Stream: Internet Engineering Task Force (IETF)
RFC: 9765
Updates: 2865, 2866, 5176, 6613, 6614, 7360
Category: Experimental
Published: April 2025
ISSN: 2070-1721
Author: A. DeKok
FreeRADIUS

RFC 9765
RADIUS/1.1: Leveraging Application-Layer Protocol Negotiation (ALPN) to Remove MD5

Abstract

This document defines Application-Layer Protocol Negotiation (ALPN) extensions for use with RADIUS/TLS and RADIUS/DTLS. These extensions permit the negotiation of an application protocol variant of RADIUS called "RADIUS/1.1". No changes are made to RADIUS/UDP or RADIUS/TCP. The extensions allow the negotiation of a transport profile where the RADIUS shared secret is no longer used, and all MD5-based packet authentication and attribute obfuscation methods are removed.

This document updates RFCs 2865, 2866, 5176, 6613, 6614, and 7360.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
2. Terminology 6
3. The RADIUS/1.1 Transport Profile for RADIUS 7
3.1. ALPN Name for RADIUS/1.1 7
3.2. Operation of ALPN 8
3.3. Configuration of ALPN for RADIUS/1.1 8
3.3.1. Using Protocol-Error for Signaling ALPN Failure 12
3.3.2. Tabular Summary 12
3.4. Miscellaneous Items 14
3.5. Session Resumption 14
4. RADIUS/1.1 Packet and Attribute Formats 15
4.1. RADIUS/1.1 Packet Format 15
4.2. The Token Field 16
4.2.1. Sending Packets 16
4.2.2. Receiving Packets 18
5. Attribute Handling 18
5.1. Obfuscated Attributes 19
5.1.1. User-Password 19
5.1.2. CHAP-Challenge 20
5.1.3. Tunnel-Password 20
5.1.4. Vendor-Specific Attributes 20
5.2. Message-Authenticator 21
5.3. Message-Authentication-Code 21
5.4. CHAP, MS-CHAP, and Similar Attributes 22

5.5. Original-Packet-Code 22
6. Other Considerations When Using ALPN 23
6.1. Protocol-Error 23
6.2. Status-Server 24
6.3. Proxies 24
7. Other RADIUS Considerations 25
7.1. Crypto-Agility 25
7.2. Error-Cause Attribute 25
7.3. Future Standards 26
8. Privacy Considerations 27
9. Security Considerations 27
10. IANA Considerations 28
11. References 28
11.1. Normative References 28
11.2. Informative References 29
Acknowledgments 31
Author's Address 31

1. Introduction

The RADIUS protocol [RFC2865] uses MD5 [RFC1321] to authenticate packets and to obfuscate certain attributes. Additional transport protocols were defined for TCP [RFC6613], TLS [RFC6614], and DTLS [RFC7360]. However, those transport protocols still use MD5 to authenticate individual packets. That is, the shared secret was used along with MD5, even when the RADIUS packets were being transported in (D)TLS. At the time, the consensus of the RADEXT Working Group was that this continued use of MD5 was acceptable. TLS was seen as a simple "wrapper" around RADIUS, while using a fixed shared secret. The intention at the time was to allow the use of (D)TLS while making essentially no changes to the basic RADIUS encoding, decoding, authentication, and packet validation.

Issues of MD5 security have been known for decades, most notably in [RFC6151] and in Section 3 of [RFC6421], among others. The reliance on MD5 for security makes it impossible to use RADIUS in secure systems that forbid the use of digest algorithms with known vulnerabilities. For example, FIPS 140 forbids systems from relying on insecure cryptographic methods for security [FIPS-140-3].

While the use of MD5 in RADIUS/TLS has not been proven to be insecure, it has not been proven to be secure. This gap means that it is difficult to use RADIUS in organizations that require the use of systems that have proven security. Those organizations tend to simply ban the use of insecure digests such as MD5 entirely, even if the use of MD5 has no known security impact. While the resulting system might still not be secure, it at least does not contain any known insecurities.

In addition, the use of MD5 in RADIUS/TLS and RADIUS/DLTS adds no security or privacy over that provided by TLS. In hindsight, the decision of the RADEXT Working Group to retain MD5 for historic RADIUS/TLS was likely wrong. It was an easy decision to make in the short term, but it has caused ongoing problems that this document addresses. The author of this document played a part in that original decision, which is now being corrected by this document.

This document defines an Application-Layer Protocol Negotiation (ALPN) [RFC7301] extension for RADIUS over (D)TLS that removes the need to use MD5 for (D)TLS, which we call RADIUS/1.1. This specification makes no changes to UDP or TCP transport. The RADIUS/1.1 protocol can be best understood as a transport profile for RADIUS over TLS, rather than a wholesale revision of the RADIUS protocol.

Systems that implement this transport profile can be more easily verified to be FIPS 140 compliant. A preliminary implementation has shown that only minor code changes are required to support RADIUS/1.1 on top of an existing RADIUS/TLS server implementation. These include:

- A method to set the list of supported ALPN protocols before the TLS handshake starts.
- A method to query if ALPN has chosen a protocol (and if yes, which protocol was chosen) after the TLS handshake has completed.
- Changes to the packet encoder and decoder, so that the individual packets are not authenticated, and no attribute is encoded with the historic obfuscation methods.

That is, the bulk of the ALPN protocol can be left to the underlying TLS implementation. This document discusses the ALPN exchange in detail in order to give simplified descriptions for the reader, and so that the reader does not have to read or understand all of [RFC7301].

The detailed list of changes from historic TLS-based transports to RADIUS/1.1 is as follows:

- ALPN is used for negotiation of this extension.
- TLS 1.3 or later is required.
- All uses of the RADIUS shared secret have been removed.
- The now unused Request and Response Authenticator fields have been repurposed to carry an opaque Token that identifies requests and responses.

- The functionality of the Identifier field has been replaced by the Token field, and the space previously taken by the Identifier field is now reserved and unused.
- The Message-Authenticator attribute ([RFC3579], Section 3.2) is not sent in any packet, and is ignored if received.
- Attributes such as User-Password, Tunnel-Password, and MS-MPPE keys are sent encoded as "text" ([RFC8044], Section 3.4) or "octets" ([RFC8044], Section 3.5), without the previous MD5-based obfuscation. This obfuscation is no longer necessary, as the data is secured and kept private through the use of TLS.
- The conclusion of the efforts stemming from [RFC6421] is that crypto-agility in RADIUS is best done via a TLS wrapper, and not by extending the RADIUS protocol.
- [RFC5176] is updated to allow the Error-Cause attribute to appear in Access-Reject packets.

The following items are left unchanged from historic TLS-based transports for RADIUS:

- The RADIUS packet header is the same size, and the Code and Length fields ([RFC2865], Section 3) have the same meaning as before.
- The default 4096-octet packet size from [RFC2865], Section 3 is unchanged, although [RFC7930] can still be leveraged to use larger packets.
- All attributes that have simple encodings (that is, attributes that do not use MD5 obfuscation) have the same encoding and meaning as before.
- As this extension is a transport profile for one "hop" (client-to-server connection), it does not impact any other connection used by a client or server. The only systems that are aware that this transport profile is in use are the client and server who have negotiated the use of this extension on a particular shared connection.
- This extension uses the same ports (2083/tcp and 2083/udp) that are defined for RADIUS/TLS [RFC6614] and RADIUS/DTLS [RFC7360].

A major benefit of this extension is that a server that implements it can also be more easily verified for FIPS 140 compliance. That is, a server can remove all uses of MD5, which means that those algorithms are provably not used for security purposes. In that case, however, the server will not support the Challenge Handshake Authentication Protocol (CHAP) or any authentication method that uses MD5. The choice of which authentication method to accept is always left to the server. This specification does not change any authentication method carried in RADIUS, and does not mandate (or forbid) the use of any authentication method for any system.

