

Independent Submission
Internet-Draft
Intended status: Informational
Expires: August 11, 2008

S. Winter
RESTENA
M. McCauley
OSC
S. Venaas
UNINETT
February 8, 2008

RadSec Version 2 - A Secure and Reliable Transport for the RADIUS
Protocol
draft-winter-radsec-01

Status of This Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on August 11, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

This document describes implementations of a reliable transport (TCP) and security on the transport layer (TLS) for the RADIUS protocol [2]. This enables dynamic trust relationships between RADIUS servers.

Internet-Draft

RadSec profile for RADIUS

February 2008

Table of Contents

1.	Introduction	3
1.1.	Requirements Language	4
2.	Reliable Transport	4
2.1.	TCP	4
2.2.	Connection Keepalive	6
2.3.	Dead Peer Detection	6
3.	Transport Layer Security	6
3.1.	Operation	6
3.2.	Ciphersuites and Compression Negotiation	8
3.3.	RADIUS Shared Secret Usage in RadSec	9
4.	Comparison of Diameter vs. RadSec	9
5.	Diameter Compatibility	10
6.	Security Considerations	10
7.	IANA Considerations	11
8.	Acknowledgements	11
9.	References	11
9.1.	Informative References	11
9.2.	Normative References	11
Appendix A.	DNS NAPTR Peer Discovery	12
Appendix B.	Implementation Overview: Radiator	13
Appendix C.	Implementation Overview: radsecproxy	14

1. Introduction

The RADIUS protocol [\[2\]](#) is a widely deployed authentication and authorisation protocol. The supplementary RADIUS Accounting specification [\[3\]](#) also provides accounting mechanisms, thus delivering a full AAA solution. However, RADIUS is experiencing several shortcomings, such as its dependency on the unreliable transport protocol UDP and the lack of security for large parts of its packet payload.

Several enhancements have been proposed to overcome RADIUS' limitations. An IETF Standards Track protocol, Diameter [\[8\]](#), has been designed to provide an AAA protocol that should deprecate RADIUS. However, given that current implementations of Diameter are either not freely accessible, or do not provide the flexibility of current RADIUS deployments, or both, an intermediate solution that is based on RADIUS but provides mechanisms to overcome many of its drawbacks has been implemented by several vendors. These implementations are interoperable and deployed in a world-wide wireless roaming infrastructure. The protocol is called RadSec. This document describes version 2 of the RadSec protocol. Version 1 of RadSec is defined in the RadSec whitepaper [\[12\]](#). The two currently existing implementations of RadSec version 2 are described in [Appendix B](#) and [Appendix C](#).

The main focus of RadSec is to provide a reliable transport for RADIUS payload by defining a transport profile for the transport layer protocol TCP and a means to secure the communication between RADIUS peers on the transport layer. The most important use of RadSec lies in roaming environments where RADIUS packets need to be transferred through different administrative domains and untrusted, potentially hostile networks. An example for a world-wide roaming environment that uses RadSec to secure communication is "eduroam", see [\[17\]](#).

Since reliable transport protocols may experience long delays until

an outage on lower layers is detected and reported to the application layer, a means to ensure quick failure detection is defined as well. A detailed explanation of the motivations for not using Diameter is provided in [Section 4](#). The new features in RadSec obsolete the use of IP addresses and shared secrets to identify other peers and thus allow the dynamic establishment of connections to peers that are not previously configured. The definition of lookup mechanisms is out of scope of this document, but an implementation of a DNS NAPTR lookup based mechanism exists and is described as an example lookup mechanism in [Appendix A](#).

Transitioning from a plain RADIUS infrastructure to a RadSec

infrastructure is very easy, since the RADIUS packet payload is identical in both protocols. Enabling RadSec can be done on a per-server basis. Unlike in Diameter, the learning curve for a new protocol does not exist, which makes it almost trivial for an experienced RADIUS server administrator to switch to a RadSec-secured transport for RADIUS packets.

