

FreeRADIUS 3: Konzepte und Konfiguration

09. Oktober 2024

Simon Ruderich

Regionales Rechenzentrum Erlangen (RRZE)
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)



1. RADIUS

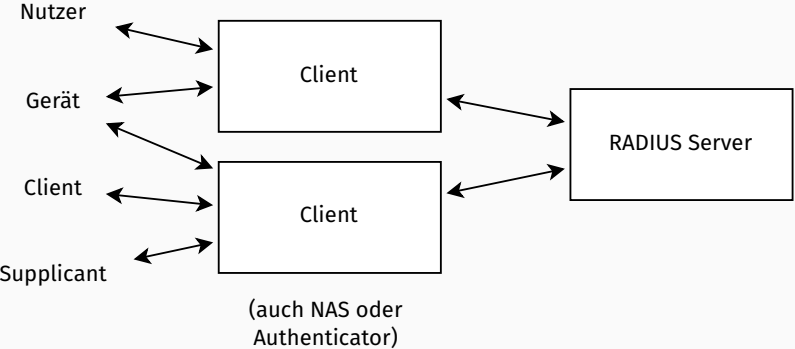
2. FreeRADIUS 3

3. Beispiel: eduroam

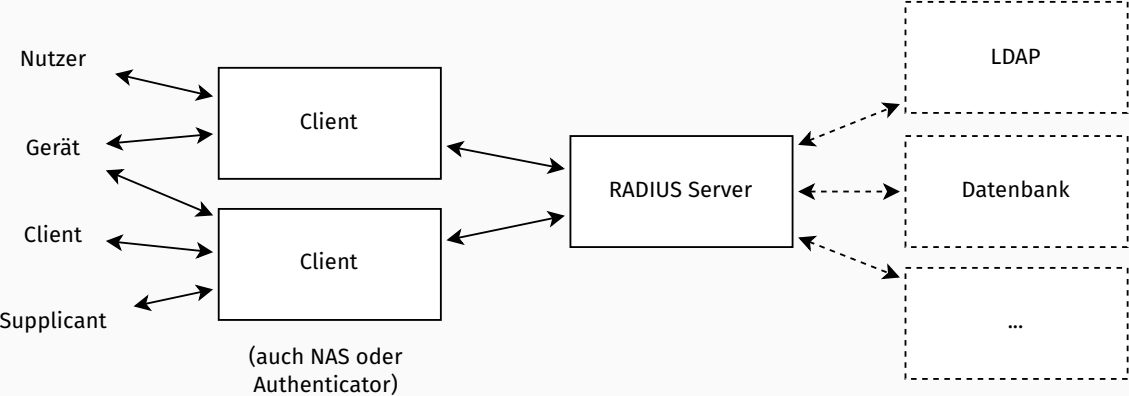
4. Debugging

RADIUS

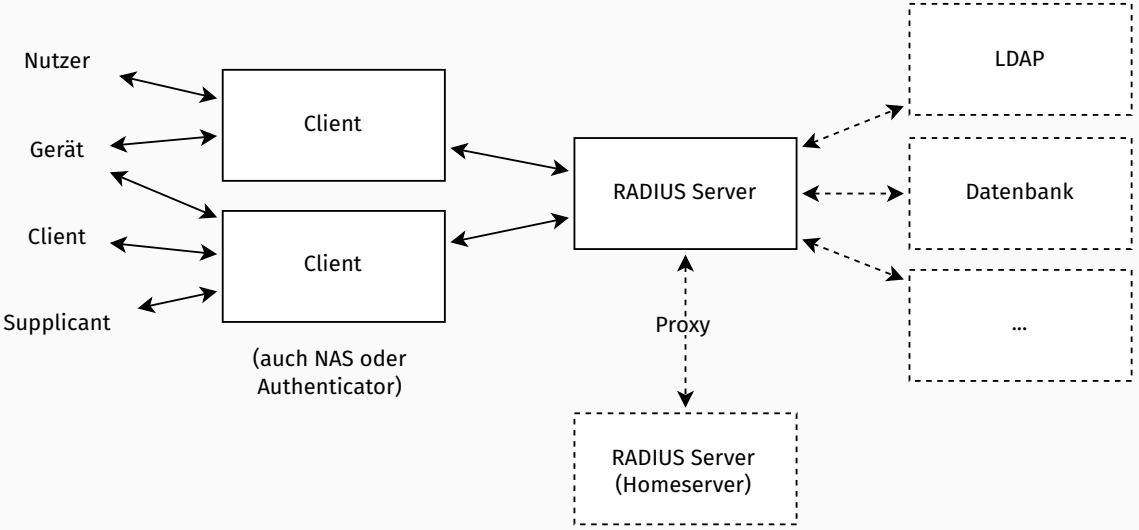
Was ist RADIUS?



