radsecproxy-addon

RADIUS/TLS: Using radsecproxy as an add-on to an existing RADIUS server

Goals and Prerequisites

This section describes a trimmed-down installation of radsecproxy which enables it to augment any RADIUS server with RADIUS/TLS transport, in order to enhance security. More precisely, with this setup radsecproxy will:

- Accept requests from a RADIUS server running on the same host
- Unconditionally forward the requests via RADIUS/TLS to an upstream server (typically a federation-level or a regional proxy server)
- Accept requests via RADIUS/TLS from the upstream server
- Unconditionally forward these requests to a RADIUS server running on the same host

The prerequisites for this deployment are:

- radsecproxy version 1.4.2 or higher
- The local RADIUS server's DEFAULT realm is configured to forward requests to the radsecproxy port on localhost.
- The local RADIUS server has configured localhost as a client (this is typically the case).
- The deployment requires a server certificate and a private key for that certificate to establish the RadSec connection which designates the server as an eduroam SP and IdP. For further information regarding eduroam certificates see this section.

Installation

On UNIX-like systems, the installation is very simple:

1. Download the code from GitHub https://radsecproxy.github.io/.
2. Unpack the code.
3. Navigate into the unpacked directory (the base directory)
4. type the usual UNIX compilation sequence:

```
./configure
make
make check
make install
```

4. After compiling and installing, the executable

```
radsecproxy
```

is in the installed directory. Execution of the installed binaries does not require root rights.

5. Copy the template configuration file below into

```
/etc/radsecproxy.conf
```

6. Create the directory */etc/radsecproxy/certs/ca/*. The template configuration file requires this directory to contain the accredited CA root certificates and the corresponding Certificate Revocation Lists (CRLs) in their OpenSSL hash form. See this section for information about the CA download
7. Fill the lines marked with *_STUFF_* with the required information as explained below.
8. Start radsecproxy and enjoy (for first-time use, starting it with the -f option is recommended, it will start radsecproxy in the foreground and show some verbose startup messages).

Sample configuration

Most of the radsecproxy configuration file is static. This section goes through the template radsecproxy.conf line by line and explains the meaning of each stanza.

```
ListenUDP                       localhost:11812
ListenTCP                       *:2083
```

Since there is a RADIUS server on the same host that occupies UDP/1812, radscproxy has to listen on a nonstandard port. It only needs to listen on the loopback device since it will only communicate with the RADIUS server on the same machine. The choice of 11812 is arbitrary and can be adapted if that port is in use. Since radsecproxy will also accept requests from an upstream RadSec-enabled server, it listens on the default TCP port for RadSec (2083) for requests from outside (the * meaning: all interfaces). If you want radsecproxy to listen only on specific interfaces, enter the interface names here. Beware: in this case you may also have to set the more exotic option SourceTCP (see the man page of radsecproxy for details).

Local Logging

A logging level of 3 is the default and recommended log level. Radsecproxy will then log successful and failed authentications on one line each. The log destination is the local syslog destination.

```
LogLevel               3
LogDestination         x-syslog:///LOG_LOCAL0
```

radsecproxy features a semi-automatic prevention of routing loops for RADIUS connections. If you define a client and server block (see below) and give them the same descriptive name, the proxy will prevent proxying from the client to that same server. Turn this feature on with:

```
LoopPrevention         On
```

F-Ticks

If you use Radsecproxy, you should send basic statistical information about the number of logins for national and international roaming to the eduroam Operations Team. The system to do that is "F-Ticks". radsecproxy has built-in support for F-Ticks: you simply add an option to all client { } definitions for which you know the country they are physically located in. That typically means all your connected institutions' RADIUS clients, at the national level, but excludes the international roaming top-level servers (e.g. the European Top-Level RADIUS Servers). For an institution it means all your WLAN controller connections. The client definition examples below assume that you do use F-Ticks.

When the client definitions are set-up, the following options enable F-Ticks and send the syslog messages in a privacy-preserving way (by hashing parts of the connecting end-user device's MAC address:

```
FTicksReporting Full
FTicksMAC VendorKeyHashed
FTicksKey arandomsalt
```

The ticks will end up in your local syslog daemon; they are NOT automatically sent forward to eduroam Operations. It will depend on your syslog configuration how to achieve forwarding of the messages. For "rsyslog", a popular recent syslog daemon, the following settings will make it work:

```
# radsecproxy

if      ($programname == 'radsecproxy') and ($msg contains 'F-TICKS') \
then    @192.0.2.204
&       ~
```

As usual, the IP address above is NOT the actual destination for the eduroam Operations F-Ticks server. Please contact eduroam OT for the the IP address of their server. Also keep your own server's IP address handy, because the F-Ticks server is firewalled to accept ticks only from known sources.