The transport profile and security layer do not require any new assignments of codepoints for the RADIUS protocol. No new attributes are defined and no new packet codes are used. Also the TCP port number for RadSec is already assigned by IANA.

[1.1](#). Requirements Language

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [\[1\]](#).

[2](#). Reliable Transport

The RADIUS specification chose UDP as a transport profile for its packets. The argumentation for this is valid as long as the authentication process can be completed with a single packet in each direction, because then none of the communicating servers needs to maintain state of the authentication process. With the advent of EAP authentications, particularly with the use of the IEEE 802.1X specification in LAN environments, authentication processes typically

require several packets, so maintaining state is important. Furthermore, the RADIUS packets during EAP conversations are dependent on being received in the correct order. This leverages the requirements on the transport profile, and brings the requirements very close to those of typical reliable transports, i.e. TCP and SCTP. Using a reliable transport in turn obsoletes several of the custom transport rules that apply to UDP RADIUS packets, like guessing the reachability of servers upon observation of a not-replied-to packet. The following section specifies the transport of RADIUS payload over TCP.

[2.1.](#) TCP

The default TCP port for servers willing to receive RadSec messages is 2083, as assigned to the initial OSC RadSec implementation by IANA. The source port for clients sending RadSec messages is arbitrary. An implementation may act as both a server (receive incoming requests and reply to these) and a client (initiate requests and receive replies) simultaneously.

A RadSec node which establishes a connection to another node (i.e. which acts as a client) uses this connection only to

- o send RADIUS Access-Request and Status-Server messages and process only the associated replies: Access-Challenge, Access-Accept and Access-Reject
- o send Accounting-Request messages and process the associated Accounting-Response messages
- o Other incoming packets MUST be rejected.

A RadSec node which listens for incoming connections (i.e. acts as a server) only

- o processes incoming Access-Request and Status-Server packets in a given stream and only sends the associated replies back into the stream (Access-Challenge, Access-Accept and Access-Reject)
- o processes incoming Accounting-Request packets and replies with the associated Accounting-Response packets
- o Other incoming packets MUST be rejected.

NOTE: This specification does not include handling of CoA and Disconnect packets from [\[7\]](#) since no RadSec implementation currently supports handling of these packet types.

The consequence of the above rules is that a node who acts as a server and a client simultaneously and communicates with another peer needs to maintain two separate TCP connections with this peer, one for sending its requests as a client and one for receiving incoming requests.

Since the TCP connections carry TLS payload and establishing a TLS tunnel is computationally intensive, statically preconfigured connections SHOULD be kept alive throughout the lifetime of the connected RadSec instances (be it server or client). All preconfigured connections SHOULD be established on startup of the RadSec node. Connections that are established by dynamic discovery MUST be established as soon as the first authentication attempt commences and SHOULD be kept alive for a configurable time period afterwards. It is NOT RECOMMENDED to keep dynamically discovered connections alive for an indefinite amount of time, since the peer will in this case aggregate more and more connections to new peers over time, which eventually may lead to resource exhaustion. When a connection to a preconfigured peer gets lost during operation, periodic reconnection attempts MUST be attempted. For lost connections to a dynamically discovered peer reconnection attempts MAY be triggered. The interval between reconnection attempts in both cases is undefined and implementation-specific.

[2.2.](#) Connection Keepalive

Firewalls and similar devices that inspect TCP streams often terminate connections that have been running for too long without transferring any payload over them. The long-term TCP connections specified earlier in this section therefore need to intermittently transfer data to prevent the connection from being deliberately killed by an intermediate device. Two mechanisms can be used to keep a RadSec connection busy:

- TCP socket options - The operating system may offer mechanisms on the transport layer that keep sending packets back and forth on a connection even when there is no payload to be transferred.

- Status-Server packet transfer - The RadSec node that initiated the connection to another server sends Status-Server packets to its peer and receives an Access-Accept as defined in [\[10\]](#).

