

# How to deploy eduroam at national level

- eduroam in a nutshell
  - General overview
  - Elements of the eduroam infrastructure
    - Confederation top-level RADIUS Server (TLR)
    - Federation-Level RADIUS servers (FLRs)
    - IdP and SP RADIUS infrastructure
    - Identity Management System
    - Supplicants
    - Access Points
    - Switches
- Becoming a Roaming Operator (RO)
  - Administrative requirements
  - Information management requirements
- Operating a Federation Level RADIUS server (FLR)
  - Hardware requirements
  - Software requirements and setup
  - Radiator
    - Version information
    - Installation
    - Sample config file
    - Base configuration / logging / F-Ticks
    - Client definition
    - Request forwarding
    - Goodies
      - Local logging of auths in one line
      - SNMP
    - Caveats
  - radsecproxy
    - Version information
    - Installation
    - Sample config file
    - Base configuration / logging / F-Ticks
      - Local Logging
      - F-Ticks
      - RADIUS/TLS
    - Client definition
    - .
    - Request forwarding
    - Goodies
      - Keeping the config file at a manageable size
    - Caveats
  - FreeRADIUS 3 - RADSEC
    - Version information
    - Sample config file
    - Installation
    - Caveats
  - VitalAAA
- Gauging your federation's performance
  - Monitoring
    - Federation monitoring in Europe: the eduroam Operational Team
    - Monitoring inside the federation
    - Nagios/Icinga: EAP Login checks
      - Preparatory work
      - Implementing the checks
    - Nagios/Icinga: RADIUS/TLS certificate validity checks
  - Statistics
- RADIUS/TLS: Obtaining and managing certificates
  - The eduroam server certificate trust model: eduPKI PMA and the eduroam Trust Profile
  - Managing accredited CAs in eduroam servers
  - Updating CRLs on your server
- Deployment Checklist

## eduroam in a nutshell

### General overview

eduroam stands for **education roaming**. It offers users from participating academic institutions secure Internet access at any other eduroam participating location. The eduroam architecture that makes this possible is based on a number of technologies and agreements, which together provide the eduroam user experience: "open your laptop and be online".

The crucial agreement underpinning the foundation of eduroam involves the mechanism by which authentication and authorisation works:

- The authentication of a user is carried out at their Identity Provider (IdP), using their specific authentication method.
- The authorisation decision allowing access to the network resources upon proper authentication is done by the Service Provider (SP), typically a WiFi hotspot (University campus, etc.).

In order to transport the authentication request of a user from the Service Provider to his Identity Provider and the authentication response back, a world-wide system of RADIUS servers is created. Typically every Identity Provider deploys a RADIUS server, which is connected to a local user database. This RADIUS server is connected to a federation level RADIUS server, which is either in turn connected to the upstream RADIUS server infrastructure or can connect to other RADIUS servers dynamically (using the protocol RADIUS/TLS). Because users are using usernames of the format "user@realm", where realm is the IdP's DNS domain name often of the form institution.tld (tld=top-level domain; both country-code TLDs and generic TLDs are supported), the RADIUS servers can use this information to route the request to the appropriate next RADIUS server until the IdP is reached. An example of the RADIUS hierarchy is shown in Figure 2.1.

To transfer the user's authentication information securely across the RADIUS-infrastructure to their IdP, and to prevent other users from hijacking the connection after successful authentication, the access points or switches deployed by the SP use the IEEE 802.1X standard that encompasses the use of the Extensible Authentication Protocol (EAP). EAP is a container that carries the actual authentication data inside, the so-called EAP methods. There are many EAP methods an IdP can choose from.

eduroam requires that the chosen EAP method must allow

- mutual authentication (i.e. the user can verify that he is connected to "his" IdP wherever the user is)
- encryption of the credentials used (i.e. only the user and his IdP will see the actual credential exchange; it will be invisible to the Service Provider and all intermediate proxies)

Some popular EAP methods in use in eduroam are

- PEAP ("Protected EAP") - a Microsoft protocol that establishes a TLS tunnel, and sends usernames and passwords in MS-CHAPv2 hashes inside)
- TTLS ("Tunneled TLS") - an IETF protocol that establishes a TLS tunnel, and sends usernames and passwords in multiple configurable formats inside)
- TLS ("Transport Layer Security") - an IETF protocol that authenticates users and the IdP with two X.509 certificates
- FAST ("Flexible Authentication via Secure Tunneling") - a Cisco protocol that establishes a TLS tunnel, and sends usernames and passwords in a custom way inside)

RADIUS transports the user's name in an attribute User-Name, which is visible in cleartext to all intermediate hosts on the way. Some EAP methods allow to put a different User-Name into the RADIUS packet than in the EAP payload. In that case, the following terms are used:

- outer identity: this is the User-Name in the RADIUS packet and visible to all intermediate parties
- inner identity: this is the actual user identification. It is only visible to the user himself and the Identity Provider

When using such EAP methods, and activating this option, the real username is not visible in RADIUS (it will only see the outer identity). Doing so will enhance the user's privacy, and is encouraged. Outer identities should be in the format "@realm" (nothing left of the @ sign, but the realm is the same as with the actual username). The realm part still must be the correct one as it is used to route the request to the respective Identity Provider. Once the IdP server decrypts the TLS tunnel in the EAP payload, it gets the inner identity and can authenticate the user.

After successful authentication by the Identity Provider and authorisation by the Service Provider, this SP grants network access to the user, possibly by placing the user in a specific VLAN intended for guests.

In the next chapter the various elements of this architecture and their functions is described.

Note: On responsibility for actions of the user: Directive 2001/31/EC article 12 defines the liability of a service provider:

- Where an information society service is provided that consists of the transmission in a communication network of information provided by a recipient of the service, or the provision of access to a communication network, Member States shall ensure that the service provider is not liable for the information transmitted, on condition that the provider:
  - (a) does not initiate the transmission;
  - (b) does not select the receiver of the transmission; and
  - (c) does not select or modify the information contained in the transmission.
- The acts of transmission and of provision of access referred to in paragraph 1 include the automatic, intermediate and transient storage of the information transmitted in so far as this takes place for the sole purpose of carrying out the transmission in the communication network, and provided that the information is not stored for any period longer than is reasonably necessary for the transmission.
- This Article shall not affect the possibility for a court or administrative authority, in accordance with Member States' legal systems, of requiring the service provider to terminate or prevent an infringement.

The complete Directive can be found at EUR-Lex1.

Figure 2.1: Layers of the eduroam RADIUS hierarchy **NEED TO (RE)CREATE DIAGRAM ??**

Please refer to deliverable DJ5.1.4 "Inter-NREN Roaming Architecture: Description and Development Items" for an in-depth description of eduroam and the underlying architecture.

## Elements of the eduroam infrastructure

### Confederation top-level RADIUS Server (TLR)

The confederation top-level RADIUS Servers, at the time of writing, are located in the Netherlands and Denmark for the European confederation, and Australia and Hong Kong for the Asian and Pacific region. Each have a list of connected country domains (.nl, .dk, .au, .cn etc.) serving the appropriate National Roaming Operators (NROs). They accept requests for federation domains for which they are authoritative, and subsequently forward them to the associated RADIUS server for that federation (and transport the result of the authentication request back). Requests for federation domains they are not responsible for are forwarded to the proper confederation TLR.

