RADIUS EXTensions
Internet-Draft
Intended status: Experimental
Expires: 17 September 2025

J.-F. Rieckers
DFN

16 March 2025

Methods for Mitigation of Congestion and Load Issues on RADIUS Servers
draft-janfred-radext-radius-congestion-control-00

Abstract

The RADIUS protocol as defined in [RFC2865] does not have a means to signal server overload or congesition to the clients. This can lead to load problems, especially in a federated RADIUS proxy fabric. This document attempts to fix this.

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-janfred-radext-radius-congestion-control/.

Discussion of this document takes place on the RADIUS EXTensions 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/.

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 17 September 2025.

Table of Contents

1. Introduction 2
2. Conventions and Definitions 3
3. Protocol Description 4
3.1. Proxy-Capability Attribute 4
3.2. Response-Delay Attribute 6
3.3. Request-Block Attribute 6
3.3.1. Request-Block-Attribute and Request-Block-Extended-Attribute 6
3.4. RADIUS Instance behavior 7
3.4.1. RADIUS proxy 7
3.4.2. Enforcing Instance 8
4. Security Considerations 10
5. Recommendations for Operators 10
6. IANA Considerations 10
7. References 11
7.1. Normative References 11
7.2. Informative References 12
Appendix A. Document Status 12
A.1. Change History 12
Acknowledgments 12
Author's Address 12

1. Introduction

The RADIUS protocol [RFC2865] does not have a means to signal a server overload or a congestion to RADIUS clients. These overload situations may be a result of a high load of legitimate traffic and might even be worsened by retransmissions of packets the server failed to answer due to the high load. These situation can happen in a lost of scenarios. In RADIUS proxy fabric, a server overload may even result from a single RADIUS client, for example when an EAP supplicant immediately starts a new authentication try without delay

when getting a reject.

Especially in RADIUS proxy fabrics, the impact of misbehaving clients on the whole proxy chain can be reduced by reducing the packet load at the entry level or as early in the proxy chain as possible. Since the end user device cannot be controlled, we have to rely on the RADIUS proxies to implement coutermeasures.

These countermeasures can be used to reduce the load by one of two methods.

First, the response to requests can be delayed. By delaying RADIUS responses, the client has to wait for the answer to send its next request, which decreases the packet load on the server. This method can also be used to slow down clients that immediately retry the authentication once they receive a reject.

When a home server knows that an authentication of this client cannot succeed (for example because it used an expired certificate with EAP-TLS), and the client keeps retrying, any RADIUS actor along the proxy chain could generate a reject for this specific user.

Pushing these countermeasures to the the earliest possible RADIUS Instance inside the proxy chain has multiple advantages over rejecting it at the home server. First, it reduces the load on all proxies in the proxy chain, since they do not need to forward traffic that will get rejected anyway. Secondly, when the response should get delayed, pushing this delay as far down the proxy chain prevents RADIUS retransmissions. When the RADIUS proxy already has the response, it then does not need to proxy the retransmitted RADIUS packets, which reduces the load for the RADIUS proxies in the later proxy chain. Instead, the RADIUS proxy just ignores the retransmission, since it already has an answer for this RADIUS packet, but the answer is just delayed.

2. Conventions and Definitions

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.

Additionally, we use the following terms in this document, in the meaning as defined here:

RADIUS Instance A single device or software module that implements the RADIUS protocol

RADIUS Client A RADIUS Instance that sends RADIUS requests and receives RADIUS responses. This is only in reference to a single RADIUS hop.

RADIUS Server A RADIUS Instance that receives RADIUS requests and sends RADIUS responses. Similar to the RADIUS Client, this is only in reference to a single RADIUS hop.

RADIUS Proxy A single device or software module that acts as RADIUS server and RADIUS client at the same time. It receives RADIUS requests and forwards them towards the next RADIUS proxy, usually based on the realm of the User-Name attribute.

RADIUS Proxy Chain the list of RADIUS Instances that a RADIUS Request traverses from the first RADIUS Client across any number of RADIUS proxies to the final RADIUS Server that responds to the RADIUS Request. When referring to the RADIUS Request, the chain starts from the first RADIUS client sending the RADIUS Request and ends at the last RADIUS Server. For the RADIUS Response, the chain is reversed. The terms "earlier" and "later" will always be used together with a reference to a request or a response. As example: a RADIUS proxy earlier in the chain for a request is located later in the chain for the response.

Enforcing Instance The RADIUS Instance that enforces the response delay or the request blocking.

