Network Working Group
INTERNET-DRAFT
Category: Informational
<draft-dekok-radext-dtls-03.txt>
Expires: February 25, 2010
26 July 2010

Alan DeKok
FreeRADIUS

DTLS as a Transport Layer for RADIUS
draft-dekok-radext-dtls-03

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 "hidden" content. Packets may be replayed verbatim by an attacker, and client-server authentication is based on fixed shared secrets. This document specifies how the Datagram Transport Layer Security (DTLS) protocol may be used as a fix for these problems. It also describes how implementations of this proposal can co-exist with current RADIUS systems.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The RADIUS protocol as described in [RFC2865], [RFC2866], and [RFC5176] 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, previous 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 [RFC4347] 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 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 continues to have all of the issues that RADIUS currently has with 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.

silently discard

This means that the implementation discards the packet without further processing. The implementation MAY provide the capability of logging the error, including the contents of the silently discarded packet, and SHOULD record the event in a statistics counter.

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 over 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 "encrypted" attributes such as Tunnel-Password.
* UDP port numbering and usage

The RADIUS packets are encapsulated in DTLS, which acts as a transport layer for it. The requirements above ensure the simplest possible implementation and widest interoperability of this specification.

The only changes made to RADIUS in this specification 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 "radsec".

All other portions of RADIUS are unchanged.

2.2. Changes from RADIUS over TLS (RADIUS/TLS)

While this specification is largely RADIUS/TLS over UDP instead of TCP, there are some differences between the two methods.

This section goes through the [RADIUS/TLS] document in detail, explaining the differences between RADIUS/TLS and RADIUS/DTLS. As most of [RADIUS/TLS] also applies to RADIUS/DTLS, we highlight only the changes here, explaining how to interpret [RADIUS/TLS] for this specification:

* 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 text below goes through some of the sections of [RADIUS/TLS], giving additional commentary only where necessary.

2.2.1. Changes from RADIUS/TLS to RADIUS/DTLS

Section 2.1 does not apply to RADIUS/DTLS. The relationship between RADIUS packet codes and UDP ports in RADIUS/DTLS is unchanged from RADIUS.

Section 2.2 applies also to RADIUS/DTLS, except for the recommendation that implementations "SHOULD" support TLS_RSA_WITH_RC4_128_SHA, which does not apply to RADIUS/DTLS.

Section 2.3 applies also to RADIUS/TLS.

Section 2.4 does not apply to RADIUS/DTLS. See the comments above on Section 2.1. The relationship between RADIUS packet codes and UDP ports in RADIUS/DTLS is unchanged from RADIUS.

Section 3.3 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 3.3 item (3) does not apply to RTDLS. The relationship between RADIUS packet codes and UDP ports in RADIUS/DTLS is unchanged from RADIUS.

Section 3.3 item (4) does not apply to RADIUS/DTLS. As RADIUS/DTLS still uses UDP for a transport, the use of negative ICMP responses is unchanged from RADIUS.

2.2.2. Reinforcement of RADIUS/TLS

We wish to re-iterate that much of [RADIUS/TLS] applies to this document. Specifically, Section 4 and Section 6 of that document are applicable in whole to RADIUS/DTLS.

3. Reception of Packets

As this specification permits implementations to to accept both traditional RADIUS and DTLS packets on the same port, we define a method to disambiguate between packets for the two protocols. This method is applicable only to RADIUS servers. RADIUS/DTLS clients SHOULD use connected sockets, as discussed in Section X.Y, below.

RADIUS/DTLS servers MUST maintain a boolean flag for each RADIUS client that indicates whether or not it supports RADIUS/DTLS. The interpretation of this flag is as follows. If the flag is "false", then the client may support RADIUS/DTLS. Packets from the client need to be examined to see if they are RADIUS or RADIUS/DTLS. If the flag is "true" then the client supports RADIUS/DTLS, and all packets from that client MUST be processed as RADIUS/DTLS.

Note that this last requirement can impose significant changes for RADIUS clients. Clients can no longer have multiple independent RADIUS implementations or processes that originate packets. We RECOMMEND that RADIUS/DTLS clients implement a local RADIUS proxy that arbitrates all RADIUS traffic.

This flag MUST be exposed to administrators of the RADIUS server. As RADIUS clients are upgraded, administrators can then manually mark them as supporting RADIUS/DTLS.

We recognize, however, the upgrade path from RADIUS to RADIUS/DTLS is important. This path requires an RADIUS/DTLS server to accept packets from a RADIUS client without knowing beforehand if they are RADIUS or DTLS. The method to distinguish between the two is defined in the next section.

Once an RADIUS/DTLS server has established a DTLS session with a client that had the flag set to "false", it MUST set the flag to "true". This change forces all subsequent traffic from that client to use DTLS, and prevents bidding-down attacks. The server SHOULD also notify the administrator that it has successfully established the first DTLS session with that client.

