RADEXT Working Group
Internet-Draft
Updates: 2865, 2866, 5176, 6613, 6614, 7360, 7930 (if approved)
Intended status: Best Current Practice
Expires: 26 December 2025

A. DeKok
InkBridge Networks

24 June 2025

Standardising Protocol-Error
draft-dekok-protocol-error-00

Abstract

We extend and standardise the Protocol-Error packet Code, first defined in RFC 7930 for the Remote Authentication Dial In User Service (RADIUS) protocol.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-dekok-protocol-error/.

Discussion of this document takes place on the RADEXT Working Group mailing list (mailto:radext@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/radext/. Subscribe at https://www.ietf.org/mailman/listinfo/radext/.

Source for this draft and an issue tracker can be found at https://github.com/freeradius/protocol-error.git.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 December 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

The Remote Authentication Dial In User Service (RADIUS) protocol is designed as a request / response protocol, where clients send requests to servers, and servers send responses to clients. There are a few different types of requests defined, and each type of request has one or more associated valid responses. However, these requests and responses are limited to indicating success or failure of the requested action, such as authentication or accounting.

There is no way for clients and servers to exchange information about their state, or their inability to perform the requested action. Instead, for actions such as authentication, servers can only reply with accept, reject, or simply fail to respond to the request. There is no way for the server to respond to a client with an error indicating "invalid request" or "unable to answer this request".

The result is that when the server experiences a failure, it either stops responding to requests, or returns a type-specific response which is arguably wrong. Depending on the kind of failure, the server may stop responding to only one request, or else the server could stop responding to all requests. The client then has to make sense of the situation where some requests get a positive acknowledgement (ACK) response, other requests get negative acknowledgement (NAK) response, and other requests simply never see a response.

This failure for the server to respond meaningfully means that the client has no way of knowing if the server is slow, is off line, or else if the server is a proxy and one of the next hops has failed. The only course of action that the client can take is to guess as what it should do when the server fails to respond. Such guesses are likely to be wrong. Incorrect decisions based on incomplete knowledge contribute to network instability and to congestive collapse.

This failure of RADIUS to provide for protocol level signaling has caused issues with deployments for decades. The network as a whole operates not because the underlying protocol is robust, but instead because of local workarounds built by implementers and operators. It is time to address this deficiency.

This document standardises the Protocol-Error packet code, which was first defined as an experimental code in [RFC7930], Section 4. This packet code allows for servers to reply to any request with a Protocol-Error packet, which serves as an explicit protocol layer signal that the server has received a request, but is unable to process it. The Protocol-Error packet contains an Error-Cause

attribute ([RFC3576], Section 3.1) which indicates whether the error is limited to just this packet or to the connection as a whole. The Error-Cause attribute also indicates whether this error is temporary or permanent.

Explicitly signaling errors for protocol-layer failures increases the stability of the network, and decreases the likelihood of congestive collapse on failure.

Where [RFC7930] gives a very short description of the Protocol-Error packet Code, this document gives a more detailed definition. It explains the reasons for standardising Protocol-Error, fully defines the behavior of clients and servers with respect to Protocol-Error, and gives suggestions for migrations paths for implementations and deployments.

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.

3. Review and Motivation

This specification defines an extension to the RADIUS protocol which makes significant changes to how the protocol operates in the event of a failure. In order to motivate this change, we need to describe the historic issues with the protocol. We describe how the request / response nature of RADIUS is not, in fact, fully met: only some packet types have a NAK response defined, and there are other cases where the existing standards require that valid requests do not get any response at all.

The result is that the RADIUS protocol is missing an important piece: signaling of protocol-level errors. This section explains the numerous situations where these errors are not signaled, and what impact this problem has for running networks.

3.1. Request / Response Packets

This section reviews what types of request and responses are defined in RADIUS, along with their behavior.

The Access-Request packet Code is defined in [RFC2865], Section 4.1, with Access-Accept ([RFC2865], Section 4.2) as an ACK indicating that authentication has succeeded, and Access-Reject ([RFC2865], Section 4.3) as an explicit NAK that authentication has failed.

The Accounting-Request packet Code is defined in [RFC2866], Section 4.1, with Accounting-Response defined in [RFC2866], Section 4.2, as an ACK that the accounting data has been stored. There is, however, no corresponding NAK which indicates that accounting has failed. Instead, [RFC2866], Section 4.1 states:

> Upon receipt of an Accounting-Request, the server MUST transmit an Accounting-Response reply if it successfully records the accounting packet, and MUST NOT transmit any reply if it fails to record the accounting packet.