## Federation-Level RADIUS servers (FLRs)

A federation RADIUS server has a list of connected IdP and SP servers and the associated realms. Typically, a FLR is authoritative for all RADIUS realms ending in its own top-level domain (e.g. a FLR for Antarctica would be authoritative for \*.aq); it may also serve a number of domains in other top-level domains (e.g. .com, .net, .org, ...) but it is not authoritative for those entire top-level domains.

The FLR receives requests from the confederation servers and IdP/SP it is connected to and forwards them to the proper server. For its authoritative top-level domain, it rejects requests for non-existent realms inside the top-level domain.

## IdP and SP RADIUS infrastructure

eduroam IdPs operate a RADIUS server which is responsible for authenticating its own users, by checking the credentials against a local identity management system.

eduroam SPs operate RADIUS capable equipment like Access Points or switches (see below). Large SPs typically also deploy an own RADIUS server, which is then responsible for forwarding requests from visiting users to the respective federation RADIUS server. Upon proper authentication of a user the SP RADIUS server may assign a VLAN to the user. Small SPs which do not require VLAN assignments can connect their RADIUS equipment directly to their FLR server, if the FLR permits that mode of operation.

Institutions which opt to be eduroam IdP and eduroam SP at the same time can have one RADIUS server that fulfills both roles simultaneously. This is the most popular deployment model in eduroam.

Note that the IdP RADIUS server is the most complex of all. Whereas the other RADIUS servers merely proxy requests, the IdP server also needs to handle the requests, and therefore needs to be able to terminate EAP requests and perform identity management system lookups.

## Identity Management System

The Identity Management System of eduroam IdPs contains the information of the end users; for instance usernames and passwords. They must be kept up-to-date by the responsible IdP. An IdP RADIUS server will query the Identity management system to perform the actual authentication for a user as he tries to log in.

## Supplicants

A supplicant is a piece of software (often built into the Operating System but also available as a separate program) that uses the 802.1X protocol to send authentication request information using EAP. Supplicants are installed and operate on end-user computing devices (e.g. notebooks, PDAs, WiFi-enabled cell phones, and so on).

## Access Points

Access Points are Wireless LAN access devices conformant to IEEE 802.11 and need to be IEEE 802.1X capable. They must be able to forward access requests coming from a supplicant to the SP RADIUS server, to give network access upon proper authentication, and to possibly assign users to specific VLANs based on information received from the RADIUS server. Furthermore Access Points exchange keying material (initialisation vectors, public and session keys, etc.) with client systems to prevent session hijacking.

## Switches

Switches need to be able to forward access requests coming from a supplicant to the SP RADIUS server, to grant network access upon proper authentication and to possibly assign users to specific VLANs based on information received from the RADIUS server.

# Becoming a Roaming Operator (RO)

An eduroam federation comes with administrative requirements as well as technical ones. This document uses the eduroam Compliance Statement and the European Configuration definitions and documents; which provide a the baseline for the world-wide eduroam community.

## Administrative requirements

Operating a federation involves managing and supervising eduroam Identity Providers, eduroam Service Providers, as well as keeping authentication logs, fulfilling uptime requirements, etc. Prospect federation operators should read and understand the requirements in DS5.1.1 ("eduroam Service Definition and Implementation Plan") at [http://www.eduroam.org/downloads/docs/GN2-07-327v2-DS5\\_1\\_1-\\_eduroam\\_Service\\_Definition.pdf](http://www.eduroam.org/downloads/docs/GN2-07-327v2-DS5_1_1-_eduroam_Service_Definition.pdf), particularly sections 4.1.4 ("Roles and Responsibilities - NROs") and section 6 ("Requirements on Confederation Members").

A prospect NRO also needs to commit to the eduroam policy. The European eduroam policy document can be found at [https://www.eduroam.org/wp-content/uploads/2016/05/GN3-12-194\\_eduroam-policy-for-signing\\_ver2-4\\_1\\_18052012.pdf](https://www.eduroam.org/wp-content/uploads/2016/05/GN3-12-194_eduroam-policy-for-signing_ver2-4_1_18052012.pdf)

The RO may outsource the operation of its technical infrastructure (particularly, the Federation Level RADIUS servers) to a third-party, but will remain responsible for eduroam within its service area.

## Information management requirements

A Roaming Operator (RO) must maintain a comprehensive overview over eduroam within its service area, and report about its federation's state regularly. The vehicle for such reports is the eduroam database, where information about the RO and all its eduroam SPs and IdPs is stored. The database web interface is open for eduroam operators only; the entry page can be found here: [http://monitor.eduroam.org/db\\_web/](http://monitor.eduroam.org/db_web/)

Generic information on how to deliver information to the eduroam database (XML Schema format) can be found here: <http://monitor.eduroam.org/database.php>

## Operating a Federation Level RADIUS server (FLR)

Federation Level RADIUS (FLR) servers are used to connect eduroam Identity Providers and eduroam Service Providers with each other, and also provide an uplink from the federation to all other eduroam federations. They are managed by Roaming Operators (ROs). The RO may outsource the operation to a third-party, but will remain responsible.

Since the concept of an eduroam federation geographically usually maps to a territory or economy, FLRs are central to the deployment of eduroam; there is conceptually only one FLR per RO territory - but for resiliency reasons, it is recommended to provide multiple instances in a failover setup.

An eduroam federation comes with administrative requirements as well as technical ones. The exact requirements may differ between federations. This document uses the European definitions and documents; which provide a baseline for the world-wide eduroam community.

## Hardware requirements

RADIUS is a very lightweight protocol, and does not require expensive hardware setups. Even the busiest eduroam federations operate their server on a single contemporary hardware or Virtual Machine, without experiencing overload conditions.

As with every other professionally-operated service though, you should keep in mind that service uptime is paramount, and plan your procurement accordingly. Examples:

- In the case of virtual machines, use an underlying infrastructure which enables you to migrate machines without VM downtime, if possible.
- In the case of physical machines, use hot-pluggable parts where possible; and ideally, keep either spare hardware parts at hand or a set up a decent service contract.

eduroam Europe is in the process of migrating to RADIUS/TLS for its federation servers. In the course of this process, hardware requirements for the servers may change. This section will be updated as necessary.

## Software requirements and setup

eduroam does not prescribe any particular RADIUS implementation. The technical requirements for eduroam however narrow the set of usable RADIUS server implementations, and the observed deployment of eduroam federation-level servers shows patterns regarding implementation popularity.

This section will present a few typical implementation setups. Note, however, that a federation is free to use a different implementation so long as the implementation can satisfy the eduroam technical requirements.

The sections for each implementation are accompanied by a skeleton configuration file, which should be usable almost as-is. However, please read and try to understand the entire corresponding section before applying the template - the information presented is valuable for daily operation and troubleshooting.

## Radiator

Radiator is perhaps the most popular server software in eduroam federations. The config file and examples below assume deployment on a UNIX-like platform, such as Linux or FreeBSD. Radiator can also be used on Windows; in which case you will have to adapt some path names etc.



### Use of IP addresses in this document

The IPv4 and IPv6 addresses below are in the IETF "documentation" prefix ranges - you will need to adapt the addresses for your production use.