As for proxies, there was never a requirement that proxies implement CHAP or Microsoft CHAP (MS-CHAP) authentication. So far as a proxy is concerned, attributes relating to CHAP and MS-CHAP are simply opaque data that is transported unchanged to the next hop. Therefore, it is possible for a FIPS 140 compliant proxy to transport authentication methods that depend on MD5, so long as that data is forwarded to a server that supports those methods.

We reiterate that the decision to support (or not support) any authentication method is entirely site local, and is not a requirement of this specification. The contents or meaning of any RADIUS attribute other than the Message-Authenticator (and similar attributes) are not modified. The only change to the Message-Authenticator attribute is that it is no longer used in RADIUS/1.1.

Unless otherwise described in this document, all RADIUS requirements apply to this extension. That is, this specification defines a transport profile for RADIUS. It is not an entirely new protocol, and it defines only minor changes to the existing RADIUS protocol. It does not change the RADIUS packet format, attribute format, etc. This specification is compatible with all RADIUS attributes of the past, present, and future.

This specification is compatible with existing implementations of RADIUS/TLS and RADIUS/DTLS. Systems that implement this specification can fall back to historic RADIUS/TLS if no ALPN signaling is performed, and the local configuration permits such fallback.

This specification is compatible with all existing RADIUS specifications. There is no need for any RADIUS specification to mention this transport profile by name or to make provisions for this specification. This document defines how to transform RADIUS into RADIUS/1.1, and no further discussion of that transformation is necessary.

We note that this document makes no changes to previous RADIUS specifications. Existing RADIUS implementations can continue to be used without modification. Where previous specifications are explicitly mentioned and updated, those updates or changes apply only when the RADIUS/1.1 transport profile is being used.

In short, when negotiated on a connection, the RADIUS/1.1 transport profile permits implementations to avoid MD5 when authenticating packets or when obfuscating certain attributes.

2. Terminology

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

The following list describes the terminology and abbreviations that are used in this document.

ALPN
: Application-Layer Protocol Negotiation (as defined in [RFC7301]).

RADIUS
: Remote Authentication Dial-In User Service (as defined in [RFC2865], [RFC2866], and [RFC5176], among others).

  While this protocol can be viewed as "RADIUS/1.0", for simplicity and historical compatibility, we keep the name "RADIUS".

RADIUS/UDP
: RADIUS over the User Datagram Protocol (see [RFC2865], [RFC2866], and [RFC5176], among others).

RADIUS/TCP
: RADIUS over the Transmission Control Protocol [RFC6613].

RADIUS/TLS
: RADIUS over Transport Layer Security [RFC6614].

RADIUS/DTLS
: RADIUS over Datagram Transport Layer Security [RFC7360].

RADIUS over TLS
: Refers to any RADIUS packets transported over TLS or DTLS. This terminology is used instead of alternatives such as "RADIUS/(D)TLS" or "either RADIUS/TLS or RADIUS/DTLS". This term is generally used when referring to TLS-layer requirements for RADIUS packet transport.

historic RADIUS/TLS
: Refers to RADIUS over (D)TLS (as defined in [RFC6614] and [RFC7360]). This term does not include the protocol defined in this specification.

RADIUS/1.1
: RADIUS version 1.1, i.e., the transport profile defined in this document. We use RADIUS/1.1 to refer interchangeably to TLS and DTLS transport.

TLS
: Transport Layer Security. Generally, when we refer to TLS in this document, we are referring interchangeably to TLS or DTLS transport.

3. The RADIUS/1.1 Transport Profile for RADIUS

This section describes the ALPN transport profile in detail. It first gives the name used for ALPN, and then describes how ALPN is configured and negotiated by the client and server. It then concludes by discussing TLS issues such as what to do for ALPN during session resumption.

3.1. ALPN Name for RADIUS/1.1

The ALPN name defined for RADIUS/1.1 is as follows:

"radius/1.1"
: The protocol defined by this specification.

Where ALPN is not configured or is not received in a TLS connection, systems supporting ALPN **MUST NOT** use RADIUS/1.1.

Where ALPN is configured, the client signals support by sending ALPN strings listing which protocols it supports. The server can accept one of these proposals and reply with a matching ALPN string, or reject this proposal and not reply with any ALPN string. A full walkthrough of the protocol negotiation is given below.

Implementations **MUST** signal ALPN "radius/1.1" in order for it to be used in a connection.

The next step in defining RADIUS/1.1 is to review how ALPN works.

3.2. Operation of ALPN

In order to provide a high-level description of ALPN for readers who are not familiar with the details of [RFC7301], we provide a brief overview here.

Once a system has been configured to support ALPN, it is negotiated on a per-connection basis as per [RFC7301]. The negotiation proceeds as follows:

1) The client sends an ALPN extension in the ClientHello. This extension lists one or more application protocols by name. These names are the protocols that the client is claiming to support.

2) The server receives the extension and validates the application protocol name(s) against the list it has configured.

   If the server finds no acceptable common protocols (ALPN or otherwise), it closes the connection.

3) Otherwise, the server returns a ServerHello with either no ALPN extension or an ALPN extension containing only one named application protocol, which needs to be one of the names proposed by the client.

   If the client did not signal ALPN, or the server does not accept the ALPN proposal, the server does not reply with any ALPN name.

4) The client receives the ServerHello, validates the received application protocol (if any) against the name(s) it sent, and records which application protocol was chosen.

   This check is necessary in order for the client to both know which protocol the server has selected, and to validate that the protocol sent by the server is one that is acceptable to the client.

The next step in defining RADIUS/1.1 is to define how ALPN is configured on the client and server and to give more detailed requirements on its configuration and operation.

3.3. Configuration of ALPN for RADIUS/1.1

Clients or servers supporting this specification can do so by extending their TLS configuration through the addition of a new configuration variable, called "Version" here. The exact name given below does not need to be used, but it is **RECOMMENDED** that administrative interfaces or programming interfaces use a similar name in order to provide consistent terminology. This variable controls how the implementation signals use of this protocol via ALPN.

When set, this variable should contain the list of permitted RADIUS versions as numbers, e.g., "1.0" or "1.1". The implementation may allow multiple values in one variable, allow multiple variables, or instead use two configurations for the "minimum" and "maximum" allowed versions. We assume here that there is one variable, which can contain either no value or a list of one or more versions that the current implementation supports. In this specification, the possible values, ALPN strings, and corresponding interpretations are:

Value	ALPN String(s)	Interpretation
unset		no ALPN strings are sent
1.0	radius/1.0	require historic RADIUS/TLS
1.0, 1.1	radius/1.0, radius/1.1	allow either historic RADIUS/TLS or RADIUS/1.1
1.1	radius/1.1	require RADIUS/1.1

Table 1

This configuration is also extensible to future RADIUS versions if that extension becomes necessary. New values and ALPN names can simply be added to the list. Implementations can then negotiate the highest version that is supported by both client and server.

Implementations **SHOULD** support both historic RADIUS/TLS and RADIUS/1.1. Such implementations **MUST** set the default value for this configuration variable to "1.0, 1.1". This setting ensures that both versions of RADIUS can be negotiated.

Implementations **MAY** support only RADIUS/1.1. In this case, the default value for this configuration variable **MUST** be "1.1". This behavior is **NOT RECOMMENDED**, as it is incompatible with historic RADIUS/TLS. This behavior can only be a reasonable default when all (or nearly all) RADIUS clients have been updated to support RADIUS/1.1.

A more detailed definition of the variable and the meaning of the values is given below.