3. Protocol Description

The protocol extension consists of two parts: First, any RADIUS proxy in the proxy chain capable of either of the two countermeasures needs to signal this capability to the following RADIUS proxies and the home server, so they know whether or not they can use this feature. Second, for the reply, the home server or RADIUS proxy needs to signal the reply policy back to the previous RADIUS proxies.

3.1. Proxy-Capability Attribute

The Proxy-Capability Attribute is used to signal the capability of a RADIUS proxy to any RADIUS entity in the later proxy chain. With the help of this, on the reply path, a RADIUS proxy can determine whether the requested action should be performed by itself or the packet will pass through another capable proxy later which can then perform the actions.

The Proxy-Capability Attribute is of type string as defined in [RFC8044], Section 3.5. The value of the Proxy-Capability Attribute is a concatenated list of the proxy capabilities the RADIUS Instance has.

Correct formal description: TODO

Informal description: concatenate all capabilities. values up to 127 are encoded in one byte, extended capabilities are encoded as two bytes. For parsing, the receiver can look at the first bit, if it is a 0 it is a single-byte value, if it is a 1, then the capability is a two-byte value. This allows for simple extension, while keeping it as simple and short as possible. The attribute MUST NOT include a capability multiple times.

Each capable RADIUS Instance in the RADIUS Proxy Chain SHOULD add the Proxy-Capability Attribute for Access-Request and Accounting-Request packets before forwarding the RADIUS packet to the next RADIUS instance. Future capabilityes MAY specify capabilities for other RADIUS packet types. The capabilities defined in this document SHOULD only be added for Access-Request and Accounting-Request packets when Proxy-Capabilitiy is used with other RADIUS packets.

When a capable RADIUS proxy receives a RADIUS packet with the Proxy-Capability Attribute, the RADIUS Proxy SHOULD add its own capabilities to the Attribute if the capability is not yet included. The RADIUS Proxy MUST NOT remove existing capabilities, unless explicitly configured to remove them. As a hint, administrators SHOULD only configure the removal of capabilities when they know that the capability is not honored.

In this document, we define two capabilities:

Capability Response-Delay-Capable The Capability Response-Delay-Capable with value 1 is used to signal that the RADIUS Instance is capable of delaying RADIUS Response packets.

Capability Request-Block-Capable The capability Request-Block-Capable with value 2 is used to signal that the RADIUS Instance is capable of blocking RADIUS Requests that match specific criteria and sending an Access-Reject instead on behalf of the home server.

3.2. Response-Delay Attribute

The Response-Delay Attribute is used to signal the desire of the home server that sending of the RADIUS response should be delayed for a certain amount. The Response-Delay Attribute is of type integer as defined in [RFC8044], Section 3.1. The value is the delay in milliseconds.

3.3. Request-Block Attribute

The Request-Block Attribute is used to signal the desire of the home server that future requests that match certain criteria should be rejected by a RADIUS Instance on behalf of the home server. The Request-Block Attribute is of type tlv as defined in [RFC8044], Section 3.13. The sub-attributes are defined as follows:

Request-Block-Period This sub-attribute contains the time span in seconds during which requests matching the description should be rejected. The attribute is of type integer as defined in [RFC8044], Section 3.1.

Request-Block-Attributes This sub-attribute contains a list of attribute types that should be used to match authentication requests of the same user. The attribute is of type string as defined in [RFC8044], Section 3.5 and contains a concatenated list of one byte attribute types.

Request-Block-Extended-Attribute This sub-attribute contains a reference to a single extended attribute that should be included in the match. The attribute is of type string as defined in [RFC8044], Section 3.5.

3.3.1. Request-Block-Attribute and Request-Block-Extended-Attribute

RADIUS attributes are formatted as Type-Length-Value with a fixed one-byte type field. Since this allows for only 256 attributes, extended attributes have been defined in [RFC6929]. Additionally, RADIUS has vendor-specific attributes ([RFC2865], Section 5.26 or [RFC6929], Section 2.4, the header of which may not be known to all implementations. To still allow to filter on individual extended or vendor-specific attributes which might be unknown to the Enforcing Instance, we need a means to reference these attributes.

The Request-Block-Attributes sub-attribute is used to reference the "primitive" RADIUS attributes, that is RADIUS attributes that only have the one-byte attribute type. Since every of these types have a one-byte-length, we can reduce the overhead by concatenating the attribute types, as they are all one byte.

For the extended or vendor-specific attributes this is not as easy, since the length of the header may vary between the different attributes. Therefore, we have the Request-Block-Extended-Attribute sub-attribute, which references a single attribute that should be included in the blocklist. This sub-attribute can be included multiple times in the Request-Block attribute.