## Version information

This section of the document was created and is verified to work with at least

- Radiator 4.7
- Net::SSLeay 1.37 [prerelease]
- Perl 5.10

It is usually safe to assume that newer versions of these programs work as well.

Net::SSLeay 1.37 is the minimum required version for the RADIUS/TLS parts of the config to work completely: the version is needed for the TLS\_PolicyOID configuration parameter to work (which is needed for RADIUS/TLS server authorisation checks).

With currently only one CA exclusively issuing eduroam server certificates, the TLS\_PolicyOID check is not essential right now.

It is thus also safe to use version 1.36 (and commenting out the configuration lines regarding TLS\_PolicyOID). You should upgrade to 1.37 as soon as it is publicly released and re-enable the parameter in the configuration.

## Installation

### Sample config file

This is the [complete sample config](#). The contents are explained below.

### Base configuration / logging / F-Ticks

Radiator expects the configuration to be in file `/etc/radiator/radius.cfg`.

The parameter LogDir defines the directory in which start-up logs and PID file reside. DbDir defines the path to Radiator's data files, such as dictionaries.

```
LogDir    /var/log/radiator
DbDir     /usr/share/radiator
```

Throughout the configuration file, you may want to use DNS names instead of IP addresses. For RADIUS/TLS with dynamic discovery, it is even required to use DNS. The configuration for DNS is as follows (replace the IP addresses with your own):

```
<Resolver>
  Nameservers    198.51.100.254
  Nameservers    2001:db8:100::254
  NAPTR-Pattern x-eduroam:(radius)\.(tls)
  DirectAddressLookup 0
  # Debug
</Resolver>
```

The logs during normal operation are defined separately in `<Log>` stanzas. The verbosity of logging depends on the Trace level in the configuration: Trace 3 logs are recommended for normal operation, while Trace 4 logs provide verbosity for debugging, if needed. You can define several `<Log>` instances with different destinations. Let's define logging to syslog with verbosity level 3, and logging to a file for debugging purposes with verbosity level 4. We also define that the log file name changes on a daily basis to enable easy deletion of old files:

```
<Log SYSLOG>
  Facility      local7
  Identifier    log-syslog
  Trace        3
</Log>

<Log FILE>
  Filename      /var/log/radiator/radiator.%Y%m%d.log
  Identifier    log-file
  Trace        4
</Log>
```

You can also log authentication events in one line per authentication separately. The eduroam statistics system, F-Ticks, makes use of that feature. The F-Ticks logging facility is defined as follows:

```

<AuthLog SYSLOG>
  Identifier TICKS
  LogSuccess 1
  LogFailure 1
  LogSock udp
  LogHost 198.51.100.253
  SuccessFormat F-TICKS/eduroam/1.0#REALM=%R#VISCOUNTRY=%{eduroam-SP-Country}#VISINST=%{Operator-Name}
#CSI=%{Calling-Station-Id}#RESULT=OK#
  FailureFormat F-TICKS/eduroam/1.0#REALM=%R#VISCOUNTRY=%{eduroam-SP-Country}#VISINST=%{Operator-Name}
#CSI=%{Calling-Station-Id}#RESULT=FAIL#
</AuthLog>

```

Here, you need to adapt LogHost to the eduroam F-Ticks logging server (whose address you'll receive from eduroam operations), and the attribute marked with read. Its contents will become clearer later in the configuration file. Note: on some versions of Sys::Syslog and Radiator, you may need to reply "udp" with "inet".

If you monitor your national infrastructure, you will probably have automatic authentications happening which are triggered by your monitoring. F-Ticks can automatically separate these from real-world traffic and keep it out of the statistics. For that to work, you will have to use a value for Calling-Station-ID in your monitoring requests which begins with 22-44-66.

Next, the ports Radiator will use to listen for Authentication and Accounting requests must be defined. The port numbers 1812 and 1813 were assigned to the RADIUS protocol by IANA. Note: Exceptionally, you may come across very old RADIUS equipment which uses non-standard ports 1645 and 1646. Please see the Radiator documentation how to handle these, or consider upgrading the corresponding equipment.

```

AuthPort 1812
AcctPort 1813

```

## Client definition

In the client section, all possible peers from which the FLR server is going to accept requests, are listed. I.e. it includes all eduroam SPs in the federation and the uplink to the other federations (in Europe, to the ETLR servers).

For RADIUS, individual clients with their IP address have to be listed and a "secret" has to be assigned to them. As this secret is the only thing that protects the communication between the RADIUS servers from eavesdropping, it must be cryptographically strong (suggested: exactly 16 characters) and well protected.

The clients should also be tagged with the attribute Operator-Name. which takes the format "1<domainname>", and for F-Ticks classification reasons, also with the country the eduroam SP is located in (or UNKNOWN for clients whose geographic location isn't known).

Example: you have an eduroam SP which operates on the address 203.0.113.5 and have negotiated the shared secret "adf7856asdcvxb5p" with it. The SP is based in Antarctica, and uses the domain name "foo.aq". You want it to show up in log files as "icecold-radius".

```

# my eduroam SP in Antarctica

<Client 203.0.113.5>
  Secret          adf7856asdcvxb5p
  Identifier       icecold-radius
  AddToRequestIfNotExist Operator-Name=1foo.aq,eduroam-SP-Country=AQ
  RequireMessageAuthenticator
</Client>

```

Note: the Operator-Name attribute has the character "1" preceding the domain name. This is intentional and required as per the corresponding RFC. Please always prepend the character "one" to the domain names of the operator.

The clients for your uplink to ETLRs will look similar to the following. Note they are tagged with Country=UNKNOWN because requests coming from these countries can originate from all over the world (they connect **all other federations**). For the same reason, it also does not make sense to set the Operator-Name attribute.

```

<Client etlrl.eduroam.org>
  IdenticalClients etlrl.eduroam.org
  Secret           (as negotiated with eduroam OT)
  Identifier       etlrl.eduroam.org
  AddToRequestIfNotExist eduroam-SP-Country=UNKNOWN
  RequireMessageAuthenticator
</Client>

```

Two additional clients are useful: one client for localhost, which can be used for local debugging purposes (and which doesn't need a strong secret); and the client which used for European FLR monitoring (negotiate the actual client address eduroam OT) at

```
<Client 192.0.2.1>
  Secret          (as negotiated with eduroam OT)
  Identifier      Monitoring-ETLR
  AddToRequestIfNotExist eduroam-SP-Country=NONE
  RequireMessageAuthenticator
</Client>

<Client localhost>
  Secret          mysecret
  DupInterval     0
  AddToRequestIfNotExist eduroam-SP-Country=NONE
  RequireMessageAuthenticator
</Client>
```

Note: all the Identifier names in the configuration need to be unique, and should be meaningful to you, the server operator.