Was ist RADIUS?

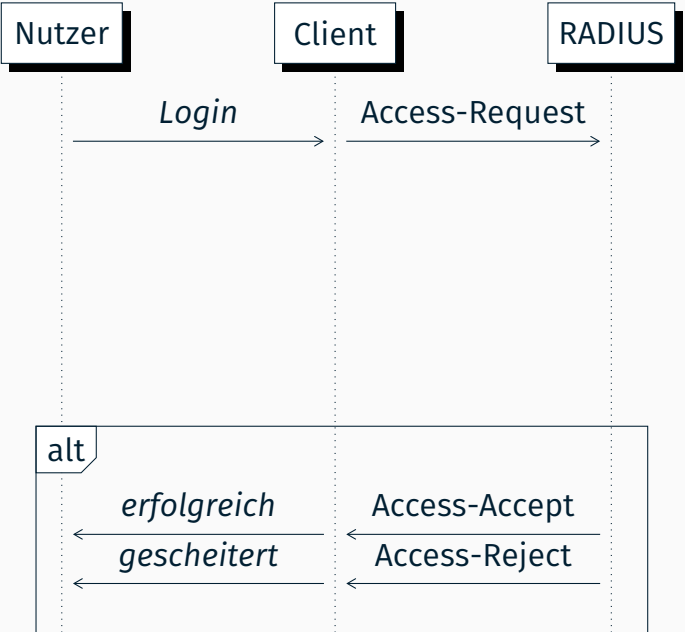


Was ist RADIUS?

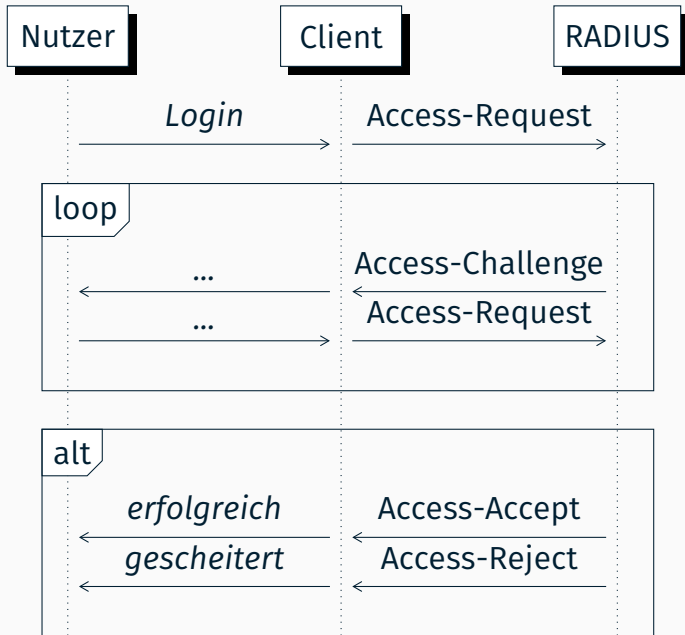


- **RADIUS:** Remote Authentication Dial-In User Service
- **AAA:** Authentication, Authorization, Accounting
- **Supplicant** oder **Endgerät:** Meldet sich für Ressource an
- **Client** oder **Network Access Server (NAS)** oder **Authenticator:** Bietet Ressource für Nutzer an (bspw. WLAN-Controller, VPN-Server)
- **Authentication Server:** RADIUS-Server, bspw. FreeRADIUS
- **Homeserver:** Anderer RADIUS-Server, an den Anfragen weiter geleitet werden