A RadSec node which initiates a RadSec connection SHOULD send the Status-Server packets defined in [10] according to the state machine in section 3.4 of [6] since RadSec operates on a reliable transport protocol. Whenever the underlying operating system permits the use of TCP keepalive socket options, their use is RECOMMENDED.

[2.3.](#) Dead Peer Detection

Once a TCP connection is established, it may take a significant amount of time for a RadSec node to detect outages of the link or the other node. A TCP connection in established state will only time out after a long delay. If the RadSec node has multiple redundant connections for a given realm, it is desirable to detect link outages as early as possible.

A RadSec server who is regularly sending Status-Server requests and does not receive a corresponding response within a configurable amount of time (according to section 3.4 of [6]) MAY treat the connection to the server as failed even when the TCP socket is not yet broken.

[3.](#) Transport Layer Security

[3.1.](#) Operation

Once the TCP connection between two RadSec nodes is established, a TLS session is established. At least TLSv1.1 [9] is used. Both nodes either mutually present a X.509 certificate which is verified according to [5] or use a shared key authentication for TLS which needs to be pre-configured out-of-band prior to the connection attempt.

The list of Certification Authorities that a node which acts as a

server is willing to accept SHOULD be sent during the Certificate Request message in the CertificateRequest struct (section 7.4.4 of [9]). Rationale: If a RadSec node acts as a client and is in possession of multiple certificates from different CAs (i.e. is part of multiple roaming consortia) and dynamic discovery is used, and the dynamic discovery mechanism does not provide sufficient meta information to identify the server's roaming consortium, then it is necessary to signal which consortium it is connecting to.

The list of Certification Authorities that a node which acts as a client is willing to accept can not be signaled within the TLS 1.1 handshake. This makes it impossible to select the right certificate if a RadSec node which is acting as a server for multiple roaming consortia (in possession of multiple certificates from different CAs) is contacted by a client. This situation is likely to change in TLS 1.2, according to [\[11\]](#). "Trusted CA Indication" as in [\[11\]](#), [section 6](#), SHOULD be used.

When using X.509 certificate validation, peer validation always includes a check on whether the DNS name or the IP address of the server that is contacted matches its certificate. When a RadSec peer establishes a new connection (acts as a client) to another peer, the following rules of precedence are used during validation:

- o If the client expects a certain fully qualified domain name (FQDN) and the presented certificate contains both at least one instance of the subjectAltName field with type dNSName and a Common Name, then the certificate is considered a match if any one of those subjectAltName fields matches the expected FQDN. The Common Name field is not evaluated in this case.
- o If the client expects a certain fully qualified domain name (FQDN) and the presented certificate does not contain any subjectAltName field with type dNSName, then the certificate is considered a match if the Common Name field matches the expected FQDN.
- o If the client expects a certain IP address and the presented certificate contains both at least one instance of the subjectAltName field with type iPAddr and a Common Name, then the certificate is considered a match if any one of those subjectAltName fields matches the expected IP address. The Common Name field is not evaluated in this case.
- o If the client expects a certain IP address and the presented certificate does not contain any subjectAltName field with type iPAddr, then the certificate is considered a match if the Common Name field matches the expected IP address.

Further restrictions on the certificate MAY be verified, depending on the trust fabric of the peering agreement.

If dynamic peer resolution is used, the above verification steps MAY not be sufficient to ensure that a connecting peer is authorised to

perform user authentications. In these cases, the trust fabric

SHOULD NOT depend on untrusted peer resolution methods like DNS to identify and authorise nodes. Instead, the operators of the RadSec infrastructure SHOULD define their own trust model and apply this model to the certificates after they have passed the standard validity checks above. Current RadSec implementations offer varying degrees of versatility for defining criteria which certificates to accept.

NOTE WELL: None of the current implementations provide configuration options for using TLS with pre-shared keys. However, the underlying libraries support it, so support for that should be implementable easily.