Finally, to enable RADIUS/TLS clients to communicate with your server, you need an additional section for RADIUS/TLS like the following. Replace your server's IP address(es) and paths to the certificate files as necessary - please refer to the "Certificates" section for details on how to obtain and manage RADIUS/TLS certificates.

```
<ServerRADSEC>
  Port            2083
  BindAddress     198.51.100.252, ipv6:2001:db8:1::26
  Secret         radsec
  Protocol        tcp
  UseTLS
  TLS_CAPath      /etc/radiator/certs/CAs/current/
  TLS_CertificateFile /etc/radiator/certs/server.pem
  TLS_CertificateType PEM
  TLS_PrivateKeyFile /etc/radiator/certs/server.key
  TLS_PolicyOID   1.3.6.1.4.1.25178.3.1.1
  TLS_RequireClientCert
  Identifier      RadSec
  AddToRequest    eduroam-SP-Country=UNKNOWN
</ServerRADSEC>
```

## Request forwarding

### Your eduroam IdPs

eduroam authentication requests are routed based on the User-Name attribute in the request. Radiator will extract the *realm* from the User-Name attribute. Radiator uses <Handler> definitions for routing decisions. Even though routing may seem straight-forward since it is based on a single string, it is unfortunately easy to introduce routing loops. Therefore, special care should be taken to prevent this. There are several approaches to that. The one presented here involves regular expressions. The following example shows these, based on the hypothetical eduroam IdP realm "foo.aq" in Antarctica, and one authoritative RADIUS server for this realm. That same IdP is also an SP and could originate requests. The handler will then look like the following:

```
<Handler Realm=/^foo\.aq$/i,Client-Identifier=/^(?!icecold-radius$)/>
  <AuthBy RADIUS>
    DisableMTUDiscovery
    RetryTimeout      3
    Retries           1
    FailureBackoffTime 300      # (adjust after own needs)
    UseExtendedIds
    <Host 203.0.113.54>
      AuthPort        1812
      AcctPort        1813
      Secret          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    </Host>
  </AuthBy>
  AuthLog TICKS
  AuthLog defaultAuthLog
</Handler>
```

Note the regular expression: it matches only exactly "foo.aq" - not "barfoo.aq" or "foo.aqx". It also contains a safety measure: since the FLR operator can make the link that the realm "foo.aq" is colocated with a eduroam SP whose Client Identifier is "icecold-radius", it can spot that there must be an error if requests for the realm "foo.aq" leave the server in question. Therefore, the Handler clause will only match if the Client-Identifier is NOT "icecold-radius".

If the eduroam IdP provides multiple servers for resiliency reasons, you can specify this in the Handler as well. Please consult the Radiator manual for further details.

Handlers are evaluated in-order, so you should list all known eduroam IdPs one after another in one big block.

You should also add several "catch-all" realms for unknown realms. They are listed below.

### Handling empty realms

Empty realms means User-Name requests that do not carry the @... suffix. In a well-behaved eduroam IdP, empty realms should not reach the FLR server (they would be discarded by the IdP already), but if they do, this following realm definition will catch them and reject the request. A reply will be added to the rejected requests explaining the reason for rejection. Replace <TLD> with the federation top-level domain you are authoritative for.

```
<Handler Realm=/^$/>
  AccountingHandled
  <AuthBy INTERNAL>
    DefaultResult REJECT
    RejectReason Misconfigured client: empty realm! Rejected by <TLD>.
  </AuthBy>
  RejectHasReason
  AuthLog defaultAuthLog
</Handler>
```

### Unknown realms in the own federation

As the FLR server, your server needs to provide authoritative answers for all possible realms under your TLD. This means that all unknown realms need to be rejected by your server. Failure to do so may lead to routing loops!

Add the following stanza (after your Handler sections for valid realms!) to catch and reject all unknown realms that end in your own TLD (obviously replacing the term TLD with your top-level domain):

```
<Handler Realm=/.*\.\.tld$/i>
  AccountingHandled
  <AuthBy INTERNAL>
    DefaultResult REJECT
    RejectReason Misconfigured supplicant or downstream server: uses non-existing realm in <TLD>
    federation!
  </AuthBy>
  RejectHasReason
  AuthLog TICKS
  AuthLog defaultAuthLog
</Handler>
```

### Other known-bad realms

In general, no further second-guessing of incoming realm names should be done. New federations join eduroam every once in a while, and some connected IdPs may reside under "surprising" TLDs (such as .com). That is not a reason to hard-codedly reject all these realms.

However, there are some few well-known, bad, realms that can safely be filtered. The following entry is such an example. For all other realms, please consult the eduroam OT before applying any rejection rules.

One such invalid realm is seen quite often due to supplicant misconfiguration: myabc.com (this is the default realm in an unconfigured Intel PRO/Set Wireless supplicant). The following stanza rejects this realm with an appropriate error message and blindly acknowledges all Accounting requests.

```

<Handler Realm=/myabc\.com$/i>
  AccountingHandled
  <AuthBy INTERNAL>
    DefaultResult REJECT
    RejectReason Misconfigured client: default realm of Intel PRO/Wireless supplicant! Rejected
  by <TLD>.
    </AuthBy>
    RejectHasReason
  AuthLog TICKS
  AuthLog defaultAuthLog
</Handler>

```

### Realms from other federations

This is the last Handler rule: it forwards all requests that haven't matched any previous Handler and determines the routing destination. It will first attempt to discover whether there is a direct RADIUS/TLS route to the destination realm's server, and if not, route the request to the ETLRs.

```

<Handler User-Name = /\@/>
  <AuthBy DNSROAM>
    Port 2083
    Protocol radsec
    Transport tcp
    UseTLS 1
    Secret radsec
    ReconnectTimeout 1
    NoreplyTimeout 5
    ConnectOnDemand
    TLS_CAPath /etc/radiator/certs/CAs/current/
    TLS_CertificateFile /etc/radiator/certs/server.pem
    TLS_CertificateType PEM
    TLS_PrivateKeyFile /etc/radiator/certs/server.key
    TLS_PolicyOID .1.3.6.1.4.1.25178.3.1.2
    TLS_ExpectedPeerName CN=.*
    <Route>
      Realm DEFAULT
      Address etlrl.eduroam.org
      Port 2083
      Transport tcp
      Protocol radsec
    </Route>
  </AuthBy>
  AuthLog TICKS
</Handler>

```

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.

## Goodies

### Local logging of auths in one line

It is useful to log each authentication locally, with more detail than is needed for F-Ticks. We suggest using the following log definition – it generates one single line of log output per authentication, which is very parser-friendly if logs need to be evaluated later:

```

<AuthLog SYSLOG>
  Identifier          defaultAuthLog
  Facility            local7
  LogIdent            radiator
  FailureFormat       Access-Reject for %u (User-Name=%{Reply:User-Name}) at Proxy=%c (CSI=%{Calling-
Station-Id}NAS=%{NAS-Identifier}/%N)
  SuccessFormat       Access-Accept for %u (User-Name=%{Reply:User-Name}) at Proxy=%c (CSI=%{Calling-
Station-Id}NAS=%{NAS-Identifier}/%N) EAP=%{HexAddress:EAP-Message}
  LogSuccess          1
  LogFailure          1
</AuthLog>

```

## SNMP

You may want to configure SNMP access to your server. SNMP allows remote monitoring of activity on a RADIUS server with tools such as RADAR from OSC (<http://www.open.com.au/radar/index.html>), or drawing simple graphs of activity by rgraph from CESNET (<http://www.eduroam.cz/rgraph/>).

```

<SNMPAgent>
  ROCommunity xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  Managers     localhost 127.0.0.1
</SNMPAgent>

```

## Caveats

The previous sections have referenced two specific RADIUS attributes, "Operator-Name" and "eduroam-SP-Country". In Radiator 4.7, these attributes aren't shipped by default and need to be registered in the server's so-called "dictionary".