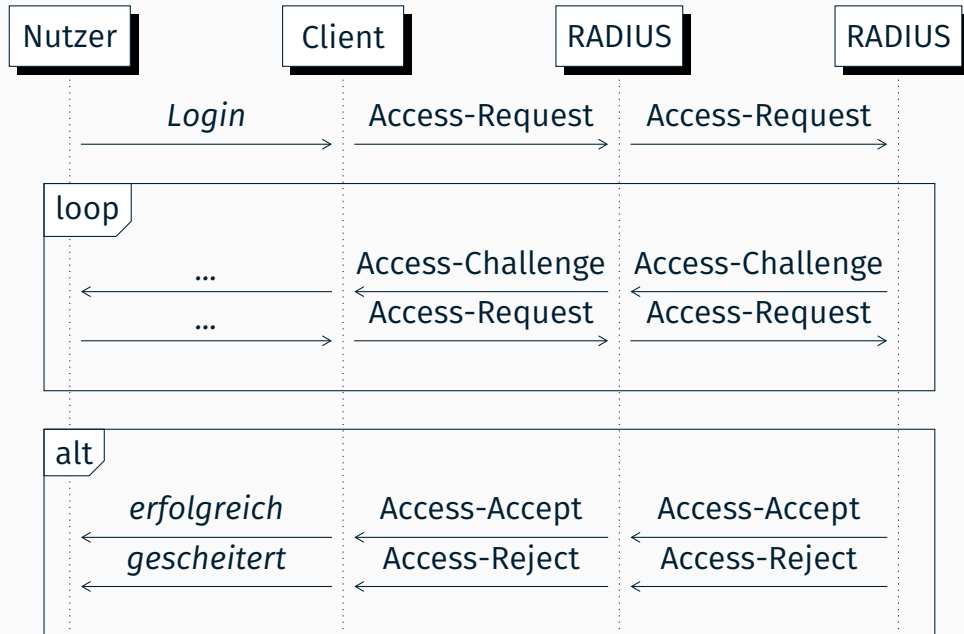
Paketfluss: Authentifizierung



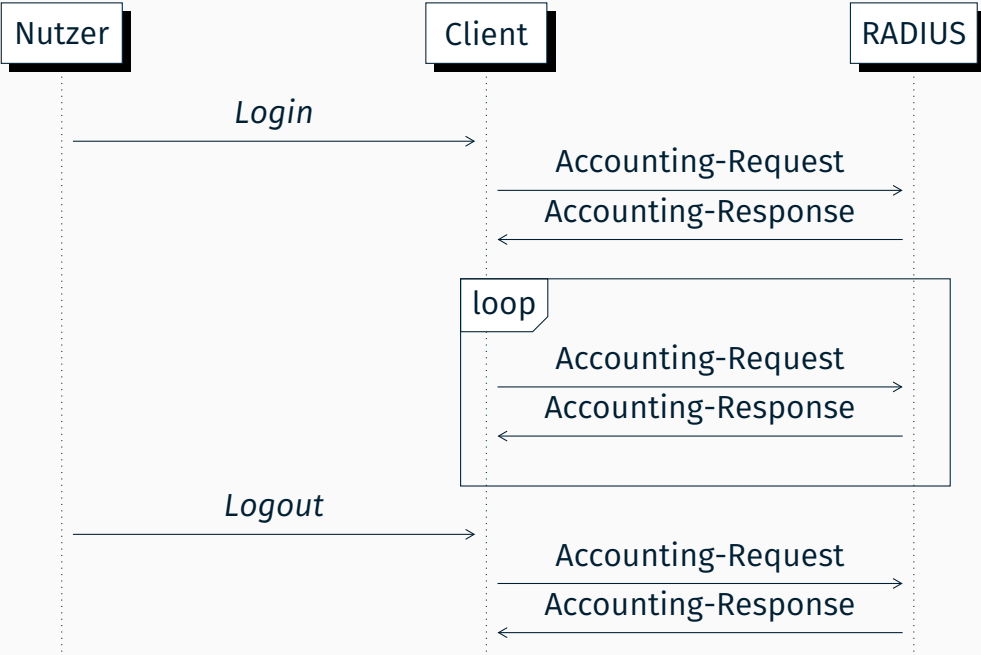
Paketfluss: Authentifizierung



Paketfluss: Authentifizierung mit Proxy



Paketfluss: Accounting

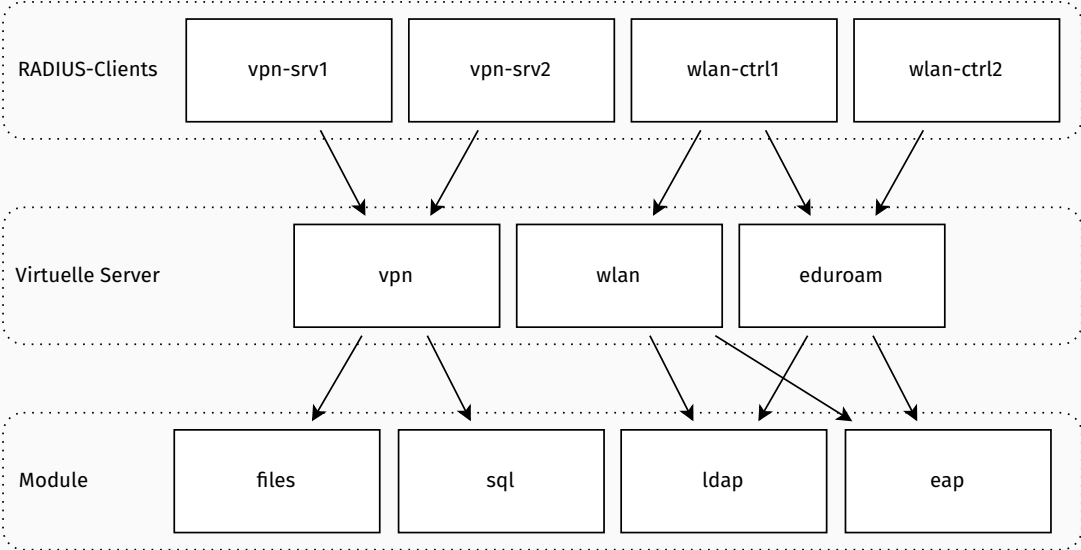


FreeRADIUS 3

- <https://www.freeradius.org/>
- Erstes Release: August 1999; aktuell ist Version 3.2
- Lizenz: GPLv2

- Konfiguration in `/etc/freeradius/3.0/` oder `/etc/raddb/`
- Generell hilft es Kommentare dort zu lesen
(oft die einzige Dokumentation)

Struktur



Clients

- Clients werden in `clients.conf` konfiguriert

```
client vpn1 {  
    ipaddr          = 192.0.2.42  
    secret          = ...  
    virtual_server  = vpn  
}  
client switches {  
    ipaddr          = 198.51.100.0  
    netmask         = 24  
    secret          = ...  
    virtual_server  = nac  
}
```

- Mehrere Server-Instanzen in einem FreeRADIUS-Server
- Konfiguration innerhalb des Servers ist lokal, Module und andere Einstellungen sind global
- Konfiguriert in `sites-available/`
- Aktiviert per Symlink in `sites-enabled/`
- Über Port bzw. IP oder `virtual_server` in Clients bzw. Modulen ausgewählt

```
server <name> {  
    ...  
}
```


- Module stellen spezifische Features zur Verfügung
- Konfiguriert in `mods-available/`
- Aktiviert per Symlink in `mods-enabled/`
- Weitere Konfiguration in `mods-config/<modul>/`
- Mehrere Instanzen mit unterschiedlicher Konfiguration möglich

```
ldap ldap_vpn {
    server = 'ldaps://vpn.example.org'
    ...
}
ldap ldap_wlan {
    server = 'ldaps://wlan.example.org'
    ...
}
```

Anfragen verarbeiten: Processing-Sections I

- Liste von Modulen und Unlang-Befehlen
- Werden sequentiell abgearbeitet
- Jedes Modul liefert „return code“ (bspw. `ok`, `not found`, `reject`)
- Bei Fehlern wird abgebrochen (bspw. `reject`, `fail`)
- Standard-Behandlung kann überschrieben werden (siehe `eap` im Beispiel im inneren Tunnel)

```
server <name> {  
  authorize {  
    filter_username  
    preprocess  
    files  
    ldap  
    pap  
  }  
  ...  
}
```