After the TLS session is established, RADIUS packet payloads are exchanged over the encrypted TLS tunnel. In plain RADIUS, the packet size can be determined by evaluating the size of the datagram that arrived. Due to the stream nature of TCP and TLS, this does not hold true for RadSec packet exchange. Instead, packet boundaries of RADIUS packets that arrive in the stream are calculated by evaluating the packet's Length field. Special care MUST be taken on the packet sender side that the value of the Length field is indeed correct before sending it over the TLS tunnel, because incorrect packet lengths can no longer be detected by a differing datagram boundary.

[3.2.](#) Ciphersuites and Compression Negotiation

RadSec implementations need not necessarily support all TLS ciphersuites listed in [\[9\]](#). Not all TLS ciphersuites are supported by available TLS tool kits and licenses may be required in some cases. The existing implementations of RadSec use OpenSSL as cryptographic backend, which supports all of the ciphersuites listed in the rules below:

To ensure interoperability, RadSec clients and servers MUST support the TLS [\[9\]](#) mandatory-to-implement ciphersuite: TLS_RSA_WITH_3DES_EDE_CBC_SHA.

In addition, RadSec servers SHOULD support and be able to negotiate all of the following TLS ciphersuites:

- o TLS_RSA_WITH_RC4_128_MD5
- o TLS_RSA_WITH_RC4_128_SHA
- o TLS_RSA_WITH_AES_128_CBC_SHA

In addition, RadSec clients SHOULD support the following TLS ciphersuites [\[4\]](#):

- o TLS_RSA_WITH_AES_128_CBC_SHA

- o TLS_RSA_WITH_RC4_128_SHA

Since TLS supports ciphersuite negotiation, peers completing the TLS negotiation will also have selected a ciphersuite, which includes encryption and hashing methods.

TLS also supports compression as well as ciphersuite negotiation. During the RadSec conversation the client and server MAY negotiate compression. However, a peer MUST continue the conversation even if the other peer rejects compression.

[3.3.](#) RADIUS Shared Secret Usage in RadSec

Within RADIUS [\[2\]](#), a shared secret is used for hiding of attributes such as User-Password, as well as in computation of the Response Authenticator. In RADIUS accounting [\[3\]](#), the shared secret is used in computation of both the Request Authenticator and the Response Authenticator.

Since in RADIUS a shared secret is used to provide confidentiality as well as integrity protection and authentication, the use of TLS ciphers which encrypt the stream payload in RadSec can provide security services sufficient to substitute for RADIUS application-layer security. Therefore, where TLS ciphers that provide encryption are used, it will not be necessary to configure a RADIUS shared secret.

Then, the secret shared between two RadSec peers MAY not be configured. In this case, the shared secret defaults to "radsec" (without the quotes). However, if the RadSec nodes negotiated a TLS cipher that provides integrity assurance only, the connection MUST be configured with a non-default RADIUS shared secret.

[4.](#) Comparison of Diameter vs. RadSec

The feature set in the Diameter Base Protocol is almost a superset of the features present in RadSec. Sophisticated mechanisms for keepalive, reliable transmission and payload security exist. The reason for specifying a short-term intermediate solution as opposed to using Diameter at the moment are the lack of suitable, publicly available and versatile implementations.

Typically, RADIUS servers do not only support the RADIUS protocol itself, but also provide interfaces to a wide variety of backends (credential stores) to actually verify a user's credentials. In highly heterogeneous environments like eduroam, where a lot of different backends are in use by the participating home servers

(OpenLDAP, Novell eDirectory, ActiveDirectory, SQL databases or plain text files, just to name a few), such versatility is a requirement.

Current Diameter server implementations focus on the validation of a small set of EAP methods (mostly EAP-SIM and EAP-TLS) and consequently on a small set of backends to verify these credentials.