Operator-Name, and a few more attributes, is defined in the IETF document RFC5580. The definitions in there are canonical, but they clash with the dictionary that's shipped with Radiator, so you will have to remove a few bogus entries, and then add the correct definitions. Please open the "dictionary" and make the following edits:

**Delete** the following bogus entries near line 230 of the dictionary file:

ATTRIBUTE	Ascend-Route-Preference	126	integer
ATTRIBUTE	Tunneling-Protocol	127	integer
ATTRIBUTE	Ascend-Shared-Profile-Enable	128	integer
ATTRIBUTE	Ascend-Primary-Home-Agent	129	string
ATTRIBUTE	Ascend-Secondary-Home-Agent	130	string
ATTRIBUTE	Ascend-Dialout-Allowed	131	integer
ATTRIBUTE	Ascend-Client-Gateway	132	ipaddr

**Replace** them with the following definitions:

ATTRIBUTE	Operator-Name	126	string
ATTRIBUTE	Location-Information	127	string
ATTRIBUTE	Location-Data	128	string
ATTRIBUTE	Basic-Location-Policy-Rules	129	string
ATTRIBUTE	Extended-Location-Policy-Rules	130	string
ATTRIBUTE	Location-Capable	131	integer
ATTRIBUTE	Requested-Location-Info	132	integer

The attribute eduroam-SP-Country is a custom extension, a so-called "vendor-specific" attribute. It is registered under the namespace of TERENA. Please add the following definition at the end of the dictionary file if you use a version of Radiator BEFORE 4.9 with the patchset of 04 April 2012. For newer versions of Radiator, this attribute is already shipped by default with the server and you do not have to change anything.

```

# TERENA VSAs
#
VENDOR      TERENA      25178
VENDORATTR  25178      eduroam-SP-Country      10      string

```

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 (the configure switch about F-Ticks is only needed if you need that functionality):

```
./configure --enable-fticks
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. Therefore, a template configuration file is provided at <http://www.eduroam.org/downloads/docs/eduroam-cookbook-scripts.zip>. A detailed explanation of this configuration file follows. However, the comments included in the file should make its action almost self-explanatory. This means you can start and experiment with it right after installation.

## Base configuration / logging / F-Ticks

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

As a National Roaming Operator, 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 if you compiled it with the

```
--enable-fticks
```

option. If that is done, 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, but excludes the international roaming top-level servers (e.g. the European Top-Level RADIUS Servers). 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 AQ                # will generate F-Ticks for "visited country = Antarctica"
}

```

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 deliberately 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 use 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 /^$/ {
    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. *\_DP\_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$ {
    replaymessage "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.

```
realm * {
    server etlr1.eduroam.org
    server etlr2.eduroam.org
}
```

## 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

### FreeRADIUS 3 - RADSEC

This section describes how to set up FreeRADIUS to handle RADSEC as a federation-level RADIUS and RADIUS/TLS server. It can then completely replace other RADSEC proxy products on the federation level (i.e. if you already have FreeRADIUS you can simply activate this virtual server and you'll be able to handle RADSEC - RADIUS/TLS over TCP).

More precisely, it will enable a server to:

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

## Version information

The prerequisites for this deployment are:

- FreeRADIUS version 3.0.0 or higher
- A server certificate and a private key for that certificate to establish the RadSec connection which designates the server as an IdP+SP.

## Sample config file

All of the RADSEC configuration for FreeRADIUS 3.x can be in a single virtual server file. A detailed explanation of this configuration file is not yet provided. However, the comments included in the file should make its action almost self-explanatory. This means you can start and experiment with it right after installation.

## Installation

Simply copy and paste this code into a new virtual server e.g. eduroam-radsec and place into your \$RADDB/sites-enabled directory

```
listen {
    ipaddr = *
    port = 2083
    type = auth

    # For now, only TCP transport is allowed.
    proto = tcp
}
```

```

clients = radsec

# This is *exactly* the same configuration as used by the EAP-TLS
# module. It's OK for testing, but for production use it's a good
# idea to use different server certificates for EAP and for RADIUS
# transport.
tls {
    # These are used to simplify later configurations.
    certdir = ${confdir}/radsec
    cadir = ${confdir}/radsec

    private_key_password = whatever
    private_key_file = ${certdir}/server.realm.tld-key.pem

    # If Private key & Certificate are located in
    # the same file, then private_key_file &
    # certificate_file must contain the same file
    # name.
    #
    # If CA_file (below) is not used, then the
    # certificate_file below MUST include not
    # only the server certificate, but ALSO all
    # of the CA certificates used to sign the
    # server certificate.
    certificate_file = ${certdir}/server.realm.tld-eduPKI.pem

    # Trusted Root CA list
    #
    # ALL of the CA's in this list will be trusted
    # to issue client certificates for authentication.
    #
    # In general, you should use self-signed
    # certificates for 802.1x (EAP) authentication.
    # In that case, this CA file should contain
    # *one* CA certificate.
    #
    # This parameter is used only for EAP-TLS,
    # when you issue client certificates. If you do
    # not use client certificates, and you do not want
    # to permit EAP-TLS authentication, then delete
    # this configuration item.
    CA_file = ${cadir}/eduPKI-CA.crt

    #
    # For DH cipher suites to work, you have to
    # run OpenSSL to create the DH file first:
    #
    #     openssl dhparam -out certs/dh 1024
    #
    dh_file = ${certdir}/dh
    random_file = ${certdir}/random

    #
    # This can never exceed the size of a RADIUS
    # packet (4096 bytes), and is preferably half
    # that, to accomodate other attributes in
    # RADIUS packet. On most APs the MAX packet
    # length is configured between 1500 - 1600
    # In these cases, fragment size should be
    # 1024 or less.
    #
    fragment_size = 1024

    # include_length is a flag which is
    # by default set to yes If set to
    # yes, Total Length of the message is
    # included in EVERY packet we send.
    # If set to no, Total Length of the
    # message is included ONLY in the
    # First packet of a fragment series.
    #

```

```

include_length = yes

# Check the Certificate Revocation List
#
# 1) Copy CA certificates and CRLs to same directory.
# 2) Execute 'c_rehash <CA certs&CRLs Directory>'.
# 'c_rehash' is OpenSSL's command.
# 3) uncomment the line below.
# 5) Restart radiusd
#     check_crl = yes
#     CA_path = ${cadir}

#
# If check_cert_issuer is set, the value will
# be checked against the DN of the issuer in
# the client certificate.  If the values do not
# match, the certificate verification will fail,
# rejecting the user.
#
# In 2.1.10 and later, this check can be done
# more generally by checking the value of the
# TLS-Client-Cert-Issuer attribute.  This check
# can be done via any mechanism you choose.
#
# this doesnt work yet
# check_cert_issuer = "/DC=org/DC=edupki/CN=eduPKI"

#
# If check_cert_cn is set, the value will
# be xlat'ed and checked against the CN
# in the client certificate.  If the values
# do not match, the certificate verification
# will fail rejecting the user.
#
# This check is done only if the previous
# "check_cert_issuer" is not set, or if
# the check succeeds.
#
# In 2.1.10 and later, this check can be done
# more generally by checking the value of the
# TLS-Client-Cert-CN attribute.  This check
# can be done via any mechanism you choose.
#
#     check_cert_cn = %{User-Name}
#
# Set this option to specify the allowed
# TLS cipher suites.  The format is listed
# in "man 1 ciphers".
cipher_list = "DEFAULT"

#

# This configuration entry should be deleted
# once the server is running in a normal
# configuration.  It is here ONLY to make
# initial deployments easier.
#
#
# This is enabled in eap.conf, so we don't need it here.
#
# make_cert_command = "${certdir}/bootstrap"

#
# Session resumption / fast reauthentication
# cache.
#
# The cache contains the following information:
#
# session Id - unique identifier, managed by SSL
# User-Name - from the Access-Accept
# Stripped-User-Name - from the Access-Request