Anfragen verarbeiten: Processing-Sections II

authorize „known good“ Passwort finden,
Auth-Methode setzen oder zum Proxying markieren
(hier keine Antwort-Attribute setzen, erst in **post-auth**)

authenticate Credentials (Nutzer/Passwort) prüfen

post-auth Nach Authentifizierung durchlaufen

Anfragen verarbeiten: Processing-Sections II

authorize „known good“ Passwort finden,
Auth-Methode setzen oder zum Proxying markieren
(hier keine Antwort-Attribute setzen, erst in **post-auth**)

authenticate Credentials (Nutzer/Passwort) prüfen

post-auth Nach Authentifizierung durchlaufen

session Nur für Simultaneous-Use (gleichzeitige Logins)

pre-proxy Bevor Pakete an Homeserver geschickt werden

post-proxy Alle Antworten von Homeservern

- Durchlaufene Processing-Sections
 - Authentifizierung: **authorize, authenticate, session, post-auth**
 - Proxy-Authentifizierung: **authorize, pre-proxy, post-proxy, post-auth**
- Ungenutzte Sections können weggelassen werden (bspw. **session**)
- Falls **accounting** nicht definiert ist, werden Accounting-Request Pakete ignoriert

- **authorize** (präziser wäre *pre-authentication*, intern auch *autz*)

- **authenticate** (intern auch *auth*)

- **authorize** (präziser wäre *pre-authentication*, intern auch *autz*)
 - Module wie `ldap` oder `files` setzen `&control` Attribute, bspw. `&control:Password-With-Header` oder `&control:Cleartext-Password`
 - Andere Module setzen daraufhin `&control:Auth-Type`, bspw. `pap` oder `mschap`
- **authenticate** (intern auch *auth*)

- **authorize** (präziser wäre *pre-authentication*, intern auch *autz*)
 - Module wie `ldap` oder `files` setzen `&control` Attribute, bspw. `&control:Password-With-Header` oder `&control:Cleartext-Password`
 - Andere Module setzen daraufhin `&control:Auth-Type`, bspw. `pap` oder `mschap`
- **authenticate** (intern auch *auth*)
 - Definiert mögliche Auth-Types
 - Führt eigentliche Authentifizierung durch („Passwort prüfen“)
 - Keine normale Processing-Section sondern Tabelle
 - Hier kein Unlang möglich

Anfragen verarbeiten: post-auth

- **post-auth** selbst wird nach erfolgreichem **authenticate** ausgeführt
- Darin können weitere Blöcke angelegt werden, bspw.
Post-Auth-Type REJECT (für Access-Reject)

```
post-auth {  
    # Befehle bei erfolgreichem authenticate (Accept)  
  
    Post-Auth-Type REJECT {  
        # Befehle bei gescheitertem authenticate (Reject)  
    }  
}
```

&request Attribute aus Anfrage

&reply Attribute für Antwort

&control Attribute für FreeRADIUS (bleiben immer lokal)

&session-state Attribute der aktuellen Session (bspw. bei EAP)
(werden automatisch gespeichert/wiederhergestellt)

&request Attribute aus Anfrage

&reply Attribute für Antwort

&control Attribute für FreeRADIUS (bleiben immer lokal)

&session-state Attribute der aktuellen Session (bspw. bei EAP)
(werden automatisch gespeichert/wiederhergestellt)

&proxy-request Attribute für Anfrage an Homeserver
(aus &request kopiert)

&proxy-reply Attribute aus Antwort von Homeserver
(für &reply genutzt)

- Zugriff per Doppelpunkt: `&request:User-Name`
- Fehlt Attributliste, wird `&request` genutzt: `&User-Name`
- `&` an einigen Stellen optional

Besondere Attribute

- `&request:Packet-Src-IP-Address`
IP des Clients, nicht fälschbar (anders als `&NAS-IP-Address`)
- `&request:Packet-Src-IPv6-Address`
Analog für IPv6
- `&request:Virtual-Server`
aktueller virtueller Server
- `&request:Client-Shortname`
aktueller Clientname
- Liste von FreeRADIUS-eigenen Attributen:
`/usr/share/freeradius/dictionary.freeradius.internal`