A further requirement in environments like eduroam is affordability. Public institutions like schools and universities often face tight budgets, and so an open source implementation of Diameter would be desirable (just as FreeRADIUS is for the RADIUS protocol). Unfortunately, the few Open Source Software implementations of the Diameter protocol like OpenDiameter [14] or JDiameter [15] only provide a set of libraries, but no server at all.

Diameter's ability to resolve peers dynamically is limited to using either SLPv2 or DNS, whereas RadSec allows arbitrary peer resolution mechanisms. This greater amount of flexibility can pay off in many roaming federations. In eduroam's case, a central SAML-based meta data repository ("eduGAIN-MDS") is in place which is able to provide peer addresses. Using RadSec it is possible to resolve a peer's address through such a meta data system, whereas with Diameter it is not possible to use this repository natively.

[5.](#) Diameter Compatibility

Since RadSec is only a new transport profile for RADIUS, compatibility of RadSec - Diameter vs. RADIUS - Diameter is almost identical. There is only one notable difference: since RadSec uses TCP to transport its payload, there is no datagram size restriction of 4096 Bytes for a RADIUS packet. In principle, very large Diameter payloads above 4096 Bytes that can not be translated into a RADIUS datagram can still be transported via RadSec. However, circumventing the size restrictions of [2] might break implementations that do not allocate sufficiently large buffers or discard payloads with a non-RFC-conforming packet size. Thus, using larger payload sizes than 4096 Bytes is NOT RECOMMENDED.

[6.](#) Security Considerations

The computational resources to establish a TCP connection and a TLS tunnel are significantly higher than simply sending mostly

unencrypted UDP datagrams. Therefore, clients connecting to a RadSec node will more easily create high load conditions and a malicious client might create a Denial-of-Service attack more easily.

In the case of dynamic peer discovery, a RadSec node needs to be able to accept connections from a large, not previously known, group of hosts, possibly the whole internet. In this case, the server's RadSec port can not be protected from unauthorised connection attempts with measures on the network layer, i.e. access lists and

firewalls. This opens more attack vectors for Distributed Denial of Service attacks, just like any other service that is supposed to serve arbitrary clients (like for example web servers).

Some TLS ciphersuites only provide integrity validation of their payload, and provide no encryption. When such ciphersuites are negotiated in a RadSec TLS handshake, the only means of protecting sensitive payload contents is the RADIUS shared secret. If the RADIUS shared secret is set to the default "radsec" and a non-encrypting TLS ciphersuite is used, implementations should either forbid transmitting payload over this connection completely or at least issue a warning to whatever logging destination is configured by the administrator.

[7.](#) IANA Considerations

This document has no actions for IANA. The TCP port 2083 was already previously assigned by IANA for RadSec. The Status-Server packet was already assigned by IANA for [\[2\]](#).

[8.](#) Acknowledgements

RadSec version 1 was first implemented by Open Systems Consultants, Currumbin Waters, Australia, for their "Radiator" RADIUS server product.

[9.](#) References

[9.1.](#) Informative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [RFC 2119](#), March 1997.

9.2. Normative References

- [2] Rigney, C., Rubens, A., Simpson, W., and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", [RFC 2865](#), June 2000.
- [3] Rigney, C., "RADIUS Accounting", [RFC 2866](#), June 2000.
- [4] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", [RFC 3268](#), June 2002.
- [5] Housley, R., Ford, W., Polk, T., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 3280](#), April 2002.

Winter, et al.

Expires August 11, 2008

[Page 11]

Internet-Draft

RadSec profile for RADIUS

February 2008

- [6] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", [RFC 3539](#), June 2003.
- [7] Chiba, M., Dommety, G., Eklund, M., Mitton, D., and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", [RFC 5176](#), January 2008.
- [8] Calhoun, P., Laughney, J., Arkko, J., Guttman, E., and G. Zorn, "Diameter Base Protocol", [RFC 3588](#), September 2003.
- [9] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", [RFC 4346](#), April 2006.
- [10] DeKok, A., "Use of Status-Server Packets in RADIUS", February 2007, <<http://www.ietf.org/internet-drafts/draft-dekok-radius-status-server-01.txt>>.
- [11] DeKok, A., "", February 2007, <<http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc4366-bis-01.txt>>.
- [12] Open System Consultants, "RadSec - a secure, reliable RADIUS Protocol", May 2005, <<http://www.open.com.au/radiator/radsec-whitepaper.pdf>>.
- [13] Open System Consultants, "Radiator Radius Server - Installation

and Reference Manual", 2006,
<<http://www.open.com.au/radiator/ref.html>>.