3.1. Protocol Disambiguation

When a RADIUS client is not marked as supporting RADIUS/DTLS, packets from that client may be, or may not be DTLS. In order to provide a robust upgrade path, the RADIUS/DTLS server MUST examine the packet to see if it is RADIUS or DTLS. In order to justify the examination methods, we first examine the packet formats for the two protocols.

The DTLS record format ([RFC4347] Section 4.1) is shown below, in pseudo-code:

```
struct {
        uint8 type;
        uint16 version;
        uint16 epoch;
        uint48 sequence_number;
        uint16 length;
        uint8 fragment[DTLSPlaintext.length];
} DTLSPlaintext;
```

The RADIUS record format ([RFC2865] Section 3) is shown below, in pseudo-code, with AuthVector.length=16.

```
struct {
        uint8 code;
        uint8 id;
        uint16 length;
        uint8 vector[AuthVector.length];
        uint8 data[RadiusPacket.length - 20];
} RadiusPacket;
```

We can see here that a number of fields overlap between the two protocols. The low byte of the DTLS version and the high byte of the DTLS epoch overlap with the RADIUS length field. The DTLS length field overlaps with the RADIUS authentication vector. At first glance, it may be difficult for an application to accept both protocols on the same port. However, this is not the case.

For the initial packet of a DTLS connection, the type field has value 22 (handshake), and the epoch and sequence number fields are initialized to zero. The RADIUS code value of 22 has been assigned as Resource-Free-Response, but it is not in wide use. In addition, that packet code is a response packet, and would not be sent by a RADIUS client to a server.

As a result, protocol disambiguation is straightforward. If the first byte of the packet has value 22, it is a DTLS packet, and is a DTLS connection initiation request. Otherwise, it is a RADIUS

packet.

Once a DTLS session has been established, a separate tracking table is used to disambiguate the protocols. The definition of this tracking table is given in the next section.

The full processing algorithm is given below, in Section X.Y.

4. Connection Management

Where [RADIUS/TLS] can rely on the TCP state machine to perform connection tracking, this specification cannot. As a result, implementations of this specification will need to perform connection management of the DTLS session in the application layer.

4.1. Server Connection Management

An RADIUS/DTLS server MUST maintain a table that tracks ongoing DTLS sessions based on a key composed of the following 4-tuple:

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

The contents of the tracking table are a implementation-specific value that describes an active DTLS session. This connection tracking allows DTLS packets that have been received to be associated with an active DTLS session.

RADIUS/DTLS servers SHOULD NOT use a "connect" API to manage DTLS connections, as a connected UDP socket will accept packets only from one source IP address and port. This limitation would prevent the server from engaging in the normal RADIUS practice of accepting packets from multiple clients on the same port.

Note that [RFC5080] Section 2.2.2 defines a duplicate detection cache which tracks packets by key similar to that defined above.

4.1.1. Table Management

This tracking table is subject to Denial of Service (DoS) attacks. RADIUS/DTLS servers SHOULD use the stateless cookie tracking technique described in [RFC4347] Section 4.2.1. DTLS sessions SHOULD NOT be added to the tracking table until a ClientHello packet has been received with an appropriate Cookie value.

Entries in the tracking table MUST deleted when a TLS Closure Alert

([RFC5246] Section 7.2.1) or a TLS Error Alert ([RFC5246] Section 7.2.2) is received. Where the RADIUS specifications require that a RADIUS packet received via the DTLS session is to be "silently discarded", the entry in the tracking table corresponding to that DTLS session MUST also be deleted, the DTLS session MUST be closed, and any TLS session resumption parameters for that session MUST be discarded.

As UDP does not offer guaranteed delivery of messages, RADIUS/DTLS servers MUST also maintain a timestamp per DTLS session. The timestamp SHOULD be updated on reception of a valid DTLS packet. The timestamp MUST NOT be updated in other situations. When a session has not been used for a period of time, the server SHOULD pro-actively close it, and delete the DTLS session from the tracking table. The server MAY cache the TLS session parameters, in order to provide for fast session resumption.

This session lifetime SHOULD be exposed as 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 keep track of the total number of sessions in the tracking table, and refuse to create new sessions when a large number are already being tracked. As system capabilities vary widely, we can only recommend that this number SHOULD be exposed as a configurable setting.

4.2. Client Connection Management

RADIUS/DTLS clients SHOULD use an operating system API to "connect" a UDP socket. This "connected" socket will then rely on the operating system to perform connection tracking, and will be simpler than the method described above for servers. RADIUS/DTLS clients SHOULD NOT use "unconnected" sockets, as it causes increased complexity in the client application.

Once a DTLS session is established, an RADIUS/DTLS client SHOULD use the application-layer watchdog algorithm defined in [RFC3539] to determine server responsiveness. The Status-Server packet defined in [STATUS] MUST be used as the "watchdog packet" in the watchdog algorithm.

RADIUS/DTLS clients SHOULD pro-actively close sessions when they have been idle for a period of time. We RECOMMEND that a session be closed when no traffic over than watchdog packets and (possibly) 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.

