

RADEXT Working Group
Internet-Draft
Updates: 2865, 2866 (if approved)
Intended status: Best Current Practice
Expires: 24 December 2025

A. DeKok
InkBridge Networks
22 June 2025

Proxy Best Practices for the Remote Authentication Dial In User Service
(RADIUS) Protocol
draft-dekok-proxy-bcp-00

Abstract

RADIUS proxying is terrible. We try to make it better. This document is currently a rough draft, and is a work in progress.

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-proxy-bcp/>.

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/proxy-bcp.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 24 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	3
1.1. Proxy versus Client or Server	4
2. Terminology	5
3. Overview of Proxy Architecture and Use Cases	5
3.1. Real World Impact	5
4. Problems and Solutions	5
4.1. Silently Discard	5
4.1.1. Problem	6
4.1.2. Solution	6
4.2. Access-Reject	6
4.2.1. Problem	6
4.2.2. Solution	7
4.3. Retransmission	7
4.3.1. Problem	7
4.3.2. Solution	7
4.4. Acct-Delay-Time and Congestive Collapse	7
4.4.1. Problem	8
4.4.2. Solution	8
4.5. There is no RADIUS Routing Protocol	8
4.5.1. Problem	8
4.5.2. Solution	8
4.6. Filtering	8
4.6.1. Problem	8
4.6.2. Solution	9
4.7. Inadequate signal of errors	9
4.7.1. Problem	9
4.7.2. Solution	9
4.8. Clients misbehave with Delayed Rejects	9
4.8.1. Problem	10
4.8.2. Solution	10
4.9. Timeouts vary wildly across a proxy chain	10
4.9.1. Problem	10

4.9.2. Solution	10
5. Best Current Practices	10
5.1. Proxy-State	10
5.2. Rate limiting	11
5.3. Load Balancing	12
5.3.1. Home Servers	12
5.4. Differing rates on inbound and outbound	13
5.5. Accounting Store and Forward	13
5.6. Protection from DoS attacks	14
5.7. Don't be AntiSocial	14
6. IANA Considerations	14
7. Privacy Considerations	14
8. Security Considerations	15
9. Acknowledgements	15
10. References	15
10.1. Normative References	15
10.2. Informative References	15
Contributors	16
Author's Address	16

1. Introduction

The Remote Authentication Dial In User Service (RADIUS) protocol defines proxying in [RFC2865], Section 2.3. That section gives a brief overview of how proxying on a packet-by-packet basis, but it does not discuss larger issues of how a proxy server should operate. The discussion of those topics is largely limited to a few sentences in [RFC2865], Section 2.3:

A forwarding server MAY need to modify attributes to enforce local policy. Such policy is outside the scope of this document ...

Implementers of forwarding servers should consider carefully which values it is willing to accept for Service-Type. Careful consideration must be given to the effects of passing along Service-Types of NAS-Prompt or Administrative in a proxied Access-Accept, and implementers may wish to provide mechanisms to block those or other service types, or other attributes. Such mechanisms are outside the scope of this document.

Some additional discussion of issues related to proxies is in [RFC2865], Section 2:

The Access-Request is submitted to the RADIUS server via the network. If no response is returned within a length of time, the request is re-sent a number of times. The client can also forward requests to an alternate server or servers in the event that the primary server is down or unreachable. An alternate server can be

used either after a number of tries to the primary server fail, or in a round-robin fashion. Retry and fallback algorithms are the topic of current research and are not specified in detail in this document.

This text goes back to the original specification of [RFC2058], in 1997. Very little research on this topic has been performed since then, with the effect that there has been very little guidance available for implementers or operators. This lack of guidance has resulted in ad-hoc solutions, and in unstable networks.

This document addresses that lack of guidance by providing operational considerations and best current practice recommendations for proxying. The recommendations here are the result of decades of experience by multiple RADIUS server implementers, along with decades of experience by operators of large-scale and large-volume proxy networks. These recommendations have been demonstrated to work, and are documented here in order to encourage their adoption.

The document first motivates the guidance by describing a set of issues with the RADIUS protocol, with associated solutions and/or recommendations for addressing these issues. It then continues with a set of recommendations for best current practices.

As a "best current practices" specification, this document does not make any changes to the RADIUS protocol.

1.1. Proxy versus Client or Server

A proxy is composed of a server to receive packets, and an associated client to forward packets to another destination. The discussion here is limited to proxies, but there is of necessity overlap with recommendations for clients and servers. However, there are additional considerations for proxies which are not seen by a stand-alone client or server. This document focusses on issues seen by proxies.

It is still RECOMMENDED that client and server implementations be aware of the issues raised in this document, and where relevant, implement the recommendations made herein. Where the recommendations below apply to a stand-alone client or server, that relevance is mentioned.