[14] Open Diameter Project, "Open Diameter", 2006,
<<http://www.opendiameter.org/>>.

[15] Svenson, E., "JDiameter Project Homepage", 2006,
<<https://jdiameter.dev.java.net/>>.

[16] Venaas, S., "radsecproxy Project Homepage", 2007,
<<http://software.uninett.no/radsecproxy/>>.

[17] UNINETT, "eduroam Homepage", 2007, <<http://www.eduroam.org/>>.

Appendix A. DNS NAPTR Peer Discovery

The following text is quoted from the file goodies/dnsroam.cfg in the Radiator distribution; further documentation of the <AuthBy DNSROAM> feature in Radiator can be found at [13]. It describes an algorithm to retrieve the RadSec route information from the global DNS using NAPTR and SRV records. The input of the algorithm is the realm part of the user name.

The following algorithm is used to discover a target server from a Realm using DNS:

1. Look for NAPTR records for the Realm.
2. For each NAPTR found record, examine the Service field and use that to determine the transport protocol and TLS requirements for the server. The Service field starts with 'AAA' for insecure and 'AAAS' for TLS secured. The Service field contains '+RADSECS' for RadSec over SCTP, '+RADSECT' for RadSec over TCP or '+RADIUS' for RADIUS protocol over UDP. The most common Service field you will see will be 'AAAS+RADSECT' for TLS secured RadSec over TCP.
3.
 - A. If the NAPTR has the 'S' flag, look for SRV records for the name. For each SRV record found, note the Port number and then look for A and AAAA records corresponding to the name in the SRV record.
 - B. If the NAPTR has the 'A' flag, look for a A and AAAA records for the name.
4. If no NAPTR records are found, look for A and AAAA records based

- directly on the realm name. For example, if the realm is 'exemplerealm.edu', it looks for records such as '_radsec._tcp.exemplerealm.edu', '_radsec._sctp.exemplerealm.edu' and '_radius._udp.exemplerealm.edu',
5. All A and AAAA records found are ordered according to their Order and Preference fields. The most preferable server address is used as the target server address, along with any other server attributes discovered from DNS. If no SRV record was found for the address, the DNSROAM configured Port is used.

For example, if the User-Name realm was 'exemplerealm.edu', and DNS contained the following records:

```
exemplerealm.edu. IN NAPTR 50 50 "s" "AAAS+RADSECT" ""
_radsec._tcp.exemplerealm.edu.
_radsec._tcp.exemplerealm.edu. IN SRV 0 10 2083
radsec.exemplerealm.edu.
radsec.exemplerealm.edu. IN AAAA 2001::202:44ff:fe0a:f704
```

Then the target selected would be a RadSec server on port 2083 at IPv6 address 2001::202:44ff:fe0a:f704. The connection would be made over TCP/IP, and TLS encryption would be used. This complete specification of the realm is the most flexible and is recommended.

[Appendix B](#). Implementation Overview: Radiator

Radiator implements the RadSec protocol for proxying requests with the <Authby RADSEC> and <ServerRADSEC> clauses in the Radiator configuration file.

The <AuthBy RADSEC> clause defines a RadSec client, and causes Radiator to send RADIUS requests to the configured RadSec server using the RadSec protocol.