RADIUS/DTLS clients SHOULD NOT send both normal RADIUS and RADIUS/DTLS packets 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 MUST NOT send both normal RADIUS and RADIUS/DTLS packets over the same key as defined in Section 4.1, abovre (source IP, source port, destination IP, destination port). Doing so would require that servers perform RADIUS and RADIUS/DTLS determination for every packet that has been received.

RADIUS/DTLS clients SHOULD use TLS session resumption, where possible. This practice lowers the time and effort required to start a DTLS session with a server, and increases network responsiveness.

5. Processing Algorithm

The following algorithm MUST be used by an implementation of this protocol. This algorithm is used to route packets to the appropriate destination. We assume the following variables:

D - implementation-specific handle to an existing DTLS session

P - UDP packet received from the network. This packet MUST also contain information about source IP/port, and destination IP/port.

R - a RADIUS packet

T - a tracking table used to manage ongoing DTLS sessions

We also presume the following functions or functionality exists:

receive_packet_from_network() - a function that reads a packet from the network, and returns P as above. We presume also that this function performs the normal RADIUS client validation, and does not return P if the packet is from an unknown client.

lookup_dtls_session() - a function that takes a packet P, a table T, and uses P to look up the corresponding DTLS session in T. It returns either a session D, or a "null" indicator that no corresponding session exists.

client_supports_rdtls() - a function that takes a packet P, and returns a boolean value as to whether or not the client

originating the packet was marked as supporting RADIUS/DTLS.

process_dtls_packet() - a function that takes a DTLS packet P, and a DTLS session D. It performs all necessary steps to use D to setup a DTLS session, and to decode P (where possible) into a RADIUS packet. This function is also expected to perform checks for TLS errors. On any fatal errors, it closes the session, and deletes D from the tracking table T. If a RADIUS packet is decoded from P, it is returned by the function as R, otherwise a "null" indicator is returned.

process_dtls_clienthello() - a function that takes a DTLS packet P, and initiates a DTLS session. If P contains a valid DTLS Cookie, a DTLS session D is created, and stored in the tracking table T. If P does not contain a DTLS Cookie, no session is created, and instead a HelloVerifyRequest containing a cookie is sent in response. Packets containing invalid cookies are discarded.

process_radius_packet() - a function that takes a RADIUS packet P, and processes it using the normal RADIUS methods.

The algorithm is as follows:

```
P = receive_packet_from_network()
D = lookup_dtls_session(T, P)

if (D || client_supports_rdtls(P)) {
   R = process_dtls_packet(D, P)
   if (R) {
      process_radius_packet(R)
   }

} else if (first_octet_of_packet_is_22(P)) {
   process_dtls_clienthello(P)

} else {
   process_radius_packet(P)
}
```

For simplicity, the timers necessary to perform expiry of "old" sessions are not included in the above algorithm. This algorithm may also need to be modified if the RADIUS/DTLS server supports client validation by methods other than source IP address.

6. Diameter Considerations

This specification is for a transport layer specific to RADIUS. As a result, there are no Diameter considerations.

7. IANA Considerations

This specification does not create any new registries, nor does it require assignment of any protocol parameters.

8. 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, 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 [RADIUS/TLS] 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 only new 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 byte. 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.

8.1. Legacy RADIUS Security

We reiterate here the poor security of the legacy RADIUS protocol. We RECOMMEND that all RADIUS clients and servers implement this specification as soon as possible. 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 that depends on RADIUS. Attackers could obtain user passwords, and possibly gain complete network access. It is difficult to 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 MUST NOT be used for DTLS. Re-using a shared secret between RADIUS and DTLS would negate all of the benefits found by using DTLS.

When using PSK methods, RADIUS/DTLS clients MUST support keys (i.e. shared secrets) that are at least 32 characters in length.

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

When using PSK methods, RADIUS/DTLS servers MUST support keys (i.e. shared secrets) that are at least 32 characters in length. RADIUS/DTLS server administrators MUST use strong shared secrets for those PSK methods. We RECOMMEND using keys derived from a cryptographically secure pseudo-random number generator (CSPRNG). For example, a reasonable key may be 32 characters of a SHA-256 hash of at least 64 bytes of data taken from a CSPRNG. If this method seems too complicated, a certificate-based TLS method SHOULD be used instead.

The previous RADIUS practice of using shared secrets that are minor variations of words is NOT RECOMMENDED, as it would negate nearly all of the security of DTLS.

9. References

9.1. 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.

[RFC4347]
Rescorla E., and Modadugu, N., "Datagram Transport Layer Security", RFC 4347, April 2006.

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

[RADIUS/TLS]
Winter. S, et. al., "TLS encryption for RADIUS over TCP", draft-ietf-radext-radsec-06.txt, March 2010 (work in progress)

[STATUS]
DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", draft-ietf-radext-status-server-09.txt, May 2010 (work in progress).

9.2. 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.

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

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

[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.

The author would like to thank Mike McCauley of Open Systems Consultants for making a Radiator server available for inter-operability testing.

Authors' Addresses

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

Email: aland@freeradius.org