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. Overview of Proxy Architecture and Use Cases

- * RADIUS proxying started for roaming
- * eduroam
- * openroaming
- * inter-ISP roaming
- * WiFi offload
- * multiple chains of proxies
- * centralized proxies
- * RFC7585 dynamic lookups

3.1. Real World Impact

- * fragility of proxy chains
- * DoS attacks from bad supplicants, APs / bad clients
- * fragility of UDP across the Internet

4. Problems and Solutions

This section outlines problems with the RADIUS protocol and the recommended solutions.

4.1. Silently Discard

[RFC2865], Section 1.2 defines the term "silently discard" 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 it can be necessary to silently discard packets for security reasons, there are other situations where packets are silently discarded for reasons unrelated to security. These silent discards are a large source of instability in RADIUS deployments.

4.1.1. Problem

There are cases where an implementations has to silently discard packets because there is no defined way to respond to them. The result is that a server may be operational, but not responsive. A client has no way of knowing if the server is down, slow, or if there is a problem somewhere else down a long chain of proxies.

Some implementations are known to silently discard packets:

- * unrecognized attributes
- * when they get packets in the middle of an EAP session which isn't theirs (see load-balancing, below).

Standards are silent or wrong:

- * when a server gets unexpected packets, even when the packets are authenticated (Accounting-Request to a port which is not configed to accept them)
- * when an accounting server can't store the data. There is no Accounting-NAK.

4.1.2. Solution

If a packet is authenticated, AND is of Code which the server normally accepts, it MUST respond to the packet. e.g. Access-Reject, or CoA-NAK, or Disconnect-NAK.

Protocol-Error. See Internet-Draft TBD.

4.2. Access-Reject

Many issues here.

4.2.1. Problem

- * Access-Rejects are not delayed, contributing to dictionary / DoS attacks
- * when they are delayed, they cause increased load on intermediate proxies to track state

- * eduroam and others see a significant percentage of traffic (30%+) as rejects

4.2.2. Solution

- * delay rejects
- * but push delays to the edge
- * cache rejects at the edge

4.3. Retransmission

Retransmits have the possibility of creating packet storms.

4.3.1. Problem

Some proxies ignore client retransmits. Others only retransmit when the client retransmits. Others do both (!!!)

4.3.2. Solution

[RFC3539], Section 2.8 explains that end-to-end retransmissions are the best. [RFC6613], Section 2.5 also says

Intrinsically, proxy systems operate with multiple control loops instead of one end-to-end loop, and so they are less stable. This is true even for TCP-TCP proxies. As discussed in [RFC3539], the only way to achieve stability equivalent to a single TCP connection is to mimic the end-to-end behavior of a single TCP connection. This typically is not achievable with an application-layer RADIUS implementation, regardless of transport.

The main lack here is the negative signaling of lost packets. Protocol-Error helps here.

Proxies MUST suppress duplicate retransmissions from a client within a short time frame, say 1s.

Proxies MUST NOT do their own retransmissions, subject to transport issues. i.e. a TCP to UDP proxy MUST retransmit on the UDP side.

4.4. Acct-Delay-Time and Congestive Collapse

Acct-Delay-Time ([RFC2866], Section 5.2 requires that Accounting-Request packets to be updated when the packet is retransmitted. It's an artifact of 1993, when NTP was not widely used.

4.4.1. Problem

Acct-Delay-Time contributes to congestive collapse of the network. One set of accounting information with the same data (other than Acct-Delay-Time) can be sent as many different packets. When seen in conjunction with the retransmission issues noted above, it is possible to increase that number substantially.

4.4.2. Solution

- * clients shouldn't send Acct-Delay-Time
- * proxies which do store and forward (see below) should just use Event-Timestamp, and suppress Acct-Delay-Time.

4.5. There is no RADIUS Routing Protocol

4.5.1. Problem

Packets may take multiple paths through the network.

Old packets for an EAP session may appear at the home server late. This happens when there is a proxy fail-over, and one element retransmits. This is not just UDP, it's possible with TCP, too, when connections go up / down.

4.5.2. Solution

Home servers MAY cache responses to Access-Request packets based on the State attribute. And then reply them when the same State is seen for a different packet src IP/port/Code. This caching is different from the [RFC5080], Section 2.2.1 packet dedup!

OR a home server MAY respond with a Protocol-Error indicating "unknown State". This can also occur for home server fail-over as seen in Section 5.3.1.

4.6. Filtering

Proxies can filter attributes in both requests and responses.

4.6.1. Problem

Access-Request and Accounting-Request packets can contain private information about the local network, which no one else needs to see.

Access-Accept packets can contain assignments which are not appropriate for a remote network.

4.6.2. Solution

Clients and proxies should allow servers or next hops to be marked as "internal" or "external". Servers marked "internal" should be sent all attributes. Servers marked "internal" should have private local information stripped, and replaced with Opaque-NAS-Identifier, etc.