The <ServerRADSEC> clause defines a RadSec server, and causes Radiator to listen on the configured port and address(es) for connections from <Authby RADSEC> clients. When an <Authby RADSEC> client connects to a <ServerRADSEC> server, the client sends RADIUS requests through the stream to the server. The server then services the request in the same way as if the request had been received from a conventional UDP RADIUS client.

Radiator is compliant to version 2 of RadSec if the following options are used:

<AuthBy RADSEC>

- * Protocol tcp
- * UseTLS
- * TLS_CertificateFile
- <ServerRADSEC>
- * Protocol tcp
- * UseTLS
- * TLS_RequireClientCert

As of Radiator 3.15, the default shared secret for RadSec connections is "mysecret" (without quotes). The implementation uses TCP keepalive socket options, but does not send Status-Server packets. Once established, TLS connections are kept open throughout the server instance lifetime.

[Appendix C](#). Implementation Overview: radsecproxy

The RADIUS proxy named radsecproxy was written in order to allow use of RadSec in current RADIUS deployments. This is a generic proxy that supports any number and combination of clients and servers, supporting RADIUS over UDP and RadSec. The main idea is that it can be used on the same host as a non-RadSec client or server to ensure RadSec is used on the wire, however as a generic proxy it can be used in other circumstances as well.

The configuration file consists of client and server clauses, where there is one such clause for each client or server. In such a clause one specifies either "type tls" or "type udp" for RadSec or UDP transport. For RadSec the default shared secret "mysecret" (without quotes), the same as Radiator, is used. A secret may be specified by putting say "secret somesharedsecret" inside a client or server clause.

In order to use TLS for clients and/or servers, one must also specify where to locate CA certificates, as well as certificate and key for the client or server. This is done in a TLS clause. There may be one or several TLS clauses. A client or server clause may reference a particular TLS clause, or just use a default one. One use for multiple TLS clauses may be to present one certificate to clients and

another to servers.

If any RadSec (TLS) clients are configured, the proxy will at startup listen on port 2083, as assigned by IANA for the OSC RadSec

implementation. An alternative port may be specified. When a client connects, the client certificate will be verified, including checking that the configured FQDN or IP address matches what is in the certificate. Requests coming from a RadSec client are treated exactly like requests from UDP clients.

The proxy will at startup try to establish a TLS connection to each (if any) of the configured RadSec (TLS) servers. If it fails to connect to a server, it will retry regularly. There is some back-off where it will retry quickly at first, and with longer intervals later. If a connection to a server goes down it will also start retrying regularly. When setting up the TLS connection, the server certificate will be verified, including checking that the configured FQDN or IP address matches what is in the certificate. Requests are sent to a RadSec server just like they would to a UDP server.

The proxy supports Status-Server messages. They are only sent to a server if enabled for that particular server. Status-Server requests are always responded to.

This RadSec implementation has been successfully tested together with Radiator. It is a freely available open-source implementation. For source code and documentation, see [[16](#)].

Authors' Addresses

Stefan Winter
Fondation RESTENA
6, rue Richard Coudenhove-Kalergi
Luxembourg 1359
LUXEMBOURG

Phone: +352 424409 1
Fax: +352 422473
EMail: stefan.winter@restena.lu
URI: <http://www.restena.lu>.

Mike McCauley
Open Systems Consultants
9 Bulbul Place
Currumbin Waters QLD 4223
AUSTRALIA

Phone: +61 7 5598 7474
Fax: +61 7 5598 7070
EMail: mikem@open.com.au
URI: <http://www.open.com.au>.

Stig Venaas
UNINETT
Abels gate 5 - Teknobyen
Trondheim 7465
NORWAY

Phone: +47 73 55 79 00
Fax: +47 73 55 79 01
EMail: stig.venaas@uninett.no
URI: <http://www.uninett.no>.

Internet-Draft

RadSec profile for RADIUS

February 2008

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgements

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA). This document was produced using xml2rfc v1.32 (of <http://xml.resource.org/>) from a source in [RFC-2629](#) XML format.