Failing to acknowledge a request with a response is, at best, impolite. More practically, it causes problems with networks. These problems will be discussed in more detail later in this document.

The CoA-Request packet code is defined in [RFC5176], Section 2.2, along with CoA-ACK to indicate that the change was successful, and CoA-NAK as an acknowledgement that the change of authorization action could not be performed.

The Disconnect-Request packet code is defined in [RFC5176], Section 2.1, along with Disconnect-ACK to indicate that the disconnection was successful, and Disconnect-NAK as an acknowledgement that the user or device could not be disconnected.

While most requests have an ACK and NAK response defined, Accounting-Request packets only have an ACK response defined. If a request does not receive a response, clients retransmit as per[RFC5080], Section 2.2.1. For Accounting-Request packets, these retransmits are unlikely to yield a positive response.

Even when the request has a NAK response defined, there are still situations where a valid request could never have a response returned. Instead, there are many cases where a server is suggested or even required to discard requests, and therefore never send a response.

3.2. Silently Discard

The existing RADIUS specifications require that a server "silently discard" requests without a response for many reasons. In some cases, this discard process is necessary for security reasons, such as when the Message-Authenticator ([RFC2869], Section 5.14 fails verification. Discarding packets for security reasons is necessary, and this specification does not change that behavior.

In other cases, this "silently discard" process is done for reasons which are not related to security. For example, a server could discard a request which contains data that the server does not understand, but which also does not affect the requested action. Or, the server could discard a request which has been authenticated to be well-formed, but which contains unexpected protocol state.

In an ideal world, the server would instead either ignore only the portions of the packet which it could not understand, or else the server would respond to the client with an indication that the request was improper. A server could also respond to a request containing invalid protocol state, and indicate that the request was authentic and well-formed, but also unexpected.

These issues are grouped under the term "silently discard", which is defined in [RFC2865], Section 1.2 as:

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

While there is not a lot of quantitative data publicly available, these silent discards are colloquially known to be a large source of instability in RADIUS deployments. There has been substantial work done by server implementations and operators of proxy networks to stablize the network by attempting to determine whether or not the lack of a response indicates an actual failure, or only an illusory one due to the limitations of the RADIUS protocol.

One ad-hoc workaround to failures involves increasing the number of Status-Server checks [RFC5997], in violation of the [RFC2865], Section 2.6 suggestion of "Keep-Alives Considered Harmful", which we quote in full here:

> Some implementers have adopted the practice of sending test RADIUS requests to see if a server is alive. This practice is strongly discouraged, since it adds to load and harms scalability without

> providing any additional useful information. Since a RADIUS request is contained in a single datagram, in the time it would take you to send a ping you could just send the RADIUS request, and getting a reply tells you that the RADIUS server is up. If you do not have a RADIUS request to send, it does not matter if the server is up or not, because you are not using it.
>
> If you want to monitor your RADIUS server, use SNMP. That's what SNMP is for.

Experience has shown that this analysis is incomplete, as it does not address the situation where a server is responding to only a subset of requests. This position is perhaps implied in [RFC5997], Section 2, which says:

> The Status-Server packet is not a "Keep-Alive" as discussed in [RFC2865], Section 2.6. "Keep-Alives" are Access-Request packets sent to determine whether a downstream server is responsive. These packets are typically sent only when a server is suspected to be down, and they are no longer sent as soon as the server is available again.

In some cases, there is a sufficiently low volume of packets that the inter-packet spacing is large enough which can trigger these Status-Server checks. In other cases, implementations will send Status-Server when some requests do not get a response, even if the server is actively responding to other requests.

Some implementations and deployments have chosen to continuously send Status-Server requests, in contradiction to the recommendations of [RFC2865], Section 2.6. Feedback from those systems indicate that this practice increases the information available to administrators, and decreases operational outages.

The rest of this section describes in more detail the numerous situations where requests are silently discarded.

3.2.1. Unknown Attributes

Some implementations discard requests when the request contains attributes which the implementation does not recognize. The behavior of servers with respect to unknown attributes is defined by [RFC2865], Section 5 which says that implementations:

> MAY ignore Attributes with an unknown Type.

[RFC6929], Section 2.8 makes a stronger suggestion that:

> It is RECOMMENDED that Attributes with unknown Type, Extended-Type, TLV-Type, or EVS-Type are treated as "invalid attributes".