Configuration Variable Name
: Version

For "Value":

A. If unset, ALPN is not used.

   Any connection **MUST** use historic RADIUS/TLS.

   This variable is included here only for logical completeness. Implementations of this specification **SHOULD** be configured to always send one or more ALPN strings. This data signals that the implementation is capable of performing ALPN negotiation, even if it is not currently configured to use RADIUS/1.1.

   Client Behavior
   : The client **MUST NOT** send any protocol name via ALPN.

   Server Behavior
   : The server **MUST NOT** signal any protocol name via ALPN.

If the server receives an ALPN name from the client, it **MUST NOT** close the connection. Instead, it simply does not reply with ALPN and finishes the TLS connection setup as defined for historic RADIUS/TLS.

Note that if a client sends "radius/1.1", the client will see that the server failed to acknowledge this request and will close the connection. For any other client configuration, the connection will use historic RADIUS/TLS.

B. If set to "1.0", "1.0, 1.1", "1.1", or future values:

Client Behavior

The client **MUST** send the ALPN string(s) associated with the configured version. For example, send "radius/1.0" for "1.0".

The client will receive either no ALPN response from the server; or it will receive an ALPN response of one version string that **MUST** match one of the strings it sent; or else they will receive a TLS alert of "no_application_protocol" (120).

If the connection remains open, the client **MUST** treat the connection as using the matching ALPN version.

Server Behavior

If the server receives no ALPN name from the client, it **MUST** use historic RADIUS/TLS.

If the server receives one or more ALPN names from the client, it **MUST** reply with the highest mutually supported version and then use the latest supported version for this connection.

If the server receives one or more ALPN names from the client, but none of the names match the versions supported by (or configured on) the server, it **MUST** reply with a TLS alert of "no_application_protocol" (120), and then it **MUST** close the TLS connection.

These requirements for negotiation are not specific to RADIUS/1.1; therefore, they can be used unchanged if any new version of RADIUS is defined.

By requiring the default configuration to allow historic RADIUS/TLS, implementations will be able to negotiate both historic RADIUS/TLS connections and also RADIUS/1.1 connections. Any other recommended default setting would prevent either the negotiation of historic RADIUS/TLS or prevent the negotiation of RADIUS/1.1.

Once administrators verify that both ends of a connection support RADIUS/1.1, and that it has been negotiated successfully, the configurations **SHOULD** be updated to require RADIUS/1.1. The connections should be monitored after this change to ensure that the systems continue to remain connected. If there are connection issues, then the configuration should be reverted to allowing both "radius/1.0" and "radius/1.1" ALPN strings, until the administrator has resolved the connection problems.

We reiterate that systems implementing this specification, but which are configured with settings that forbid RADIUS/1.1, will behave largely the same as systems that do not implement this specification. The only difference is that clients may send the ALPN name "radius/1.0".

Systems implementing RADIUS/1.1 **SHOULD NOT** be configured by default to forbid that protocol. That setting exists mainly for completeness, and to give administrators the flexibility to control their own deployments.

While [RFC7301] does not discuss the possibility of the server sending a TLS alert of "no_application_protocol" (120) when the client does not use ALPN, this behavior appears to be useful. As such, servers **MAY** send a TLS alert of "no_application_protocol" (120) when the client does not use ALPN.

However, some TLS implementations may not permit an application to send a TLS alert of its choice at a time of its choice. This limitation means that it is not always possible for an application to send the TLS alert as discussed in the previous section. The impact is that an implementation may attempt to connect and then see that the connection fails, but it may not be able to determine why that failure has occurred. Implementers and administrators should be aware that unexplained connection failures may be due to ALPN issues.

The server **MAY** send this alert during the ClientHello if it requires ALPN but does not receive it. That is, there may not always be a need to wait for the TLS connection to be fully established before realizing that no common ALPN protocol can be negotiated.

Where the client does perform signaling via ALPN, and the server determines that there is no compatible application protocol name, then as per [RFC7301], Section 3.2, it **MUST** send a TLS alert of "no_application_protocol" (120).

The server **MUST** close the connection whether or not the server sent a TLS alert for no compatible ALPN. The above requirements on ALPN apply to both new sessions and to resumed sessions.

In contrast, there is no need for the client to signal that there are no compatible application protocol names. The client sends zero or more protocol names, and the server responds as above. From the point of view of the client, the list it sent results in either a connection failure or a connection success.

It is **RECOMMENDED** that the server logs a descriptive error in this situation, so that an administrator can determine why a particular connection failed. The log message **SHOULD** include information about the other end of the connection, such as the IP address, certificate information, etc. Similarly, when the client receives a TLS alert of "no_application_protocol" (120), it **SHOULD** log a descriptive error message. Such error messages are critical for helping administrators diagnose connectivity issues.

3.3.1. Using Protocol-Error for Signaling ALPN Failure

When it is not possible to send a TLS alert of "no_application_protocol" (120), then the only remaining method for one party to signal the other is to send application data inside of the TLS tunnel. Therefore, for the situation when one end of a connection determines that it requires ALPN, while the other end does not support ALPN, then the end requiring ALPN **MAY** send a Protocol-Error packet [RFC7930] inside of the tunnel and then **MUST** close the connection. If this is done, the Token field of the Protocol-Error packet cannot be copied from any request; therefore, that field **MUST** be set to all zeros.

The Protocol-Error packet **SHOULD** contain a Reply-Message attribute with a textual string describing the cause of the error. The packet **SHOULD** also contain an Error-Cause attribute, with value 406 (Unsupported Extension). The packet **SHOULD NOT** contain other attributes.

An implementation sending this packet could bypass any RADIUS encoder and simply write this packet as a predefined, fixed set of data to the TLS connection. That process would likely be simpler than trying to call the normal RADIUS packet encoder to encode a reply packet with no corresponding request packet.

As this packet is an unexpected response packet, existing client implementations of RADIUS over TLS will ignore it. They may either log an error and close the connection, or they may discard the packet and leave the connection open. If the connection remains open, the end supporting ALPN will close the connection, so there will be no side effects from sending this packet. Therefore, while using a Protocol-Error packet in this way is unusual, it is both informative and safe.

The purpose of this packet is not to have the other end of the connection automatically determine what went wrong and fix it. Instead, the packet is intended to be (eventually) seen by an administrator, who can then take remedial action.

3.3.2. Tabular Summary

The preceding text gives a large number of recommendations. In order to give a simpler description of the outcomes, a table of possible behaviors for client/server values of the Version variable is given below. The row and column headings are the RADIUS version numbers sent in ALPN (or no ALPN). The contents of the table are the resulting RADIUS version that is negotiated. For clarity, only the RADIUS version numbers have been given, and not the full ALPN strings (e.g., "radius/1.0").

This table and the names given below are for informational and descriptive purposes only.

Client	Server			
	no ALPN	**1.0**	**1.0, 1.1**	**1.1**
no ALPN	TLS	TLS	TLS	Close-S

Client	Server			
	no ALPN	**1.0**	**1.0, 1.1**	**1.1**
1.0	TLS	TLS	TLS	Alert
1.0, 1.1	TLS	TLS	1.1	1.1
1.1	Close-C	Alert	1.1	1.1

Table 2: Possible Outcomes for ALPN

The table entries above have the following meaning:

Alert
: The client sends ALPN, and the server does not agree to the client's ALPN proposal. The server replies with a TLS alert of "no_application_protocol" (120) and then closes the TLS connection.

  As the server replies with a TLS alert, the Protocol-Error packet is not used here.

Close-C
: The client sends ALPN, but the server does not respond with ALPN. The client closes the connection.

  As noted in the previous section, the client **MAY** send a Protocol-Error packet to the server before closing the connection.

Close-S
: The client does not send ALPN string(s), but the server requires ALPN. The server closes the connection.