```

```

# Cached-Session-Policy - from the Access-Accept
#
# The "Cached-Session-Policy" is the name of a
# policy which should be applied to the cached
# session. This policy can be used to assign
# VLANs, IP addresses, etc. It serves as a useful
# way to re-apply the policy from the original
# Access-Accept to the subsequent Access-Accept
# for the cached session.
#
# On session resumption, these attributes are
# copied from the cache, and placed into the
# reply list.
#
# You probably also want "use_tunneled_reply = yes"
# when using fast session resumption.
#
cache {
#
# Enable it. The default is "no".
# Deleting the entire "cache" subsection
# Also disables caching.
#
# You can disallow resumption for a
# particular user by adding the following
# attribute to the control item list:
#
#           Allow-Session-Resumption = No
#
# If "enable = no" below, you CANNOT
# enable resumption for just one user
# by setting the above attribute to "yes".
#
enable = yes

#
# Lifetime of the cached entries, in hours.
# The sessions will be deleted after this
# time.
#
lifetime = 24 # hours

#
# The maximum number of entries in the
# cache. Set to "0" for "infinite".
#
# This could be set to the number of users
# who are logged in... which can be a LOT.
#
max_entries = 255
}

#
# Require a client certificate.
#
require_client_cert = yes

#
# As of version 2.1.10, client certificates can be
# validated via an external command. This allows
# dynamic CRLs or OCSP to be used.
#
# This configuration is commented out in the
# default configuration. Uncomment it, and configure
# the correct paths below to enable it.
#
verify {
# A temporary directory where the client
# certificates are stored. This directory
# MUST be owned by the UID of the server,
# and MUST not be accessible by any other

```

```

# users. When the server starts, it will do
# "chmod go-rwx" on the directory, for
# security reasons. The directory MUST
# exist when the server starts.
#
# You should also delete all of the files
# in the directory when the server starts.
    tmpdir = /etc/raddb/temporary

# The command used to verify the client cert.
# We recommend using the OpenSSL command-line
# tool.
#
# The ${..CA_path} text is a reference to
# the CA_path variable defined above.
#
# The %{TLS-Client-Cert-Filename} is the name
# of the temporary file containing the cert
# in PEM format. This file is automatically
# deleted by the server when the command
# returns.

# this doesnt work yet either
    #client = "/usr/bin/openssl verify -CAfile /etc/raddb/radsec/eduPKI-CA.crt -purpose
crlsign  %{TLS-Client-Cert-Filename}"
    }
}

# IPv6 listener - config comments cleared for brevity

listen {
    ipv6addr = ::
    port = 2083
    type = auth
    proto = tcp
    clients = radsec

    tls {
        certdir = ${confdir}/radsec
        cadir = ${confdir}/radsec
        private_key_password = whatever
        private_key_file = ${certdir}/server.realm.tld-key.pem
        certificate_file = ${certdir}/server.realm.tld-eduPKI.pem
        CA_file = ${cadir}/eduPKI-CA.crt
        dh_file = ${certdir}/dh
        random_file = ${certdir}/random
        fragment_size = 1024
        include_length = yes
        # this doesnt work yet
        # check_cert_issuer = "/DC=org/DC=edupki/CN=eduPKI"

        cipher_list = "DEFAULT"
        cache {
            enable = yes
            max_entries = 255
        }
        require_client_cert = yes
        verify {
            tmpdir = /etc/raddb/temporary
            # doesnt work yet
            #client = "/usr/bin/openssl verify -CAfile /etc/raddb/radsec/eduPKI-CA.crt -purpose
crlsign  %{TLS-Client-Cert-Filename}"
        }
    }
}

clients radsec {
    client 127.0.0.1 {
        ipaddr = 127.0.0.1
        proto = tcp

```

```

        secret = testing123
    }
    client etlr1.eduroam.org {
        ipaddr = 192.87.106.34
        proto = tcp
        secret = radsec
    }
    client etlr2.eduroam.org {
        ipaddr = 130.225.242.109
        proto = tcp
        secret = radsec
# IPv6 for ETRL too - unfamiliar with details, so commented out
#     client etlr1-v6.eduroam.org {
#         ipv6addr = ?????????????????????????????????????????
#         proto = tcp
#         secret = radsec
#     }
#     client etlr2-v6.eduroam.org {
#         ipv6addr = ?????????????????????????????????????????
#         proto = tcp
#         secret = radsec
#     }
}

# local test listener for debug (present by default)
listen {
    ipaddr = 127.0.0.1
    port = 4000
    type = auth
}

home_server etlr1 {
    ipaddr etlr1.eduroam.org
    port = 2083
    type = auth
    secret = radsec
    proto = tcp
    status_check = status-server

    tls {
        #
        # These are used to simplify later configurations.
        #
        certdir = ${confdir}/radsec
        cadir = ${confdir}/radsec

        private_key_password = whatever
        private_key_file = ${certdir}/server.realm.tld-key.pem

        # If Private key & Certificate are located in
        # the same file, then private_key_file &
        # certificate_file must contain the same file
        # name.
        #
        # If CA_file (below) is not used, then the
        # certificate_file below MUST include not
        # only the server certificate, but ALSO all
        # of the CA certificates used to sign the
        # server certificate.
        certificate_file = ${certdir}/server.realm.tld-eduPKI.pem

        # Trusted Root CA list
        #
        # ALL of the CA's in this list will be trusted
        # to issue client certificates for authentication.
        #
        # In general, you should use self-signed
        # certificates for 802.1x (EAP) authentication.
        # In that case, this CA file should contain

