radsecproxy-flr

radsecproxy

This section describes how to set up radsecproxy to act as a federation-level RADIUS and RADIUS/TLS server. It can then completely replace other RADIUS server products on the federation level.

More precisely, it will enable a server to:

- Accept requests from connected service providers via RADIUS and RADIUS/TLS.
- Forward requests to connected identity providers via RADIUS and RADIUS/TLS.
- Forward requests from international visitors to the European eduroam confederation root servers via RADIUS/TLS.
- Accept requests from the root servers via RADIUS/TLS for the own federation's users when they are roaming in another federation.

Version information

The prerequisites for this deployment are:

- radsecproxy version 1.6 or higher
- A server certificate and a private key for that certificate to establish the RADIUS/TLS connection which designates the server as an IdP+SP.

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 config file

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

```
ListenUDP                 *:1812
ListenTCP                 *:2083
```

radsecproxy will receive requests from all connected Service Providers via both RADIUS and RadSec. Therefore it has to listen on the appropriate ports on its network interfaces (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 options SourceUDP and/or 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

```
client 127.0.0.1 {
        type     udp
        secret   testing123
}

client ::1 {
        type     udp
        secret   testing123
}
```

There is no other RADIUS server running on localhost, which makes these client definitions almost superfluous. They may be of some use however to initiate debugging requests and tests from the server itself, so it is considered good practice to list localhost as a client. If your system is not IPv6-enabled, simply delete the second stanza.

```
client __SP_IP_ADDR__ {
        type     udp
        secret   __SP_SECRET__
        FTicksVISCOUNTRY XZ                 # will generate F-Ticks for a non-existant visited country
 }
```

Stanzas like this one are used for each connected service provider that is connected via RADIUS. You need to know the IP address of every SP's RADIUS server and negotiate a shared secret with the SP

Please note that the client and server stanza for the GEANT Monitoring (SA3-T2 activity) have the same host address, but different stanza names. This is important: it disables the LoopDetection for this host, and the SA3 monitoring deliberate uses loops to do its tests. The following stanza is the eduroam Service Activity's monitoring client. Negotiate the IP address and shared secret for European monitoring with the operators in SA3-T2 (eduroam Operational Team) and enter it here.

```
client SA3-monitoring-incoming {
         host            x.y.z.a
         type            UDP
         secret          __MONITORING_SECRET__
}
```

.

```
client incoming {
        host                            0.0.0.0/0
        host                            [::]/0
        type                            TLS
        tls                             defaultClient
        secret                          radsec
}
```

After all specific clients in the configuration, you can the above stanza as a "catch-all" for incoming RADIUS/TLS connections.It does not need to be modified (if you do not support IPv6, you can delete the second "host" line though). In particular, the string "radsec" for secret is fixed by the RADIUS/TLS protocol and is required to remain unchanged. It also has no effect; RADIUS/TLS depends on TLS security rather than the shared RADIUS secret.

The eduroam trust model requires that a SP that tries to connect has:

- A X.509 certificate from an eduroam-accredited CA
- which carries a Policy OID as configured above to prove authorisation as a eduroam Service Provider

These checks were defined via "tls defaultClient", above.

Request forwarding

To deliver requests to your connected IdPs, their servers need to be configured. This stanza is for IdP servers using RADIUS.

```
server __DESCRIPTIVE_NAME_ {
          host     __IP_ADDR__
          type     UDP
          secret   __SERVER_SECRET__
}
```

This is the equivalent stanza for IdP servers using RADIUS/TLS.

```
server __RADSEC_PEER_DNS_NAME_ {
          type         TLS
          tls          defaultServer
          secret       radsec
          statusserver on
}
```

The two following stanzas define the uplink to the European eduroam Confederation root servers. This entry can be kept as it stands and doesn't need any further configuration.

```
server etlr1.eduroam.org {
          type                  TLS
          tls                   defaultServer
          secret                radsec
          statusserver          on
}

server etlr2.eduroam.org {
          type                  TLS
          tls                   defaultServer
          secret                radsec
          statusserver          on
}
```

European monitoring works both ways. The client entry near the beginning of the configuration file was needed for incoming requests from the monitoring servers. The entry below specifies the outgoing connections to the monitoring server. Outgoing connections are currently monitored with RADIUS only. Use the negotiated IP address and shared secret with SA3-T2 Monitoring in the following stanza:

```
server SA3-monitoring-outgoing {
          host                  a.b.c.d
          type                  UDP
          secret                __MONITORING_SECRET__
}
```

After defining the server configurations, we need to define which RADIUS realms are going to be forwarded to which server(s). This is done in the remainder of the configuration file.

First, there are (very few) known-bad realms which are not forwarded at all. They should ideally never reach the FLR server, and be caught by the SP local RADIUS server, but as an extra safety measure they are filtered (i.e. immediately rejected) here:

```
realm /myabc\.com$/ {
          replymessage "Misconfigured client: default realm of Intel PRO/Wireless supplicant! Rejected by
<TLD>."
          accountingresponse on
}

realm /@.*3gppnetwork\.org$/ {
                  replymessage "Misconfigured client: Unsupported 3G EAP-SIM client!"
                  accountingresponse on
}

realm /^$/ {
          replymessage "Misconfigured client: empty realm! Rejected by <TLD>."
          accountingresponse on
}
```

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.

Requests for proper realms that are coming in from upstream and are supposed to be handled by an identity provider are listed in stanzas like the below. _*IDP_REALM*_ contains the realm of the connected IdP. Create one such stanza for each IdP realm. If an IdP has multiple servers for a failover configuration, you can list all servers in a row, as in the example below.

```
realm /IDP_REALM$/ {
              server         __FROM_SERVER_STANZAS_ABOVE__
              server         __BACKUP_NAME__
}
```

The configuration stanza below is for outgoing European monitoring connections.

```
realm /eduroam\.YOUR_TLD/ {
              server         SA3-monitoring-outgoing
}
```

All the valid realms were listed earlier in the configuration file, and this server is authoritative for the own TLD. If a supplicant or downstream servers sends a realm with the own TLD, but also with a realm name that is not registered, this request is unauthorised and bound to fail. It will be rejected immediately to prevent routing loops.

```
realm /\.YOUR_TLD$/ {
              replymessage "Misconfigured supplicant or downstream server: uses known-bad realm in <TLD> 
federation!"
}
```

Finally, all realms that do not belong to the own federation are forwarded to the European eduroam Confederation root servers. However, we limit this to 'sane' realms: these must include a tld of at least 2 characters. Anything else is dropped.

```
realm /@.+\..{2,}$/ {
              server           etlr1.eduroam.org
              server           etlr2.eduroam.org
}

realm * {
                          replymessage "Misconfigured client: username does not contain a valid realm!"
}
```

Goodies

This section contains some optional configuration parameters that can do good in many cases.

Keeping the config file at a manageable size

radsecproxy allows to split the configuration file into several files on disk and include the parts into the main configuration file. This is very practical when many sites have to be managed. You can create a subdir and put the client, server, realm parts together in one file per participant. By adding

```
include /etc/radsecproxy.conf.d/*.conf
```

into the main config file, you can put all the participant files into that directory.

Caveats