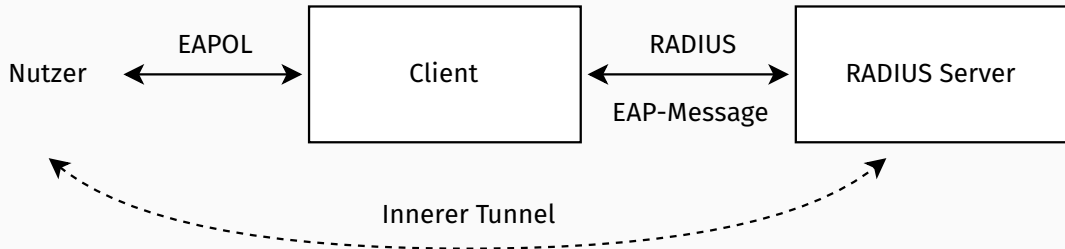
- FreeRADIUS Konfigurationssprache
- `man unlang` und <https://www.freeradius.org/documentation/freeradius-server/3.2.7/unlang/>
- Existiert Attribut?
`if (&Attribut) {}`
- Hat Attribut den Wert?
`if (&Attribut == "Wert") {}`
- Wenn ein Attribut häufiger auftritt, gibt es passenden Wert?
`if (&Attribut[*] == "Wert") {}`

Unlang: Attribute ändern

```
update {  
    # <Attribut> <Operator> <Wert>  
    &reply:User-Name := &session-state:User-Name  
}
```

- = Setze Attribut wenn noch nicht vorhanden
- := Setze Attribut und überschreibe falls vorhanden
- += Füge Attribut am Ende der Liste hinzu
- ^= Füge Attribut am Anfang der Liste hinzu
- = Entferne Attribute mit Wert <Wert>
- !* Entferne Attribute unabhängig vom Wert (ANY als <Wert>)

EAP und innere Tunnel



- EAP-Tunnel ist eigener virtueller Server (im eap-Modul konfiguriert)
- Äußerer Tunnel
 - Anonymer Nutzernamen mit Realm
- Innerer Tunnel
 - Echter Nutzernamen
 - Für Authentifizierung genutzt (Ausnahme ist bspw. EAP-TLS)

- `&outer` ermöglicht Zugriff auf Attribute des äußeren Tunnels
- Bspw. `&outer.session-state:Aruba-User-Group`
- `use_tunneled_reply = yes` sollte nicht mehr verwendet werden (keine präzise Kontrolle was kopiert wird)
- Stattdessen manuell in **post-auth** im äußeren Tunnel kopieren

- Realm ist „Domain“ im Nutzernamen
- Wird von suffix-Modul gesetzt, ohne haben Realms keinen Effekt
- `&User-Name = user@example.org`
→ `&Stripped-User-Name = user, &Realm = example.org`

- Realm ist „Domain“ im Nutzernamen
- Wird von `suffix`-Modul gesetzt, ohne haben Realms keinen Effekt
- `&User-Name = user@example.org`
→ `&Stripped-User-Name = user, &Realm = example.org`
- Realm muss konfiguriert sein, sonst tut `suffix` nichts
- Realms ohne Punkt können von Nutzern nicht gesetzt werden
- Spezielle Realms: `NULL` (ohne Realm), `DEFAULT` (Standard-Realm)
- Proxying kann manuell mit `&control:Proxy-To-Realm` erzwungen werden

- Realm ist „Domain“ im Nutzernamen
- Wird von `suffix`-Modul gesetzt, ohne haben Realms keinen Effekt
- `&User-Name = user@example.org`
→ `&Stripped-User-Name = user, &Realm = example.org`
- Realm muss konfiguriert sein, sonst tut `suffix` nichts
- Realms ohne Punkt können von Nutzern nicht gesetzt werden
- Spezielle Realms: `NULL` (ohne Realm), `DEFAULT` (Standard-Realm)
- Proxying kann manuell mit `&control:Proxy-To-Realm` erzwungen werden
- Homeserver werden per `home_server` und `home_server_pool` definiert
- Und im Realm als `auth_pool` konfiguriert

Beispiel: eduroam

eduroam Outer-Tunnel (1)

```
server eduroam {
  authorize {
    filter_username          # falsche Nutzernamen filtern
    eduroam_auth_log        # extra Logging für eduroam
    rewrite_calling_station_id # Konsistente CSI für Log
    suffix                  # Realm/Stripped-User-Name

    if (!&Realm || &Realm == "NULL") {
      update {
        &request:Module-Failure-Message += \
          "Username without realm forbidden"
      }
      reject
    }
  }
}
```