```

```

# *one* CA certificate.
#
# This parameter is used only for EAP-TLS,
# when you issue client certificates. If you do
# not use client certificates, and you do not want
# to permit EAP-TLS authentication, then delete
# this configuration item.
CA_file = ${cadir}/eduPKI-CA.crt

#
# For DH cipher suites to work, you have to
# run OpenSSL to create the DH file first:
#
#         openssl dhparam -out certs/dh 1024
#
dh_file = ${certdir}/dh
random_file = ${certdir}/random

#
# This can never exceed the size of a RADIUS
# packet (4096 bytes), and is preferably half
# that, to accomodate other attributes in
# RADIUS packet. On most APs the MAX packet
# length is configured between 1500 - 1600
# In these cases, fragment size should be
# 1024 or less.
#
fragment_size = 1024

# include_length is a flag which is
# by default set to yes If set to
# yes, Total Length of the message is
# included in EVERY packet we send.
# If set to no, Total Length of the
# message is included ONLY in the
# First packet of a fragment series.
#
include_length = yes

# Check the Certificate Revocation List
#
# 1) Copy CA certificates and CRLs to same directory.
# 2) Execute 'c_rehash <CA certs&CRLs Directory>'.
# 'c_rehash' is OpenSSL's command.
# 3) uncomment the line below.
# 5) Restart radiusd
#
# check_crl = yes
CA_path = ${cadir}

#
# If check_cert_issuer is set, the value will
# be checked against the DN of the issuer in
# the client certificate. If the values do not
# match, the certificate verification will fail,
# rejecting the user.
#
# In 2.1.10 and later, this check can be done
# more generally by checking the value of the
# TLS-Client-Cert-Issuer attribute. This check
# can be done via any mechanism you choose.
#
# check_cert_issuer = "/C=GB/ST=Berkshire/L=Newbury/O=My Company Ltd"

#
# If check_cert_cn is set, the value will
# be xlat'ed and checked against the CN
# in the client certificate. If the values
# do not match, the certificate verification
# will fail rejecting the user.
#
# This check is done only if the previous

```

```

# "check_cert_issuer" is not set, or if
# the check succeeds.
#
# In 2.1.10 and later, this check can be done
# more generally by checking the value of the
# TLS-Client-Cert-CN attribute. This check
# can be done via any mechanism you choose.
#
# check_cert_cn = %{User-Name}
#
# Set this option to specify the allowed
# TLS cipher suites. The format is listed
# in "man 1 ciphers".
cipher_list = "DEFAULT"
}
}

#second home server config cleared to the values required for brevity

home_server etlr2 {
    ipaddr etlr2.eduroam.org
    port = 2083
    type = auth
    secret = radsec
    proto = tcp
    status_check = status-server

    tls {
        certdir = ${confdir}/radsec
        cadir = ${confdir}/radsec
        private_key_password = whatever
        private_key_file = ${certdir}/server.realm.tld-key.pem
        certificate_file = ${certdir}/server.realm.tld-eduPKI.pem
        CA_file = ${cadir}/eduPKI-CA.crt
        dh_file = ${certdir}/dh
        random_file = ${certdir}/random
        fragment_size = 1024
        include_length = yes
        cipher_list = "DEFAULT"
    }
}

home_server_pool ETLR {
    type = load-balance
    home_server = etlr1
    home_server = etlr2
}

realm eduroam {
    auth_pool = ETLR
}

```

## Caveats

Currently (10th June 2011) there are some bugs with handling unreachable remote proxies which causes the daemon to die. A few of these have already been dealt with via bug reports but some still lurk. Also, the certificate checking/verification code does not currently work - we hope to be able to verify the certificate issuer and OID as we do with RADIUS and RadSecProxy. Note that this software only does RADSEC/TLS with TCP - DTLS over UDP is not yet an option. Clients are 'radsec' only and the standard naslist or naslist imported from SQL won't operate with radsec.

## VitalAAA

To set up a federation-level RADIUS proxy server for VitalAAA you must change the following configuration files:

- server\_properties

- method\_dispatch
- clients

You must also download the following files from <http://www.eduroam.org/downloads/docs/eduroam-cookbookscrips.zip>:

- aaa.pf
- error.pf
- proxy.pf
- prepare.pfserver\_properties file:

```
Radius-Acct-Address = "*:1813"
Radius-Auth-Address = "*:1812"
Database-Address = "0"
Radius-CharSet = UTF8
Delimiter-Precedence = "@"
Suffix-Delimiters = ""
```

method\_dispatch file:

```
radius          Auth 1          prepare          setWorkingVars
radius          acct 4          aaa              dropRadiusAcct
```

*clients file*

Add the lines with the eduroam proxy server and the local RADIUS servers to the *clients* file:

```
192.87.106.34      <eduroam_secret>
130.225.242.109   <eduroam_secret>
<192.168.1.10>    <local_server_secret>
<192.168.1.20>    <local_server_secret>
```

## Gauging your federation's performance

### Monitoring

It is important to constantly monitor your infrastructure on all levels, in order to react to system failure and see upcoming problems. There is a multitude of monitoring solutions on the market, and it is not possible to describe ways to monitor eduroam infrastructure for all of them; but we have provided a selection below.

First, for Europe, some parts of monitoring are done by the eduroam Operation Team which we will describe in the following section; please contact your own regional operator for the corresponding monitoring solution in your area if you are operating outside Europe.

In the then-following sections, we provide general tips for infrastructure monitoring.

### Federation monitoring in Europe: the eduroam Operational Team

When you set up a federation-level RADIUS server, the OT will start monitoring your server availability and will send out email alerts in case of failure. This is done by the OT sending authentication requests for the special realm @eduroam.<TLD> from their monitoring server to your server, and your server is expected to mirror these back to the OT monitoring infrastructure. The technical set-up of this is described in the corresponding HOWTOs for federation-level RADIUS servers.

Server availability is tested every hour and the results are summarised on the following web page: <http://monitor.eduroam.org/>

Note that you can also get more detailed info, including a history, by navigating on the left-hand pane on that website.

There is also a more detailed diagnosis test, where a federation operator can request that a specific path (i.e. from federation A via the European root to federation B) is tested real-time on-demand. The web interface for this testing facility is online at: [http://monitor.eduroam.org/inter/test\\_otm.php](http://monitor.eduroam.org/inter/test_otm.php) (access is restricted to eduroam federation operators only).

### Monitoring inside the federation

There are several dimensions to infrastructure monitoring; most of which are unrelated to eduroam: system utilisation, hardware health, network reachability, a.s.o. There are many market solutions to monitor these aspects. It is beneficial to use a monitoring solution which can use plugins to execute some more eduroam-specific monitoring. Nagios and its fork Icinga have proven to be valuable to many eduroam participants, and the following plugins are considered useful.

## Nagios/Icinga: EAP Login checks

### Preparatory work

The tool "rad\_eap\_test", which is a frontend to wpa\_supplicant's "eapol\_test", can be used for scripted authentication checks in Nagios. The added value over eapol\_test is that eapol\_test requires a configuration file on disk by the time of execution. rad\_eap\_test is completely command-line driven; it generates a temporary configuration file and deletes it again after eapol\_test execution.

You can download rad\_eap\_test from here: [http://www.eduroam.cz/rad\\_eap\\_test/](http://www.eduroam.cz/rad_eap_test/)

It requires eapol\_test, part of wpa\_supplicant from here: <http://hostap.epitest.fi/>

To compile eapol\_test, unpack the wpa\_supplicant distribution, change into the wpa\_supplicant/ subdirectory and create the default config file by executing

```
cp defconfig .config
```

Then, enable compilation of eapol\_test by editing the .config file and setting (i.e. uncommenting)

```
CONFIG_EAPOL_TEST=y
```

You can then compile eapol\_test with

```
make eapol_test
```

Now, you need to tell the shell script rad\_eap\_test where to find the eapol\_test executable; and tell the eduroam F-Ticks system that these are monitoring-only requests by setting a corresponding MAC address. Edit the rad\_eap\_test file and replace the lines

```
EAPOL_PROG=<your path to eapol_test here>  
MAC="22:44:66:xx:yy:zz" (replace x,y,z with arbitrary values to your liking)
```

That's it for the prerequisites - we can now start defining Nagios/Icinga checks.