Despite this recommendation to ignore these unknown attributes, some implementations will instead silently discard the entire request. This behavior is not standards compliant, and is not allowed by any interpretation of the text in [RFC2865], Section 5.

For example, the BlastRADIUS mitigations (TBD) were negatively affected by some implementations which would discard packets which contain a Message-Authenticator attribute. This behavior was seen in 2024, decades after the attribute was defined in [RFC2869], Section 5.12 (2000). This failure to handle unknown attributes in a reasonable manner made it difficult and expensive for operators to secure their networks via the recommended BlastRADIUS mitigations.

3.2.2. Invalid Attributes

[RFC6929], Section 2.8 defines "invalid attributes" as attributes where the Type is known, but the Value is malformed. This topic was not discussed in the original RADIUS specifications, which led to a wide range of implementation-specific behavior, including ones where the packet would be silently discarded.

The behavior of implementations with respect to invalid attributes was clarified in [RFC6929], Section 2.8, which requires the following behavior:

> The existence of an "invalid attribute" in a packet or attribute MUST NOT result in the implementation discarding the entire packet ...

Even though this specification being over a decade old, some implementations still do not follow this requirement. Instead, some implementations will silently discard the entire request. This behavior has caused wide-spread issues in proxy networks, most notably with Operator-Name [RFC5580], Section 4.1.

One widely used implementation had erroneously created a local definition for attribute 126, whih was then in conflict with Operator-Name when that attribute was defined. The implemenation also discarded requests which contained invalid attributes. The combination of the two behaviors led to wide-spread outages and significant effort by operators to perform a root-cause analyis and then to correct the issue.

This behavior violates the robustness principle on which the Internet was founded.

3.2.3. Unknown State

In some cases, as server can receive a request which contains a State attribute with an unknown value. This situation can occur, for example, when EAP [RFC3579] authentications are sent through a load balancer, and one of the final home servers fails. The load balancer does not necessarily know anything about the underlying EAP session, and therefore may redistribute requests for an EAP session to a server which is unaware of that session.

The result is that a home server can receive a request which is in the middle of an EAP session. Since the server recieving the request did not start the EAP session, it does not recognize the value of the State attribute, and is unable to process the request.

Some implementations will response with an Access-Reject. Others will silently discard the request. Both behaviors are arguably wrong.

3.3. Unexpected Request Code

[RFC2865] and [RFC2866] defined different ports for Access-Request and Accounting-Request packets. [RFC5176] defines a different port entirely for both CoA-Request and Disconnect-Request packets. In contrast, [RFC6613], [RFC6614], and [RFC7360] define one port for all request Codes. Some implementations have also chosen to accept multiple request Codes on one port, even for RADIUS/UDP.

As more than one port is used for RADIUS, there is a potential for mismatched configuration between a client and server. Each RADIUS specification ([RFC2865], Section 3, [RFC2866], Section 3, and [RFC5176], Section 2.3) has the same text related to receiving packets:

> The Code field is one octet, and identifies the type of RADIUS packet. Packets received with an invalid Code field MUST be silently discarded.

However, this text does not address the issue where a client sends a request Code which is correctly authenticated, but which the server does not expect to receive. It is reasonable for the server to not perform the requested action. It is not reasonable, however, for the server to simply fail to respond to authentic requests.

3.3.1. Accounting Failure

[RFC2866], Section 4.1 says that servers need to respond to Accounting-Request packets when the accounting action is successful, but not when the accounting action fails:

> Upon receipt of an Accounting-Request, the server MUST transmit an Accounting-Response reply if it successfully records the accounting packet, and MUST NOT transmit any reply if it fails to record the accounting packet.

In other words, servers are mandated to silently discard authentic packets.

3.3.2. Network Loss and Fragmentation

When RADIUS/UDP is used, it is possible for either valid requests or responses to be lost in the network. RADIUS/UDP has another issue noted in [RFC3579], Section 2.4, which notes that packets can be fragmented. [RFC6613], Section 1 also notes that:

> Transport of fragmented UDP packets appears to be a poorly tested code path on network devices. Some devices appear to be incapable of transporting fragmented UDP packets, making it difficult to deploy RADIUS in a network where those devices are deployed.

RADIUS/TCP and RADIUS/TLS are only partial solutions to the above problems. While they solve the fragmentation issue, there is still the issue where a server can receive a request, and then the connection can drop before the response is sent. From the clients point of view then, the request was sent, and then was silently discarded by the server.