In general, only send attributes which are necessary for authentication / accounting, and filter out everything else. i.e. have a positive filter, not a negative one

These recommendations can be superseded by agreements between parties, such as to send location information.

- * include attributes needed for the process
- * or which are required by the inter-party agreements
- * everything else should be filtered out.

4.7. Inadequate signal of errors

It is hard to distinguish protocol-layer errors from (e.g.) authentication errors.

4.7.1. Problem

A policy decision says the user can be rejected. "credentials are OK, but you're not allowed to roam there".

The non-technical decisions should be signaled in some shape or form

There is also no way to signal "account was deleted 10 years ago".

4.7.2. Solution

Error-Cause should be allowed in Access-Reject or other packets maybe just have one "non-technical reason".

Perhaps also have an Error-Cause for "please delete these credentials".

4.8. Clients misbehave with Delayed Rejects

4.8.1. Problem

Some NASes deal badly with delays. If they don't get a response within <0.5s, they either retransmit aggressively, or fail over to another server. This behavior is ridiculous.

4.8.2. Solution

Clients MUST implement the [RFC5080] timeouts. Anything else will cause problems.

4.9. Timeouts vary wildly across a proxy chain

4.9.1. Problem

The timeouts on the original client and proxies are generally uncorrelated. This issue means that we have multiple points in the proxy chain where fail-over and/or timeouts can occur. This behavior does not contribute to network stability.

4.9.2. Solution

- * proxy timeouts should be lower than NAS timeouts, and then send Protocol-Error
 - timeouts should get shorter along the path.
- * perhaps client can add indication of timeout to packets, so proxies can know when the clients will give up.
- * perhaps a separate document?
- * should be on a per-packet basis
- * we don't know what the client timeout is?

5. Best Current Practices

5.1. Proxy-State

The only real utility of Proxy-State is loop detection. It is possible to implement a RADIUS server / proxy which does nothing more than forward / copy reply of Proxy-State as per [RFC2865], Section 5.33. A proxy doesn't in fact need to add a Proxy-State attribute in order to work in most situations.

Proxy implementations SHOULD create Proxy-State using a site-local 64-bit value, taken from a secure random number generator. This value can then be used to identify a particular proxy server, or set of servers. This value should be the same for all packets proxied by the server. This same value should be used by all "equivalent" proxies, such as ones used in a load-balance configuration. Using 64 bits means that the chance of a hash collision is approximately one in 2^{32} , which should be sufficient. i.e. there are significantly fewer than 2^{32} RADIUS servers on the planet.

Any one of the equivalent proxy servers can then examine Proxy-State, and look for their unique token. If the token exists, then a loop has been detected.

This recommendation isn't perfect. Other servers can still delete the Proxy-State, or copy the values that they see. But such corner cases are best addressed by administrator intervention.

This recommendation allows administrators and implementations to better detect inadvertent proxy loops, rather than deliberate misconfigurations.

Proxies should also NOT copy the Proxy-State attributes from the proxied reply to their reply. The downstream server may have deleted or modified the attributes. Instead a proxy should copy the Proxy-State from the original request to the reply.

5.2. Rate limiting

It is possible for a malicious client to send enormous amounts of packets to proxies in a roaming consortium. There are a few different cases:

- * users immediately retrying after a reject. Home servers SHOULD implement a reject delay. Proxies SHOULD also enforce a reject delay for proxied packets. Note that this reject delay should enforce AT LEAST a delta between request and response. It MUST NOT simply add a delay to the response.
- * clients overloading upstream proxies with thousands of packets. e.g. when a power failure occurs, there may be thousands or tens of thousands of devices which are all trying to authenticate in a short period of time. Such behavior can result in congestive collapse, and overload of home servers. Clients (and proxies) should implement some kind of rate limiting.

5.3. Load Balancing

Load balance Access-Request packets using Calling-Station-Id where it exists, because it's usually the MAC address. The User-Name may be "@example.com", so it is unsuitable for load balancing. Use it only when Calling-Station-ID does not exist.

Or, load balance based on session identification attributes (NAS-IP-Address, NAS-IPv6-Address, NAS-Port). Do NOT load-balance on packet source IP, or anything which identifies the NAS itself. A NAS may have multiple sessions!

If the system proxies both Access-Request and Accounting-Request packets, try to load balance on the same information for both types of packets. This consistency will ensure that all of the data for one sessions shares the same path (and fate) through the network.

When the next hop is known to be a home server, use consistent hashing for load balancing, in order to ensure that packets for one session are sent to the same destination. This is most important for EAP.

5.3.1. Home Servers

Load balancing is made more complex when the next hop is a home server, and the authentication is a multi-step process (challenge-response or EAP). When a home server fails, all "in process" authentications associated with that home server will fail.

