate all of the security added by RADIUS/DTLS.

As a result, RADIUS clients SHOULD NOT be located behind a NAT gateway. If clients are located behind a NAT gateway, then a secure transport such as DTLS MUST be used. As discussed below, a method for uniquely identifying each client MUST be used.

10.4. 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 use the same shared secret.

When a client is a "wildcard", RADIUS/DTLS MUST be used. Clients MUST be uniquely identified, and any certificate or PSK used MUST be unique to each client.

10.5. Session Closing

Section 5.1.1 above requires that DTLS sessions be closed when the transported RADIUS packets are malformed, or fail various authenticator checks. This requirement is due to security considerations.

When an implementation has a DTLS connection, it is expected that the connection be used to transport RADIUS. Any non-RADIUS traffic on that connection means the other party is misbehaving, and a potential security risk. Similarly, any RADIUS traffic failing validation means that two parties do not share the same security parameters, and the session is therefore a security risk.

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

10.6. Clients 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 connections 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.

[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.

[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.

[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