  As noted in the previous section, the server **MAY** send a Protocol-Error packet to the client before closing the connection. The server **MAY** also send a TLS alert of "no_application_protocol" (120) before closing the connection.

TLS
: Historic RADIUS/TLS is used. The client either sends no ALPN string or sends "radius/1.0". The server either replies with no ALPN string or with "radius/1.0". The connection **MUST** use historic RADIUS/TLS.

1.1
: The client sends the ALPN string "radius/1.1". The server acknowledges this negotiation with a reply of "radius/1.1", and then RADIUS/1.1 is used.

Implementations should note that this table may be extended in future specifications. The above text is informative, and does not mandate that only the above ALPN strings are used. The actual ALPN takes place as defined in the preceding sections of this document and in [RFC7301].

3.4. Miscellaneous Items

Implementations of this specification **MUST** require TLS version 1.3 or later.

The use of the ALPN string "radius/1.0" is technically unnecessary, as it is largely equivalent to not sending any ALPN string. However, that value is useful for RADIUS administrators. A system that sends the ALPN string "radius/1.0" is explicitly signaling that it supports ALPN, but that it is not currently configured to support RADIUS/1.1. That information can be used by administrators to determine which devices are capable of ALPN.

The use of the ALPN string "radius/1.0" also permits server implementations to send a TLS alert of "no_application_protocol" (120) when it cannot find a matching ALPN string. Experiments with TLS library implementations suggest that in some cases it is possible to send that TLS alert when ALPN is not used. However, such a scenario is not discussed in [RFC7301] and is likely not universal. As a result, ALPN as defined in [RFC7301] permits servers to send that TLS alert in situations where it would be otherwise forbidden or perhaps unsupported.

Finally, defining ALPN strings for all known RADIUS versions will make it easier to support additional ALPN strings if that functionality is ever needed.

3.5. Session Resumption

[RFC7301], Section 3.1 states that ALPN is negotiated on each connection, even if session resumption is used:

> When session resumption or session tickets [RFC5077] are used, the previous contents of this extension are irrelevant, and only the values in the new handshake messages are considered.

(Note: RFC 5077 was obsoleted by [RFC8446].)

In order to prevent down-bidding attacks, RADIUS systems that negotiate the "radius/1.1" protocol **MUST** associate that information with the session ticket and enforce the use of "radius/1.1" on session resumption. That is, if "radius/1.1" was negotiated for a session, both clients and servers **MUST** behave as if the RADIUS/1.1 variable was set to "require" for that session.

A client that is resuming a "radius/1.1" connection **MUST** advertise only the capability to do "radius/1.1" for the resumed session. That is, even if the client configuration allows historic RADIUS/TLS for new connections, it **MUST** signal "radius/1.1" when resuming a session that had previously negotiated "radius/1.1".

Similarly, when a server does resumption for a session that had previously negotiated "radius/1.1", if the client attempts to resume the sessions without signaling the use of RADIUS/1.1, the server **MUST** close the connection. The server **MUST** send an appropriate TLS error, and also **SHOULD** log a descriptive message as described above.

In contrast, there is no requirement for a client or server to force the use of RADIUS/TLS from [RFC6614] on session resumption. Clients are free to signal support for "radius/1.1" on resumed sessions, even if the original session did not negotiate "radius/1.1". Servers are free to accept this request and to negotiate the use of "radius/1.1" for such sessions.

4. RADIUS/1.1 Packet and Attribute Formats

This section describes the application-layer data that is sent inside of (D)TLS when using the RADIUS/1.1 protocol. Unless otherwise discussed herein, the application-layer data is unchanged from historic RADIUS. This protocol is only used when "radius/1.1" has been negotiated by both ends of a connection.

4.1. RADIUS/1.1 Packet Format

When RADIUS/1.1 is used, the RADIUS header is modified from standard RADIUS. While the header has the same size, some fields have different meanings. The Identifier and the Request and Response Authenticator fields are no longer used in RADIUS/1.1. Any operations that depend on those fields **MUST NOT** be performed. As packet authentication, secrecy, and security are handled by the TLS layer, RADIUS-specific cryptographic primitives are no longer needed or used in RADIUS/1.1.

A summary of the RADIUS/1.1 packet format is shown below. The fields are transmitted from left to right.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Code      |  Reserved-1   |            Length             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                             Token                             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 |                           Reserved-2                          |
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Attributes ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Figure 1: The RADIUS/1.1 Packet Format

Code
: The Code field is one octet and identifies the type of RADIUS packet.

  The meaning of the Code field is unchanged from previous RADIUS specifications.

Reserved-1
: The Reserved-1 field is one octet.

This field was previously used as the "Identifier" in historic RADIUS/TLS. It is now unused, as the Token field replaces it both as the way to identify requests and to associate responses with requests.

When sending packets, the Reserved-1 field **MUST** be set to zero. The Reserved-1 field **MUST** be ignored when receiving a packet.

Length
: The Length field is two octets.

The meaning of the Length field is unchanged from previous RADIUS specifications.

Token
: The Token field is four octets and aids in matching requests and replies, as a replacement for the Identifier field. The RADIUS server can detect a duplicate request if it receives the same Token value for two packets on a particular connection.

All values are possible for the Token field. Implementations **MUST** treat the Token as an opaque blob when comparing Token values.

Further requirements are given below in Section 4.2.1 for sending packets and in Section 4.2.2 for receiving packets.

Reserved-2
: The Reserved-2 field is twelve (12) octets in length.

These octets **MUST** be set to zero when sending a packet.

These octets **MUST** be ignored when receiving a packet.

These octets are reserved for future protocol extensions.

4.2. The Token Field

This section describes in more detail how the Token field is used.

4.2.1. Sending Packets

The Token field **MUST** change for every new unique packet that is sent on the same connection. For DTLS transport, it is possible to retransmit duplicate packets, in which case the Token value **MUST NOT** be changed when a duplicate packet is (re)sent. When the contents of a retransmitted packet change for any reason (such as changing Acct-Delay-Time as discussed in [RFC2866], Section 5.2), the Token value **MUST** be changed. Note that on reliable transports, packets are never retransmitted; therefore, every new packet that is sent has a unique Token value.

We note that in previous RADIUS specifications, the Identifier field could have the same value for different packets on the same connection. For example, Access-Request (Code 1) and Accounting-Request (Code 4) packets could both use ID 3 and still be treated as different packets.

This overlap requires that RADIUS clients and servers track the Identifier field, not only on a per-connection basis, but also on a per-Code basis. This behavior adds complexity to implementations.

In contrast, the Token values **MUST** be generated from a 32-bit counter that is unique to each connection. Such a counter **SHOULD** be initialized to a random value, taken from a random number generator, whenever a new connection is opened. The counter **MUST** then be incremented for every unique new packet that is sent by the client. Retransmissions of the same packet **MUST** use the same unchanged Token value. As the Token value is mandated to be unique per packet, a duplicate Token value is the only way that a server can detect duplicate transmissions.

This counter method ensures that the Tokens are unique and are also independent of any Code value in the RADIUS packet header. This method is mandated because any other method of generating unique and non-conflicting Token values is more complex, with no additional benefit and only the likelihood of increased bugs and interoperability issues. Any other method for generating Token values would require substantially more resources to track outstanding Token values and their associated expiry times. The chance that initial values could potentially cause any confusion by being reused across two connections is one in 2^{32}, which is acceptable.

The purpose for initializing the Token to a random counter is mainly to aid administrators in debugging systems. If the Token values always used the same sequence, then it would easier for a person to confuse different packets that have the same Token value. By instead starting with a random value, those values are more evenly distributed across the set of allowed values; therefore, they are more likely to be unique.