However, as proxies do not track authentication sessions, the load balancing algorithms recommended above will simply redirect packets from the failed home server to other home servers. Those home servers will be unable to process packets in the middle of an authentication session.

And as noted above in Section 4.1, implementations are known to discard these packets without response, which contributes to instability.

One solution is for the proxy closest to the home server to maintain state for an authentication session. This is simple to do at low packet rates, but becomes more difficult at high packet rates.

One solution (experimental, but it should work) is to have the home server and the local proxies agree on magic values for the State attribute.

[RFC2865], Section 5.24 limits the use of State:

... the client MUST NOT interpret the attribute locally.

We relax this statement to say that a proxy MAY interpret the State attribute, but it MUST NOT do so without prior consultation with the next hop server.

The two systems could agree (for example) that the first octet of the State attribute is an identifier for the home server. Where State does not exist, the proxy can load-balance as described above. Where State does exist, it can bypass load balancing, and send the packet directly to the specified home server.

If the home server is down, the proxy knows that the packet cannot be processed by other home servers, and therefore does not redirect the packet to them. Instead, it returns a Protocol-Error indicating that the home server is unreachable.

Where the proxy cannot coordinate with the home server, it could instead add an octet (as an identifier) to the State for responses, and remove that octet from requests. This does violate the prohibitions of [RFC2865], Section 5.24, but it is likely to work in limited situations. i.e. when the only way to reach the home servers is either directly from clients (which do not do this), or when all proxies implement the same method. It is therefore fragile.

This behavior is likely to not work for other proxies in a multi-hop proxy chain.

5.4. Differing rates on inbound and outbound

This isn't limited to UDP to TCP proxying. Even for UDP, there may be differing limits on packets in / packets out. The best solution here is Protocol-Error.

There is no overload indication in RADIUS. Instead, there is only per-packet signaling.

Failing that, adding capacity.

5.5. Accounting Store and Forward

- * don't use a per-packet "store and forward", as multiple accounting packets with the same data will be sent. * instead, use session-based store and forward. Keep a session table (or even the last packet for a session), and send only that.

Session based store and forward also helps with poorly behaved clients, which are known to send accounting packets for all open sessions at fixed intervals. This is effectively a DoS attack on the server. A store and forward proxy can quench the attack by storing the data locally, and then proxying it upstream using [RFC5080], Section 2.2 algorithms.

5.6. Protection from DoS attacks

bad clients are attacking servers, with 1000's of packets a second.

Sometimes it's the NAS, which is allowing the user to retry immediately (i.e lms later) after a reject. OR allowing the supplicant to retry even before the reject is sent! so 1000's of packets a second. This appears to be a violation of 802.1X requirements on rate limiting

openroaming is public, and are getting attacked with Syn flood attacks.

Syn flood mitigation is critical for public RADIUS Servers.

Public RADIUS servers should have aggressive timers on new connections. If a connection doesn't complete within a short period of time, it should be closed or discarded. i.e. connections which complete and which make progress should be prioritized over connections which don't complete, or which don't make progress.

5.7. Don't be AntiSocial

See "deprecating insecure practices" Section 1.3.1 on standards. The standards are imperfect. Simply because something isn't forbidden, doesn't mean that it's allowed, or that it should be engaged in.

Ignoring standards or best practices is an artificial savings. Failing to spend the time on good engineering doesn't save you time or money; it just pushes that cost to your customers, and to your support organization.

6. IANA Considerations

This document has no actions for IANA.

7. 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.

8. Security Considerations

This specification has no direct impact on security. It does not add or change any existing cryptography 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 attacks, misconfiguration, and improper implementations. As a result, RADIUS proxy networks which implement this specification should become more stable, and therefore more secure.

9. Acknowledgements

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

10. References

10.1. 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>>.
- [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>>.
- [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>>.

10.2. Informative References

- [RFC2058] Rigney, C., Rubens, A., Simpson, W., and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2058, DOI 10.17487/RFC2058, January 1997, <<https://www.rfc-editor.org/rfc/rfc2058>>.

- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866,
DOI 10.17487/RFC2866, June 2000,
<<https://www.rfc-editor.org/rfc/rfc2866>>.
- [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/rfc/rfc3539>>.
- [RFC6613] DeKok, A., "RADIUS over TCP", RFC 6613,
DOI 10.17487/RFC6613, May 2012,
<<https://www.rfc-editor.org/rfc/rfc6613>>.

Contributors

Paul Dekkers

Karri Huhtanen
Radiator Software

Heikki Vatiainen
Radiator Software

Fabian Mauchle

Stefan Peatow
JISC

Jan-Frederik Rieckers
DFN

Author's Address

Alan DeKok (Ed)
InkBridge Networks
Email: aland@inkbridgenetworks.com