### Implementing the checks

You would typically execute the Nagios checks by defining your Nagios server as a client to your FLR server, and send requests for known test accounts of your realms to that server.

You can define check commands like the following:

```
define command{  
    command_name    check_eduroam_login  
    command_line    $USER1$/rad_eap_test -H <your FLR hostname> -P 1812 -S <shared secret for your Nagios  
client> -A $ARG1$ -u $ARG2$ -p $ARG3$ -e $ARG4$ -m WPA-EAP -t 7  
}
```

```
}
```

and later use the arguments as follows in your individual checks:

- ARG1 = anonymous outer identity
- ARG2 = inner username
- ARG3 = password
- ARG4 = EAP type (TTLS/PEAP)

You can also define similar checks for other EAP types; simply execute rad\_eap\_test without arguments to see which parameters it supports.

**Example:** You want to test a participating realm foobar.aq which uses PEAP, and for which you have the test credentials "testuser" and "testpass", and you want to test whether anonymous outer identities work properly. The corresponding service check is:

```
define service{
    use                generic-service
    host_name          <your FLR server>
    service_description EDUROAM_FOOBAR
    contact_groups     ...
    check_command      check_eduroam_login!@foobar.aq!testuser@foobar.aq!testpass!PEAP
}
```

## Nagios/Icinga: RADIUS/TLS certificate validity checks

You can use the commodity Nagios plugin "check\_ssl\_cert" from: [https://trac.id.ethz.ch/projects/nagios\\_plugins/wiki/check\\_ssl\\_cert](https://trac.id.ethz.ch/projects/nagios_plugins/wiki/check_ssl_cert) for this purpose. The check command is then:

```
define command {
    command_name = radius_tls
    command_line = $USER1$/check_ssl_cert --host $HOSTADDRESS$ --port 2083 --noauth --warning 14 --critical
}
3
```

and will warn you two weeks in advance that your certificate is about to expire when added to the host as a service check.

## Statistics

It is also important to measure how successful the service is in your area of responsibility. eduroam Operations has set up a statistics system called F-Ticks, which is able to capture all roaming events both on a national as well as an international level. It does not cover local campus usage though.

If your FLR server is configured to support F-Ticks (it is, if configured according to this cookbook), statistics will be generated automatically for that federation. They are accessible at the following website: <http://monitor.eduroam.org/f-ticks/>

On that web page, you can find historical evolution of roaming service usage in federations, as well as an overview which realms were most active, and from which countries visitors come from. In the future, detailed views per SP and per IdP can be made available if your federation opts to send the data in the extended detail level. Please contact your federation operator to find out which level of statistics your federation provides.

If you have configured your federation

## RADIUS/TLS: Obtaining and managing certificates

RADIUS over TLS is a new way of interconnecting federations (and later, if desired, eduroam IdPs and eduroam SPs). It uses TLS encryption instead of IP address and shared secret pairs to authenticate and authorise eduroam servers. When replacing such explicit configuration-based authorisation with a dynamic, automatic provisioning model, it is important to clearly define the rules for issuance of an eduroam server certificate, because the possession of the certificate will enable the holder to participate in eduroam.

In order to make use of this new feature, your FLR server must have acquired an eduroam server certificate. Depending on which federation or world region you are from, the procedures for getting a certificate will differ. The following two subsections are a globally valid description of the eduroam Trust Model. This trust model is currently only implemented by one CA, which operates in Europe. The last subsection provides details for European eduroam participants.

### The eduroam server certificate trust model: eduPKI PMA and the eduroam Trust Profile

During the design of the X.509 trust model for eduroam, certain requirements had to be considered.

- It became clear that no single one Certificate Authority (CA) can or should issue all eduroam certificates world-wide. Instead, rules were defined under which multiple CAs can issue eduroam certificates.
- These CAs could possibly be general-purpose CAs that also manage certificates for other services besides eduroam. Consequently, the eduroam trust model had to allow to differentiate eduroam server certificates from other certificates from the same CA in a standardised manner.
- A CA would need to conform to certain quality assurance criteria, which need to be assessed by an oversight committee.

As a result of these requirements, the GEANT project's eduPKI task created a framework for the eduroam trust model:

- an oversight body, the "eduPKI Policy Management Authority" (eduPKI-PMA) was created and produced a document with defined Quality Assurance criteria for CAs which would like to become part of the eduroam trust model. The rules for CA accreditation are set forth in section "CA

Accreditation Process' at <https://www.edupki.org/edupki-pma/pma-governing-documents/>. Note that eduPKI PMA is currently the only PMA, but this doesn't preclude other PMAs in other world regions from emerging.

- a X.509 trust profile for the eduroam service was created, which designates two so-called "policy OID" fields to eduroam IdP and SP servers. The trust profile can be found on this page: <https://www.edupki.org/edupki-pma/edupki-trust-profiles/>
- this trust profile requires that CAs which use this policy OID will check the authorisation of a certificate applicant whether or not he is actually an eduroam IdP and/or SP server operator.

This way, it can be assured that only authorised eduroam operators get eduroam certificates and can establish connections to other eduroam servers.

## Managing accredited CAs in eduroam servers

The number of accredited CAs and the list of certificates can change at any time. It is important that all eduroam servers consult an up-to-date list of accredited CAs. The list of currently accredited CAs is maintained in a TERENA repository of the TACAR service. A browsable list can be found here: <https://www.tacar.org/cert/list/>

Please refrain from manually downloading CAs as a one-time action. Otherwise, your CA list will eventually become outdated and this will create service disruption for some eduroam users!

There is currently one accredited Certification Authority: the eduPKI CA, located at <https://www.edupki.org/edupki-ca/>. eduPKI CA acts as a catch-all world-wide for eduroam participant countries which do not have their own accredited CA for the eduroam service. Such further CAs are welcome to apply for eduPKI PMA accreditation.

eduroam operators should request their eduPKI CA eduroam certificate by following the instructions on the eduPKI CA eduroam RA pages at: <http://www.eduroam.org/index.php?p=europe&s=edupki>

## Updating CRLs on your server

Since certificate possibly need to be revoked in case of private key compromise or other reasons, it is important that all RADIUS servers which validate eduroam-accredited CAs consult an up-to-date CRL list for each of the CAs. eduroam suggests to use the script "FetchCRL3" which was developed in the Grid community for this very purpose ([download here](#)).

Usage:

- place the .info files of all accredited CAs into one otherwise empty directory ( download [edupki.info](#) ) - let's assume for this example that the path to those files is

```
/path/to/certificates/
```

- find out the command which restarts your RADIUS server on your system - let's assume for this example that the command is

```
systemctl restart radiusd.service
```

- The following command will attempt to fetch an up-to-date CRL for the CA, and only if successful, will restart your server:  
`fetch-crl -l /path/to/certificates/ && systemctl restart radiusd.service`

- This script should be executed in a cron job on a regular basis (we suggest daily).

## Deployment Checklist

PLEASE ADD TEXT