As there is no special meaning for the Token, there is no meaning when a counter "wraps" around from a high value back to zero. The originating system can simply continue to increment the Token value without taking any special action in that situation.

Once a RADIUS response to a request has been received and there is no need to track the packet any longer, the Token value can be reused. This reuse happens only when the counter "wraps around" after 2^{32} packets have been sent over one connection. This method of managing the counter automatically ensures a long delay (i.e., 2^{32} packets) between multiple uses of the same Token value. This large number of packets ensures that the only possible situation where there may be conflict is when a client sends billions of packets a second across one connection, or when a client sends billions of packets without receiving replies. We suggest that such situations are vanishingly rare. The best solution to those situations would be to limit the number of outstanding packets over one connection to a number much lower than billions.

If a RADIUS client has multiple independent subsystems that send packets to a server, each subsystem **MAY** open a new connection that is unique to that subsystem. There is no requirement that all packets go over one particular connection. That is, despite the use of a 32-bit Token field, RADIUS/1.1 clients are still permitted to open multiple source ports as discussed in [RFC2865], Section 2.5.

While multiple connections from client to server are allowed, we reiterate the suggestion of [RFC3539], Section 3.3 that a single connection is preferred to multiple connections. The use of a single connection can improve throughput and latency, while simplifying the client's efforts to determine server status.

4.2.2. Receiving Packets

A server that receives RADIUS/1.1 packets **MUST** perform packet deduplication for all situations where it is required by RADIUS. Where RADIUS does not require deduplication (e.g., TLS transport), the server **SHOULD NOT** do deduplication. However, DTLS transport is UDP-based, and therefore still requires deduplication.

When using RADIUS/1.1, implementations **MUST** do deduplication only on the Token field, and not on any other field or fields in the packet header. A server **MUST** treat the Token as being an opaque field with no intrinsic meaning. This requirement makes the receiver behavior independent of the methods by which the Counter is generated.

Where Token deduplication is done, it **MUST** be done on a per-connection basis. If two packets that are received on different connections contain the same Token value, then those packets **MUST** be treated as distinct (i.e., different) packets. Systems performing deduplication **MAY** still track the packet Code, Length, and Attributes that are associated with a Token value. If it determines that the sender is reusing Token values for distinct outstanding packets, then an error should be logged, and the connection **MUST** be closed. There is no way to negotiate correct behavior in the protocol. Either both parties operate normally and can communicate, or one end misbehaves and no communication is possible.

Once a reply has been sent, a system doing deduplication **SHOULD** cache the replies as discussed in [RFC5080], Section 2.2.2:

> Each cache entry **SHOULD** be purged after a period of time. This time **SHOULD** be no less than 5 seconds, and no more than 30 seconds. After about 30 seconds, most RADIUS clients and end users will have given up on the authentication request. Therefore, there is little value in having a larger cache timeout.

This change from RADIUS means that the Identifier field is no longer useful for RADIUS/1.1. The Reserved-1 field (previously used as the Identifier) **MUST** be set to zero when encoding all RADIUS/1.1 packets. Implementations of RADIUS/1.1 that receive packets **MUST** ignore this field.

5. Attribute Handling

Most attributes in RADIUS have no special encoding "on the wire", or any special meaning between client and server. Unless discussed in this section, all RADIUS attributes are unchanged in this specification. This requirement includes attributes that contain a tag, as defined in [RFC2868].

5.1. Obfuscated Attributes

Since the (D)TLS layer provides for connection authentication, integrity checks, and confidentiality, there is no need to hide the contents of an attribute on a hop-by-hop basis. As a result, all attributes defined as being obfuscated via the shared secret no longer have the obfuscation step applied when RADIUS/1.1 is used. Instead, those attributes **MUST** be encoded using the encoding for the underlying data type, with any encryption / obfuscation step omitted. For example, the User-Password attribute is no longer obfuscated and is instead sent as data type "text".

There are risks from sending passwords over the network, even when they are protected by TLS. One such risk comes from the common practice of multi-hop RADIUS routing. As all security in RADIUS is on a hop-by-hop basis, every proxy that receives a RADIUS packet can see (and modify) all of the information in the packet. Sites wishing to avoid proxies **SHOULD** use dynamic peer discovery [RFC7585], which permits clients to make connections directly to authoritative servers for a realm.

There are other ways to mitigate these risks. The simplest is to follow the requirements of item (3) from [RFC6614], Section 3.4 and also follow [RFC7360], Section 10.4, which mandates that RADIUS over TLS implementations validate the peer before sending any RADIUS traffic.

Another way to mitigate these risks is for the system being authenticated to use an authentication protocol that never sends passwords (e.g., an Extensible Authentication Protocol (EAP) method like EAP-pwd [RFC5931]), or one that sends passwords protected by a TLS tunnel (e.g., EAP Tunneled Transport Layer Security (EAP-TTLS) [RFC5281]). The processes to choose and configure an authentication protocol are strongly site dependent, so further discussions of these issues are outside of the scope of this document. The goal here is to ensure that the reader has enough information to make an informed decision.

We note that as the RADIUS shared secret is no longer used in this specification, it is no longer possible or necessary for any attribute to be obfuscated on a hop-by-hop basis using the previous methods defined for RADIUS.

5.1.1. User-Password

The User-Password attribute ([RFC2865], Section 5.2) **MUST** be encoded the same as any other attribute of data type "string" ([RFC8044], Section 3.5).

The contents of the User-Password field **MUST** be at least one octet in length and **MUST NOT** be more than 128 octets in length. This limitation is maintained from [RFC2865], Section 5.2 for compatibility with historic transports.

Note that the User-Password attribute is not of data type "text". The original reason in [RFC2865] was because the attribute was encoded as an opaque and obfuscated binary blob. This document does not change the data type of User-Password, even though the attribute is no longer obfuscated. The contents of the User-Password attribute do not have to be printable text or UTF-8 data as per the definition of the "text" data type in [RFC8044], Section 3.4.

However, implementations should be aware that passwords are often printable text, and where the passwords are printable text, it can be useful to store and display them as printable text. Where implementations can process non-printable data in the "text" data type, they **MAY** use the data type "text" for User-Password.

5.1.2. CHAP-Challenge

[RFC2865], Section 5.3 allows for the CHAP challenge to be taken from either the CHAP-Challenge attribute ([RFC2865], Section 5.40) or the Request Authenticator field. Since RADIUS/1.1 connections no longer use a Request Authenticator field, it is no longer possible to use the Request Authenticator field as the CHAP-Challenge when this transport profile is used.

Clients that send a CHAP-Password attribute ([RFC2865], Section 5.3) in an Access-Request packet over a RADIUS/1.1 connection **MUST** also include a CHAP-Challenge attribute ([RFC2865], Section 5.40).

Proxies may need to receive Access-Request packets over a non-RADIUS/1.1 transport and then forward those packets over a RADIUS/1.1 connection. In that case, if the received Access-Request packet contains a CHAP-Password attribute but no CHAP-Challenge attribute, the proxy **MUST** create a CHAP-Challenge attribute in the proxied packet using the contents from the incoming Request Authenticator of the received packet.

5.1.3. Tunnel-Password

The Tunnel-Password attribute ([RFC2868], Section 3.5) **MUST** be encoded the same as any other attribute of data type "string" that contains a tag, such as Tunnel-Client-Endpoint ([RFC2868], Section 3.3). Since the attribute is no longer obfuscated in RADIUS/1.1, there is no need for a Salt field or Data-Length fields as described in [RFC2868], Section 3.5. The textual value of the password can simply be encoded as is.