3.4. RADIUS Instance behavior

TODO - mostly stub or not complete specification, general description of the behavior for every involved party

3.4.1. RADIUS proxy

Any RADIUS Instance capable of delaying or blocking SHOULD add the Proxy-Capability attribute to every RADIUS Access-Request they send to a RADIUS server. If a RADIUS Instance receives a RADIUS request with this attribute, it SHOULD add its own capability, if not present already, to the proxied RADIUS packet and SHOULD NOT remove any other capability.

Upon reception a RADIUS Proxy needs to decide if it is the Enforcing Instance or not, by looking at the original request. This decision has to be done individually for the Response-Delay and the Request-Block policy.

If the received RADIUS response contains a Response-Delay attribute and the original request contained the Response-Delay capability, the RADIUS Proxy SHOULD NOT enforce the policy and instead forward the RADIUS response to the RADIUS client. If the original request did not contain the Response-Delay capability, the RADIUS Proxy MUST become the Enforcing Instance for the Response-Delay.

The algorithm for the Request-Block functionality is basically similar, but needs additional considerations in regards to present attributes. If the received RADIUS response contains a Request-Block attribute and the original request did not contain the Request-Block capability, the RADIUS Proxy MUST become the Enforcing Instance for the Request-Block. Otherwise, the RADIUS Proxy MUST check the presence of all attributes referenced in the Request-Block, whether or not they are present in the original RADIUS request. If an attribute was not present in the RADIUS request, the RADIUS Proxy MUST check if the missing attribute was added by the RADIUS Proxy and it is present in RADIUS request the RADIUS proxy sent to the next hop RADIUS server. In this case the RADIUS Proxy MUST become the Enforcing Instance, since any RADIUS Instances after the current RADIUS Instance cannot enforce the requested blocking, as at least one of the attributes is missing. If the missing attribute was not

added by the RADIUS Proxy, the RADIUS Proxy SHOULD remove the Request-Block attribute before forwarding the packet to the next RADIUS Client. In this case, the missing attribute was added by a RADIUS Instance not capable of Response-Block and positioned between the current and a later Response-Block capable RADIUS Instance in the RADIUS Proxy Chain of the RADIUS Request. Since we have no RADIUS-native method to signal this condition back, the best approach to deal with this is to ignore the block request. By removing the Request-Block attribute from the response, we reduce the load on later RADIUS Instances on the RAIDUS Proxy Chain for the RADIUS Response, since they do not need to perform the attribute checks. This removes the Response-Block functionality completely without signalling back to the capable RADIUS Instances earlier in the Response Proxy Chain. There is no easy solution to this problem, but this is considered the best solution compared to complicated signalling mechanisms that would only be beneficial in a limited number of use cases and increase the complexity of implementations. We therefore rely on the RADIUS server not to request a block based on attributes that may have been added during the proxy chain. See Section 5 for further discussion.

3.4.2. Enforcing Instance

An Enforcing Instance is in charge of performing the requested action.

In general, an Enforcing Instance MUST remove the corresponding RADIUS Attribute with the request to delay or block from the RADIUS response before forwarding it to the next RADIUS Client.

3.4.2.1. Response-Delay

An Enforcing Instance for the Response-Delay MUST delay the response it received from the RADIUS server before forwarding the RADIUS response to the next RADIUS Client. If the Enforcing Instance is not a RADIUS Proxy, any action that would normally follow the reception of the RADIUS response MUST be delayed, i.e. the Enforcing Instance acts as if the RADIUS response has not been received until the delay timespan has passed.

An Enforcing Instance MUST NOT retransmit the RADIUS Request again to the next hop RADIUS Server if it already received a RADIUS response with the Response-Delay attribute. If the Enforcing Instance is a RADIUS Proxy and the RADIUS Client retransmits the RADIUS request, the Enforcing Instance MUST silently discard the retransmission. This only applies for Enforcing Instances, any other RADIUS Proxy will still follow its normal retransmission policy.

The Enforcing Instance SHOULD delay the RADIUS response according to the time span in the Response-Delay attributes, however the Enforcing Instance MAY have an upper limit for the delaying response timespan. By default, this upper limit SHOULD not be less than 10000 milliseconds (10 seconds) and it SHOULD be configurable by the administrator.