3.3.3. Proxy Chains

Even when none of the above problems are seen, a responsive and correctly configured server could still fail to respond when it is acting as a proxy. A proxy is conceptually just a server which then acts as a client to another server. The client side of the proxy can therefore run into the above issues when the next hop in the proxy chain fails to respond to a request.

When the next hop fails to respond to the proxied request, there is very little positive action that the proxy can take. It can perhaps send the request over a different connection, or to a different server, but that step is not always possible. In the end, the proxy has to decide where or not to originate an answer itself (e.g. Access-Reject, CoA-NAK, etc.), or whether to just never respond to the client.

4. Impact

Thare a number of impacts which result from NAK responses which do not distinguish between different types of failures, and from request simply disappearing into the network without response.

It may seem useful for a server to NAK an action which it cannot perform, but this response is not appropriate for a number of reasons. The problem could be a due to a failures of user credential being incorrect, failures of policy, or it could be another failure which is unrelated to the action being requested. As such, a user-level NAK response to a network-level failure is inappropriate, for a number of reasons.

For example, if a user is attempting to access the network, an Access-Reject is interpreted as indicating that the users credentials are invalid. This indication could result in the user device taking action, such as invalidating or discarding the users password. The user device could even present the user with a prompt which indicates that the password is incorrect, and that the user should re-enter it. Such actions are likely to confuse the user and to cause problems.

For accounting, the situation is made worse by the behavior of Acct-Delay-Time ([RFC2866], Section 5.2). When a client retransmits an Accounting-Request packet, the Acct-Delay-Time attribute is updated, which negates the benefits of the [RFC5080], Section 2.2.2 packet deduplication cache. The client can then send multiple new packets which contribute to congestive collapse of the network.

The impact for CoA-Request and Disconnect-Request packets is similar: either a NAK is sent back incorrectly, or the request never sees a response.

The lack of responses also contributes to network instability. When a client sends a request which never yields a response, it can only assume that the server is down. The client may then fail over to a different server, or open new connections to the same server, or simply stop sending packets to the server.

5. Protocol-Error Code

This document make the RADIUS packet Code 52 (Protocol-Error) standard instead of experimental, as an update to [RFC7930]. In order to have one document defining this RADIUS packet Code, the definition of Protocol-Error is repeated here with only minor edits from the original in [RFC7930], Section 4.

The Protocol-Error packet Code may be used in response to any request packet, such as Access-Request, Accounting-Request, CoA-Request, or Disconnect-Request. It is a response packet sent by a server to a client. The packet indicates to the client that the server is unable to process the request for some reason.

A Protocol-Error packet MUST contain an Original-Packet-Code attribute, along with an Error-Cause attribute. Other attributes MAY be included if desired. 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. Regardless of the original packet code, the RADIUS Server calculates the Message-Authenticator attribute as if the original packet were an Access-Request packet. The identifier is copied from the original request.

Clients processing Protocol-Error MUST ignore unknown or unexpected attributes.

This RADIUS packet Code is hop by hop. Proxies MUST NOT simply forward a Protocol-Error response that they receive. Instead, a proxy MUST examine the Error-Cause attribute to determine whether the error is due to a fault at the home server, or if it is a fault in the RADIUS proxy network.

Where the Error-Cause value indicates a fault in the RADIUS proxy network, the proxy which receives the Protocol-Error SHOULD try to proxy the request to a different destination, or over a different connection to the same destination. Futher handling of proxy fail-over is out of scope of this specification.

The following values for Error-Cause indicate a fault in the RADIUS proxy network. Other values for Error-Cause are possible, and may be defined in a future specification:

* 502, Request Not Routable (Proxy)

* 505, Other Proxy Processing Error

The remainder of the Error-Cause values SHOULD be interpreted as originating at the home server, and the proxy SHOULD return the Protocol-Error to the client, and include the Error-Cause attribute from the response to the Proxied Request

5.1. Impact on existing systems

TBD - clients ignore packet Codes they don’t understand. Therefore Protocol-Error is always safe to send. Even if the client does not process it and take action, an administrator can see the packet and take action.

5.2. Negotiation and Configuration

TBD - no negotiation, but configuration based on agreement of administrators of client and servers.

TBD - if a proxy knows that a client does not understand Protocol-Error, it MAY instead respond to the client with a relevant NAK packet. This SHOULD NOT be done when the client is another proxy, but only when the client is a NAS.

5.3. When to send Protocol-Error Responses

TBD - mainly authentic packets, where there is no NAK (Accounting-Request), or where the NAK is not appropriate (unknown EAP State, etc.)