Note that the Tunnel-Password attribute is not of data type "text". The original reason in [RFC2868] was because the attribute was encoded as an opaque and obfuscated binary blob. We maintain that data type here, even though the attribute is no longer obfuscated. The contents of the Tunnel-Password attribute do not have to be printable text or UTF-8 data as per the definition of the "text" data type in [RFC8044], Section 3.4.

However, implementations should be aware that passwords are often printable text, and where the passwords are printable text, it can be useful to store and display them as printable text. Where implementations can process non-printable data in the "text" data type, they **MAY** use the data type "text" for Tunnel-Password.

5.1.4. Vendor-Specific Attributes

Any Vendor-Specific attribute that uses similar obfuscation **MUST** be encoded as per their base data type. Specifically, the MS-MPPE-Send-Key and MS-MPPE-Recv-Key attributes ([RFC2548], Section 2.4) **MUST** be encoded as any other attribute of data type "string" ([RFC8044], Section 3.4).

5.2. Message-Authenticator

The Message-Authenticator attribute ([RFC3579], Section 3.2) **MUST NOT** be sent over a RADIUS/1.1 connection. That attribute is not used or needed in RADIUS/1.1.

If the Message-Authenticator attribute is received over a RADIUS/1.1 connection, the attribute **MUST** be silently discarded or treated as an "invalid attribute", as defined in [RFC6929], Section 2.8. That is, the Message-Authenticator attribute is no longer used to authenticate packets for the RADIUS/1.1 transport. Its existence (or not) in this transport is meaningless.

A system that receives a Message-Authenticator attribute in a packet **MUST** treat it as an "invalid attribute" as defined in [RFC6929], Section 2.8. That is, the packet can still be processed, even if the Message-Authenticator attribute is ignored.

For proxies, the Message-Authenticator attribute has always been defined as being created and consumed on a "hop-by-hop" basis. That is, a proxy that received a Message-Authenticator attribute from a client would never forward that attribute as is to another server. Instead, the proxy would either suppress or recreate the Message-Authenticator attribute in the outgoing request. This existing behavior is leveraged in RADIUS/1.1 to suppress the use of the Message-Authenticator over a RADIUS/1.1 connection.

A proxy may receive an Access-Request packet over a RADIUS/1.1 connection and then forward that packet over a RADIUS/UDP or a RADIUS/TCP connection. In that situation, the proxy **SHOULD** add a Message-Authenticator attribute to every Access-Request packet that is sent over an insecure transport protocol.

The original text in [RFC3579], Section 3.3, Note 1 required that the Message-Authenticator attribute be present for certain Access-Request packets. It also required the use of the Message-Authenticator when the Access-Request packet contained an EAP-Message attribute. Experience has shown that some RADIUS clients never use the Message-Authenticator, even for the situations where its use is suggested.

When the Message-Authenticator attribute is missing from Access- Request packets, it is often possible to trivially forge or replay those packets. As such, it is **RECOMMENDED** that RADIUS clients always include Message-Authenticator in Access-Request packets when using UDP or TCP transport. As the scope of this document is limited to defining RADIUS/1.1, we cannot mandate that behavior here. Instead, we can note that there are no known negatives to this behavior, and there are definite positives, such as increased security.

Further issues related to Message-Authenticator are discussed in [DEPRECATE-RADIUS].

5.3. Message-Authentication-Code

Similarly, the Message-Authentication-Code attribute defined in [RFC6218], Section 3.3 **MUST NOT** be sent over a RADIUS/1.1 connection. If it is received in a packet, it **MUST** be treated as an "invalid attribute" as defined in [RFC6929], Section 2.8.

As the Message-Authentication-Code attribute is no longer used in RADIUS/1.1, the related MAC-Randomizer attribute ([RFC6218], Section 3.2) **MUST NOT** be sent over a RADIUS/1.1 connection. If it is received in a packet, it **MUST** be treated as an "invalid attribute" as defined in [RFC6929], Section 2.8.

5.4. CHAP, MS-CHAP, and Similar Attributes

While some attributes such as CHAP-Password depend on insecure cryptographic primitives such as MD5, these attributes are treated as opaque blobs when sent between a RADIUS client and server. The contents of the attributes are not obfuscated, and they do not depend on the RADIUS shared secret. As a result, these attributes are unchanged in RADIUS/1.1.

Similarly, MS-CHAP depends on MD4, and RADIUS/1.1 does not change the definition of any MS-CHAP attributes. However, MS-CHAP has been broken for decades, as noted in [ASLEAP]. The only appropriate use case for MS-CHAP is when it is protected by a secure transport such as RADIUS/TLS or RADIUS/DTLS, or when it is used for "inner tunnel" authentication methods as with the Protected Extensible Authentication Protocol (PEAP) or TTLS.

A server implementing this specification can proxy and authenticate CHAP, MS-CHAP, etc. without any issue. The RADIUS/1.1 protocol changes how RADIUS packets are authenticated and how "secret" data is obfuscated inside of a RADIUS packet. It does not change any authentication method that is transported inside of RADIUS.

5.5. Original-Packet-Code

[RFC7930], Section 4 defines an Original-Packet-Code attribute. This attribute is needed because otherwise it is impossible to correlate the Protocol-Error response packet with a particular request packet. The definition in [RFC7930], Section 4 describes the reasoning behind this need:

> The Original-Packet-Code contains the code from the request that generated the protocol error so that clients can disambiguate requests with different codes and the same ID.

This attribute is no longer needed in RADIUS/1.1. The Identifier field is unused, so it impossible for two requests to have the "same" ID. Instead, the Token field permits clients and servers to correlate requests and responses, independent of the Code value being used.

Therefore, the Original-Packet-Code attribute ([RFC7930], Section 4) **MUST NOT** be sent over a RADIUS/1.1 connection. If it is received in a packet, it **MUST** be treated as an "invalid attribute" as defined in [RFC6929], Section 2.8.

6. Other Considerations When Using ALPN

Most of the differences between RADIUS and RADIUS/1.1 are in the packet header and attribute handling, as discussed above. The remaining issues are a small set of unrelated topics, and are discussed here.

6.1. Protocol-Error

There are a number of situations where a RADIUS server is unable to respond to a request. One situation is where the server depends on a database, and the database is down. While arguably the server should close all incoming connections when it is unable to do anything, this action is not always effective. A client may aggressively try to open new connections or send packets to an unconnected UDP destination where the server is not listening. Another situation where the server is unable to respond is when the server is proxying packets, and the outbound connections are either full or failed.

In all RADIUS specifications prior to this one, there is no way for the server to send a client the positive signal that it received a request but is unable to send a response. Instead, the server usually just discards the request, which to the client is indistinguishable from the situation where the server is down. This failure case is made worse by the fact that perhaps some proxied packets succeed while others fail. The client can only conclude then that the server is randomly dropping packets and is unreliable.

It would be very useful for servers to signal to clients that they have received a request but are unable to process it. This specification uses the Protocol-Error packet ([RFC7930], Section 4) as that signal. The use of Protocol-Error allows for both hop-by-hop signaling in the case of proxy forwarding errors, and also for end-to-end signaling of server to client. Such signaling should greatly improve the robustness of the RADIUS protocol.

When a RADIUS/1.1 server determines that it is unable to process an Access-Request or Accounting-Request packet, it **MUST** respond with a Protocol-Error packet containing an Error-Cause attribute. A proxy that cannot forward the packet **MUST** respond with either 502 (Request Not Routable (Proxy)) or 505 (Other Proxy Processing Error). This requirement is to help distinguish failures in the proxy chain from failures at the final (i.e., home) server.

For a home server, if none of the Error-Cause values match the reason for the failure, then the value 506 (Resources Unavailable) **MUST** be used.