RADIUS/TLS

The following two sections define which TLS certificates should be used by default. This installation of radsecproxy always uses the same certificates, so this is the only TLS section. CACertificatePath contains the eduroam-accredited CA certificates with filenames in the OpenSSL hash form. The parameters below need to be adapted to point to your server certificate in PEM format, the private key for this certificate and the password for this private key if needed, respectively. If no password is needed for the private key, you can comment this line (precede it with a # sign). The option CRLCheck validates certificates against the Certificate Revocation List (CRL) of the CAs. It requires a valid CRL in place, or else the certificate validation will fail. Therefore, it is important to regularly update the CRLs by re-downloading them as described above.

Right now, checking CRLs is discouraged due to a pending bug in OpenSSL regarding CRL reloading.

Replace your paths to the certificate files as necessary - please refer to the "Certificates" section for details on how to obtain and manage RADIUS/TLS certificates.

```
tls defaultClient {
    CACertificatePath                   /etc/radsecproxy/certs/ca/
    CertificateFile                     /etc/radsecproxy/certs/CERT_PEM__
    CertificateKeyFile                  /etc/radsecproxy/certs/CERT_KEY__
    CertificateKeyPassword              __CERT_PASS__
    policyOID                           1.3.6.1.4.1.25178.3.1.1
#    CRLCheck                            On
}

tls defaultServer {
    CACertificatePath                   /etc/radsecproxy/certs/ca/
    CertificateFile                     /etc/radsecproxy/certs/CERT_PEM__
    CertificateKeyFile                  /etc/radsecproxy/certs/CERT_KEY__
    CertificateKeyPassword              __CERT_PASS__
    policyOID                           1.3.6.1.4.1.25178.3.1.2
#    CRLCheck                            On
}
```

The following section deletes attributes from RADIUS requests that convey VLAN assignment information. If VLAN information is sent inadvertently, it can cause a degraded or non-existent service for the end user because he might be put into the wrong VLAN. Connected service providers should filter this attribute on their own. Connected Identity Providers should not send this attribute at all. Checking for the existence of these attributes on your server is just an optional additional safety layer. If you do have a roaming use for cross-institution VLAN assignment, you may want to delete this stanza.

```
rewrite defaultClient {
     removeAttribute                      64
     removeAttribute                      65
     removeAttribute                      81
}
```

Client definition and request forwarding

To accept requests from the RADIUS server on localhost, this server needs to be listed as a client. That server may either use IPv4 or IPv6 for its communication, so both variants are defined. If your system is not IPv6-enabled, simply delete the stanza about client ::1. The *_LOCAL_SECRET_* must match the secret which you configured in your RADIUS server for the server catering the DEFAULT realm.

```
client localhost {
        host        127.0.0.1
        type        udp
        secret      __LOCAL_SECRET__
}

client ::1 {
        type        udp
        secret      __LOCAL_SECRET__
}
```

To accept requests from the upstream RADIUS/TLS-enabled server, this other server needs to be listed. Replace *_RADSEC_PEER_* with the hostname of your upstream server. The traditional RADIUS shared secret has no meaning in RADIUS/TLS, and must instead be statically set to "radsec" throughout eduroam. This has no security implications.

```
client _RADSEC_PEER_ {
        type        TLS
        secret     radsec
}
```

To deliver requests to the local RADIUS server, this server needs to be configured. See above for the parameter *_LOCAL_SECRET_*.

```
server localhost {
        type         UDP
        port         1812
        secret       __LOCAL_SECRET__
}
```

This server needs to be configured to deliver requests to the upstream RadSec-enabled server. See above for the configuration item *_RADSEC_PEER_*.

```
server _RADSEC_PEER_ {
        type         TLS
        secret      radsec
        statusserver on
}
```

There are some known client-side misconfigurations. If they were not already caught by the local RADIUS server, it does not make sense for the proxy to send these requests up to the eduroam infrastructure. These requests are immediately rejected.

Note: if you need to blacklist an existing realm for some reason, you can follow the myabc.com example, copying and replacing it with the realm to be blacklisted.

```
realm /myabc\.com$/ {
         replymessage "Misconfigured client: default realm of Intel PRO/Wireless supplicant!"
}

realm /^$/ {
          replymessage "Misconfigured client: empty realm!"
}
```

Requests that are coming in from upstream and are supposed to be handled by the own RADIUS server are listed in *_OWN_REALM_*. Create multiples of these stanzas if your local server serves more than one realm.

```
realm /_OWN_REALM_$/ {
             server localhost
}
```

The configuration stanza above will terminate all requests that end in *_OWN_REALM_* but which were not caught by previous realm definitions, which prevents a possible loop. This is achieved by having the client name = localhost and the server name = localhost, and LoopPrevention enabled, which checks for this condition.

Finally, all other realms which are not to be authenticated locally are sent to the upstream RADIUS/TLS peer:

```
realm * {
          server    __RADSEC_PEER__
}
```