See the (TBD Solutions) section below for more details.

6. Original-Packet-Code Attribute

This document standardizes the Original-Packet-Code attribute, which was originally defined in [RFC7930].

Description

The Original-Packet-Code contains the packet Code from the request that generated the Protocol-Error response. It allows clients to disambiguate requests with different packet Codes which use the same ID.

The Original-Packet-Code attribute MUST NOT be sent in any packet Code other than Protocol-Error.

A client which receives a Protocol-Error in a packet Code other than Protocol-Error MUST treat it as an "invalid attribute" as per [RFC6929], Section 2.8.

Type

241.4

Length

6

Data Type

integer

Value

The Value is taken from the original request packet Code which was received by the server. Values which are not in the the IANA "RADIUS Packet Type Codes" registry are likely to be invalid.

A client which receives an Original-Packet-Code attribute that does not match any outstanding request MUST silently discard the response.

7. Solutions to Problems

This section describes the changes to RADIUS which address the above problems.

7.1. Invalid Attributes

We reiterate here the mandate of [RFC6929], Section 2.8:

> The existence of an "invalid attribute" in a packet or attribute MUST NOT result in the implementation discarding the entire packet ...

Implementations which discard packets due to invalid attributes are not standards compliant.

7.2. Unknown Attributes

Some implementations discard requests when the request contains attributes which the implementation does not recognize. This behavior is permitted by [RFC2865], Section 5 which says that implementations:

> MAY ignore Attributes with an unknown Type.

[RFC6929], Section 2.8 makes a stronger suggestion that:

> It is RECOMMENDED that Attributes with unknown Type, Extended-Type, TLV-Type, or EVS-Type are treated as "invalid attributes".

We update [RFC2865], [RFC2866], [RFC5176] and [RFC6929] here to say that implementations MUST NOT silently discard packets which contain attributes of an known Type, Extended-Type, TLV-Type, or EVS-Type. This mandate applies to all RADIUS packets, for all values of the packet Code field.

Note that this prohibition does not apply to known attributes which have unknown values. For example, [RFC2865], Section 5.6 says that for the Service-Type attribute:

> A NAS is not required to implement all of these service types, and MUST treat unknown or unsupported Service-Types as though an Access-Reject had been received instead.

That requirement is not changed by this specification.

7.3. Unknown State

TBD - try not to send Access-Reject. Maybe recommend a new Error-Cause value?

8. IANA Considerations

IANA is instructed to update the "RADIUS Packet Type Codes" registry, and change the Protocol-Error entry from Reference RFC7930 to THIS-DOCUMENT.

IANA is instructed to update the "RADIUS Attribute Types" registry, and change the Original-Packet-Code entry from Reference RFC7930 to THIS-DOCUMENT.

9. Privacy Considerations

There are no privacy considerations in this specification. It does not add or change any transport protocols, and it does not change the roles of any participant in the RADIUS ecosystem.

10. Security Considerations

This specification has no direct impact on security. It does not add or change any existing cryprography in RADIUS. It is compatible with all RADIUS standards, including (ALPN draft).

This specification does, however, recommend ways that networks can be better protected from a variety of problems, misconfigurations, and erroneous implementations. As a result, RADIUS proxy networks which implement this specification should become more stable, and therefore more secure.

11. Acknowledgements

This document came out of discussions both in the RADEXT WG, and in the 2025 RADIUS Technical Conference in Tampere, Finland.

12. References

12.1. 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/rfc/rfc2119>.

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

[RFC3576] Chiba, M., Dommety, G., Eklund, M., Mitton, D., and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 3576, DOI 10.17487/RFC3576, July 2003, <https://www.rfc-editor.org/rfc/rfc3576>.

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

[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/rfc/rfc6929>.

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

12.2. Informative References

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

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

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

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

[RFC5580] Tschofenig, H., Ed., Adrangi, F., Jones, M., Lior, A., and B. Aboba, "Carrying Location Objects in RADIUS and Diameter", RFC 5580, DOI 10.17487/RFC5580, August 2009, <https://www.rfc-editor.org/rfc/rfc5580>.

[RFC5997] DeKok, A., "Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol", RFC 5997, DOI 10.17487/RFC5997, August 2010, <https://www.rfc-editor.org/rfc/rfc5997>.

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

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

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

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

Author’s Address

Alan DeKok
InkBridge Networks
Email: aland@inkbridgenetworks.com