When a RADIUS proxy receives a Protocol-Error reply, it **MUST** examine the value of the Error-Cause attribute. If there is no Error-Cause attribute, or if its value is something other than 502 (Request Not Routable (Proxy)), 505 (Other Proxy Processing Error), or 506 (Resources Unavailable), then the proxy **MUST** return the Protocol-Error response packet to the client and include the Error-Cause attribute from the response it received. This process allows for full "end-to-end" signaling of servers to clients.

In all situations other than those outlined in the preceding paragraph, a client that receives a Protocol-Error reply **MUST** reprocess the original outgoing packet through the client forwarding algorithm. This requirement includes both clients that originate RADIUS traffic and proxies that see an Error-Cause attribute of 502 (Request Not Routable (Proxy)) or 505 (Other Proxy Processing Error).

The expected result of this processing is that the client forwards the packet to a different server. Clients **MUST NOT** forward the packet over the same connection and **SHOULD NOT** forward it over a different connection to the same server.

This process may continue over multiple connections and multiple servers, until the client either times out the request or fails to find a forwarding destination for the packet. A proxy that is unable to forward a packet **MUST** reply with a Protocol-Error packet containing an Error-Cause, as defined above. A client that originates packets **MUST** treat such a request as if it had received no response.

This behavior is intended to improve the stability of the RADIUS protocol by addressing issues first raised in [RFC3539], Section 2.8.

6.2. Status-Server

[RFC6613], Section 2.6.5, and by extension [RFC7360], suggest that the Identifier value zero (0) be reserved for use with Status-Server as an application-layer watchdog. This practice **MUST NOT** be used for RADIUS/1.1, as the Identifier field is not used in this transport profile.

The rationale for reserving one value of the Identifier field was the limited number of Identifiers available (256) and the overlap in Identifiers between Access-Request packets and Status-Server packets. If all 256 Identifier values had been used to send Access-Request packets, then there would be no Identifier value available for sending a Status-Server packet.

In contrast, the Token field allows for 2^{32} outstanding packets on one RADIUS/1.1 connection. If there is a need to send a Status-Server packet, it is nearly always possible to allocate a new value for the Token field. If instead there are 2^{32} outstanding packets for one connection, then it is likely that something has gone catastrophically wrong. In that case, the safest way forward is likely to just close the connection.

6.3. Proxies

A RADIUS proxy normally decodes and then re-encodes all attributes, including obfuscated ones. A RADIUS proxy will not generally rewrite the content of the attributes it proxies (unless site-local policy requires such a rewrite). While some attributes may be modified due to administrative or policy rules on the proxy, the proxy will generally not rewrite the contents of attributes such as User-Password, Tunnel-Password, CHAP-Password, MS-CHAP-Password, MS-MPPE keys, etc. Therefore, all attributes are transported through a RADIUS/1.1 connection without changing their values or contents.

A proxy may negotiate RADIUS/1.1 (or not) with a particular client or clients, and it may negotiate RADIUS/1.1 (or not) with a server or servers it connects to, in any combination. As a result, this specification is fully compatible with all past, present, and future RADIUS attributes.

7. Other RADIUS Considerations

This section discusses issues in RADIUS that need to be addressed in order to support ALPN, but which aren't directly part of the RADIUS/1.1 protocol.

7.1. Crypto-Agility

The crypto-agility requirements of [RFC6421] are addressed in [RFC6614], Appendix C and in [RFC7360], Section 10.1. This specification makes no changes or additions to those specifications. The use of ALPN and the removal of MD5 has no impact on the security or privacy of the protocol.

RADIUS/TLS has been widely deployed in at least eduroam ([RFC7593] and [EDUROAM]) and in OpenRoaming [OPENROAMING]. RADIUS/DTLS has seen less adoption, but it is known to be supported in many RADIUS clients and servers.

It is **RECOMMENDED** that all implementations of historic RADIUS/TLS be updated to support this specification. Where a system already implements RADIUS over TLS, the additional effort to implement this specification is minimal. Once implementations support it, administrators can gain the benefit of it with little or no configuration changes. This specification is backwards compatible with [RFC6614] and [RFC7360]. It is only potentially subject to down-bidding attacks if implementations do not enforce ALPN correctly on session resumption.

All crypto-agility needed or used by this specification is implemented in TLS. This specification also removes all cryptographic primitives from the application-layer protocol (RADIUS) being transported by TLS. As discussed in the following section, this specification also bans the development of all new cryptographic or crypto-agility methods in the RADIUS protocol.

7.2. Error-Cause Attribute

The Error-Cause attribute is defined in [RFC5176]. The "Table of Attributes" section given in [RFC5176], Section 3.6 permits that attribute to appear in CoA-NAK and Disconnect-NAK packets. As no other packet type is listed, the implication is that the Error-Cause attribute cannot appear in any other packet. [RFC7930] also permits Error-Cause to appear in Protocol-Error packets.

However, [RFC5080], Section 2.6.1 suggests that Error-Cause may appear in Access-Reject packets. No explanation is given for this change from [RFC5176]. There is not even an acknowledgment that this suggestion is a change from any previous specification. We correct that issue here.

This specification updates [RFC5176] to allow the Error-Cause attribute to appear in Access-Reject packets. It is **RECOMMENDED** that implementations include the Error-Cause attribute in Access-Reject packets where appropriate.

That is, the reason for sending the Access-Reject packet (or the Protocol-Error packet) may match a defined Error-Cause value. In that case, it is useful for implementations to send an Error-Cause attribute with that value. This behavior can help RADIUS system administrators debug issues in complex proxy chains.

For example, a proxy may normally forward Access-Request packets that contain EAP-Message attributes. The proxy can determine if the contents of the EAP-Message are invalid. One example of an invalid EAP-Message is where the first octet has value larger than 4. In that case, there may be no benefit to forwarding the packet, as the home server will reject it. It may then be possible for the proxy (with the knowledge and consent of involved parties) to immediately reply with an Access-Reject containing an Error-Cause attribute with value 202 (Invalid EAP Packet (Ignored)).

Another possibility is that a proxy is configured to forward packets for a particular realm, but it has determined that there are no available connections to the next hop for that realm. In that case, it may be possible for the proxy (again, with the knowledge and consent of involved parties) to reply with an Access-Reject containing an Error-Cause attribute with value 502 (Request Not Routable (Proxy)).

These examples are given only for illustrative and informational purposes. While it is useful to return an informative value for the Error-Cause attribute, proxies can only modify the traffic they forward with the explicit knowledge and consent of all involved parties.

7.3. Future Standards

Future work may define new attributes, packet types, etc. It is important to be able to do such work without requiring that every new standard mention RADIUS/1.1 explicitly. This document defines RADIUS/1.1 as having functional overlap with legacy RADIUS: the protocol state machine is unchanged, the packet header Code field is unchanged, and the attribute format is largely unchanged. As a result, any new packet Code or attribute defined for RADIUS is explicitly compatible with RADIUS/1.1; the field contents and meanings are identical. The only difference between the two protocols is that obfuscated attributes in RADIUS are not obfuscated in RADIUS/1.1, and this document defines how that mapping is done.

Any future specification only needs to mention RADIUS/1.1 if it adds fields to the RADIUS/1.1 packet header. Otherwise, transport considerations for RADIUS/1.1 are identical to RADIUS over (D)TLS.

We reiterate that this specification defines a new transport profile for RADIUS. It does not define a completely new protocol. Any future specification that defines a new attribute **MUST** define it for RADIUS/UDP first, and afterwards those definitions can be applied to this transport profile.

New specifications **MAY** define new attributes that use the obfuscation methods for User-Password as defined in [RFC2865], Section 5.2 or for Tunnel-Password as defined in [RFC2868], Section 3.5. There is no need for those specifications to describe how those new attributes are transported in RADIUS/1.1. Since RADIUS/1.1 does not use MD5, any obfuscated attributes will by definition be transported as their underlying data type "text" ([RFC8044], Section 3.4) or "string" ([RFC8044], Section 3.5).