// TODO: Reasoning behind the 10 sec delay: A common timeout limit
// for "response is missing, stop retrying" is 30 seconds, so by
// keeping the default upper limit below this we ensure that the
// response gets to the client, but it is delayed. We want to have
// the time configurable, because delaying responses uses up
// resources on the server. I delibirately didn't include text that
// the response should be sent if the ID space is exhausted, because
// the ID exhaustion may be the reason for the response delay further
// down the proxy chain, so in order to prevent impact on the later
// proxy chain, we need to shift the problem as far to the beginning
// of the proxy chain as possible.
//
// -- Janfred

// TODO: Maybe we should define a lower limit for the upper-limit
// config, i.e. the upper limit must not be less than 500
// milliseconds.
//
// -- Janfred

3.4.2.2. Request-Block

An Enforcing Instance for the Request-Block MUST reject the RADIUS requests with certain attributes.

In order to avoid servers from blocking legitimate traffic by setting the block-filter to arbitrary values, the Request-Block is always dependent on the attributes of the original RADIUS requests.

TODO: From here only stub.

General algorithm: * create a list of attributes * attributes that appear multiple times in the request must be included multiple times in the list as well * for extended attributes: add every attribute to the list with the prefix in the request-block-extended-attribute (but skip over the length byte in the attribute in the request) * When a request is received with attributes matching the list (every attributes must be present and match): send an Access-Reject with an Error-Cause attribute set to value TBD4 (Request ratelimited)

The considerations for upper limit should probably also apply, similar to the response-delay, but with way higher defaults

Maybe add functionality that the block is automatically timed out after a time when no login has be observed (to free up space), but "reset" the counter if observed again.

4. Security Considerations

TODO Security

5. Recommendations for Operators

TODO.

Elements to consider: * When should be delayed? * nearing ID-Exhaustion due to many ongoing EAP sessions, add small delays * on high server load add small delay * add a delay for a reject. (FreeRADIUS has the option already, let’s push this to the edge instead) * When should the home server request a block for how long? * outer username in EAP wrong - probably hours * inner username in EAP does not exist (and has several failed attempts shortly) - few minutes (outer username may stay the same if user changes only inner username) * username points to someone that is no longer eligible for access - probably hours * When should a proxy request a block? * repeated requests immediately after a reject with impact on the network - seconds to minutes * realm in username is unroutable - minutes to hours * What are good and bad choices for attributes? * good * User-Name. Always. * Calling-Station-ID * Called-Station-ID (i.e. only block phone on this specific access point) * NAS-Identifier, NAS-IPAddr, NAS-IPv6Addr (only block from this specific NAS, helpful in roaming scenarios) * bad * Proxy-State (added by proxies that might not understand it. MUST NOT be used) * Any other proxy-specific attribute * Only User-Name (may be too broad in case of anonymous identities)

6. IANA Considerations

This document will have IANA actions.

They are still TODO in detail.

Roughly the following things should be allocated:

* Attribute Type (possibly from extended attributes) for Proxy-Capability of type string (Extended-Attribute-1, TBD1)

* New registry table for for types in the Proxy-Capability attribute

- 0 - reserved
- 1 - Response-Delay-Capable
- 2 - Request-Block-Capable
- 3-125 - reserved for future use
- 126 , 127 - experimental
- 128 - 255 - Extended Capability

* Attribute Type for Response-Delay of type integer (Extended-Attribute-1, TBD2)
* Attribute Type for Request-Block of type tlv (Extended-Attribute-1, TBD3)
* New registry table for types in the Response-Delay attributes
  - 0 - reserved
  - 1 - Request-Block-Period, Type integer, request to stop sending data for this particular user for period of time, time in seconds
  - 2 - Request-Block-Attributes, type string
  - 3 - Request-Block-Extended-Attribute, type string
  - 4-250 - reserved for future use
  - 251 - private use
  - 252-255 - experimental
* New entry in the registry for Values for RADIUS Attribute 101, Error-Cause Attribute
  - 4XX (TBD4) with description "Request ratelimited"

7. References

7.1. 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>.

[RFC8044] DeKok, A., "Data Types in RADIUS", RFC 8044, DOI 10.17487/RFC8044, January 2017, <https://www.rfc-editor.org/rfc/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/rfc/rfc8174>.

7.2. Informative References

[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>.

Appendix A. Document Status

Note to RFC Editor: Remove this section before publication

A.1. Change History

draft-janfred-radext-radius-congestion-control-00:

- Initial draft version

Acknowledgments

TODO acknowledge.

Author’s Address

Jan-Frederik Rieckers
Deutsches Forschungsnetz | German National Research and Education Network
Alexanderplatz 1
10178 Berlin
Germany
Email: rieckers@dfn.de
URI: www.dfn.de