New RADIUS specifications **MUST NOT** define attributes that can only be transported via RADIUS over TLS. The RADIUS protocol has no way to signal the security requirements of individual attributes. Any existing implementation will handle these new attributes as "invalid attributes" ([RFC6929], Section 2.8) and could forward them over an insecure link. As RADIUS security and signaling is hop-by-hop, there is no way for a RADIUS client or server to even know if such forwarding is taking place. For these reasons and more, it is therefore inappropriate to define new attributes that are only secure if they use a secure transport layer.

The result is that specifications do not need to mention this transport profile or make any special provisions for dealing with it. This specification defines how RADIUS packet encoding, decoding, authentication, and verification are performed when using RADIUS/1.1. So long as any future specification uses the existing schemes for encoding, decoding, etc., that are defined for RADIUS, no additional text in future documents is necessary in order to be compatible with RADIUS/1.1.

We note that it is theoretically possible for future standards to define new cryptographic primitives for use with RADIUS/UDP. In that case, those documents would likely have to describe how to transport that data in RADIUS/1.1. We believe that such standards are unlikely to be published, as other efforts in the RADEXT Working Group are forbidding such updates to RADIUS.

8. Privacy Considerations

This specification requires secure transport for RADIUS. RADIUS/1.1 has all of the privacy benefits of RADIUS/TLS [RFC6614] and RADIUS/DTLS [RFC7360] and none of the privacy or security issues of RADIUS/UDP [RFC2865] or RADIUS/TCP [RFC6613].

9. Security Considerations

The primary focus of this document is addressing security considerations for RADIUS. This specification relies on TLS and associated ALPN for much of its security. We refer the reader to [RFC8446] and [RFC7360] for discussions of the security of those protocols. The discussion in this section is limited to issues unique to this specification.

Implementations should rely on the underlying TLS library to perform ALPN version negotiation. That is, implementations should supply a list of permitted ALPN strings to the TLS library, and let it return the negotiated value.

There are few other opportunities for security issues. If an implementation gets ALPN wrong, then the wrong application data will be transported inside of TLS. While RADIUS/1.0 and RADIUS/1.1 share similar packet formats, the protocols are not mutually compatible.

When an implementation receives the packets for a RADIUS version that is not supported by this connection, it will not be able to process the packets. Implementations can produce log messages indicating that the application-layer data is unexpected and close the connection. In addition, the implementations that see the incorrect application data already have full access to all secrets, passwords, etc. being transported, so any protocol differences do not result in any security issues. Since all of the application data is protected by TLS, there is no possibility for an attacker to obtain any extra data as a result of this misconfiguration.

RADIUS/1.0 requests sent over a RADIUS/1.1 connection may be accepted by the RADIUS/1.1 server, as the server will ignore the ID field and try to use portions of the Request Authenticator as a Token. However, the reply from the RADIUS/1.1 server will fail the Response Authenticator validation by the RADIUS/1.0 client. Therefore, the responses will be dropped. The client will generally log these failures, and an administrator will address the issue.

RADIUS/1.1 requests sent over a RADIUS/1.0 connection will generally be discarded by the RADIUS/1.0 server, as the packets will fail the Request Authenticator checks. That is, all request packets such as Accounting-Request, CoA-Request, and Disconnect-Request will be discarded by the server. For Access-Request packets containing EAP-Message, the packets will be missing Message-Authenticator and will therefore be discarded by the server. Other Access-Request packets that contain obfuscated attributes such as User-Password will have those attributes decoded to nonsense, thus resulting in Access-Reject responses.

RADIUS/1.1 Access-Request packets containing non-obfuscated attributes such as CHAP-Password may be accepted by a RADIUS/1.0 server, but the response will contain a Response Authenticator (i.e., MD5 hash) and not a Token that matches the Token in the request. A similar analysis applies for Access-Request packets containing Service-Type = Authorize-Only.

In conclusion, any mismatch of versions between client and server will result in most request packets being discarded by the server and all response packets being discarded by the client. Therefore, the two protocols are incompatible and safe from misconfigurations or erroneous implementations.

10. IANA Considerations

IANA has updated the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry with two new entries:

Protocol:	RADIUS/1.0
Identification Sequence:	0x72 0x61 0x64 0x69 0x75 0x73 0x2f 0x31 0x2e 0x30 ("radius/1.0")
Reference:	RFC 9765

Protocol:	RADIUS/1.1
Identification Sequence:	0x72 0x61 0x64 0x69 0x75 0x73 0x2f 0x31 0x2e 0x31 ("radius/1.1")
Reference:	RFC 9765

11. References

11.1. Normative References

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

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

[RFC6421] Nelson, D., Ed., "Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS)", RFC 6421, DOI 10.17487/RFC6421, November 2011, <https://www.rfc-editor.org/info/rfc6421>.

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

[RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, DOI 10.17487/RFC6929, April 2013, <https://www.rfc-editor.org/info/rfc6929>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

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

[RFC8044] DeKok, A., "Data Types in RADIUS", RFC 8044, DOI 10.17487/RFC8044, January 2017, <https://www.rfc-editor.org/info/rfc8044>.

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

11.2. Informative References

[ASLEAP] "asleap - recovers weak LEAP and PPTP passwords", commit 254acab, November 2020, <https://github.com/joswr1ght/asleap>.

[DEPRECATE-RADIUS] DeKok, A., "Deprecating Insecure Practices in RADIUS", Work in Progress, Internet-Draft, draft-ietf-radext-deprecating-radius-05, 26 November 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-radext-deprecating-radius-05>.

[EDUROAM] eduroam, "eduroam", <https://eduroam.org>.

[FIPS-140-3] NIST, "Security Requirements for Cryptographic Modules", NIST FIPS 140-3, DOI 10.6028/NIST.FIPS.140-3, March 2019, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf>.

[OPENROAMING] Wireless Broadband Alliance, "OpenRoaming: One global Wi-Fi network", <https://wballiance.com/openroaming/>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[RFC2548] Zorn, G., "Microsoft Vendor-specific RADIUS Attributes", RFC 2548, DOI 10.17487/RFC2548, March 1999, <https://www.rfc-editor.org/info/rfc2548>.

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

[RFC2868] Zorn, G., Leifer, D., Rubens, A., Shriver, J., Holdrege, M., and I. Goyret, "RADIUS Attributes for Tunnel Protocol Support", RFC 2868, DOI 10.17487/RFC2868, June 2000, <https://www.rfc-editor.org/info/rfc2868>.

[RFC3539] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, DOI 10.17487/RFC3539, June 2003, <https://www.rfc-editor.org/info/rfc3539>.

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

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

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

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

[RFC5281] Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, DOI 10.17487/RFC5281, August 2008, <https://www.rfc-editor.org/info/rfc5281>.

[RFC5931] Harkins, D. and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", RFC 5931, DOI 10.17487/RFC5931, August 2010, <https://www.rfc-editor.org/info/rfc5931>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6218] Zorn, G., Zhang, T., Walker, J., and J. Salowey, "Cisco Vendor-Specific RADIUS Attributes for the Delivery of Keying Material", RFC 6218, DOI 10.17487/RFC6218, April 2011, <https://www.rfc-editor.org/info/rfc6218>.

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

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

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

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

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

Acknowledgments

Thanks to Bernard Aboba, Karri Huhtanen, Heikki Vatiainen, Alexander Clouter, Michael Richardson, Hannes Tschofenig, Matthew Newton, and Josh Howlett for reviews and feedback.

Author's Address

Alan DeKok
FreeRADIUS
Email: aland@freeradius.org