eduroam Outer-Tunnel (2)

```
# authorize
  if (&Realm == "fau.de") {
    # ID für äußeren und inneren Tunnel für Logs
    if (!&session-state:Tmp-String-9) {
      update {
        &session-state:Tmp-String-9 := \
          "[ID:%{randstr:aaaaaaaaaaaa} \
          CSI:%{Calling-Station-Id}]"
      }
    }
    update {
      &control:Tmp-String-9 := &session-state:Tmp-String-9
    }
    # Braucht Konfiguration in radius.conf unter log {}:
    # msg_goodpass/msg_badpass = "%{%{control:Tmp-String-9}:-}"
  }
```

eduroam Outer-Tunnel (3)

```
# authorize
  # Unterschiedliche CAs
  if (&Stripped-User-Name == "eduroam2023") {
    eap_eduroam2023
  } elsif (&Stripped-User-Name == "eduroam2024") {
    eap_eduroam2024
  } else {
    update {
      &request:Module-Failure-Message += \
        "Invalid anonymous_identity"
    }
    reject
  }
}
```


eduroam Outer-Tunnel (4)

```
# authorize
  # &Realm != "fau.de"
  } else {
    # Nutzer von anderen Einrichtungen werden geproxied
    eap_eduroam2024
  }
}
authenticate {
  Auth-Type eap_eduroam2023 {
    eap_eduroam2023
  }
  Auth-Type eap_eduroam2024 {
    eap_eduroam2024
  }
}
```

eduroam Outer-Tunnel (5)

```
post-auth {
    eduroam_reply_log
    # Attribute für unsere Controller in Antwort kopieren;
    # radsecproxy ist hier Client, keine Attribute leaken;
    # session-state ist im Proxy-Fall leer und hat keinen Effekt
    if (&request:Client-Shortname != "radsecproxy") {
        update {
            &reply:User-Name := &session-state:User-Name
            &reply:Aruba-User-Group := &session-state:Aruba-User-Group
        }
    }
    update {
        &request:Module-Failure-Message += \
            &session-state:Module-Failure-Message
    }
}
```

eduroam Outer-Tunnel (6)

```
# post-auth
# Aus "default" kopiert
remove_reply_message_if_eap
if (&EAP-Key-Name && &reply:EAP-Session-Id) {
    update reply {
        &EAP-Key-Name := &reply:EAP-Session-Id
    }
}
Post-Auth-Type REJECT {
    eduroam_reply_log
    attr_filter.access_reject
    eap_eduroam2024
    remove_reply_message_if_eap
}
}
```

eduroam Outer-Tunnel (7)

```
pre-proxy {
    eduroam_pre_proxy_log
    # Damit klar ist woher die Anfragen kommen, "1" ist wichtig
    update {
        &proxy-request:Operator-Name := "1rrze.de"
    }
    eduroam_attr_filter.pre-proxy # Attribute filtern
}
post-proxy {
    eduroam_post_proxy_log
    eduroam_attr_filter.post-proxy # Attribute filtern
    eap_eduroam2024
}
} # server eduroam
```

eduroam Inner-Tunnel (1)

```
server eduroam-inner {  
listen {  
    ipaddr = 127.0.0.1  
    port = 18120  
    type = auth  
}  
  
authorize {  
    filter_username  
    eduroam_auth_log  
  
    # ID aus äußerem Tunnel übernehmen  
    update {  
        &control:Tmp-String-9 := &outer.session-state:Tmp-String-9  
    }  
}
```

eduroam Inner-Tunnel (2)

```
# authorize
  suffix
  if (&Realm != &outer.Realm) {
    update {
      &request:Module-Failure-Message += \
        "Inner realm doesn't match outer realm"
    }
    reject
  }
# Sicherstellen, dass Anfragen nicht geproxied werden
update {
  &control:Proxy-To-Realm := LOCAL
}
```

eduroam Inner-Tunnel (3)

```
# authorize
  eap_eduroam2024 {
    # Nicht weiter machen, solange EAP noch nicht fertig ist
    ok = return
  }

# LDAP nur einmal aufrufen
if (!&outer.session-state:Aruba-User-Group) {
  ldap_eduroam

  update {
    &outer.session-state:Aruba-User-Group := \
      &control:Aruba-User-Group
  }
}
```

eduroam Inner-Tunnel (4)

```
# authorize
    mschap
    pap
}
}

authenticate {
    Auth-Type eap_eduroam2024 {
        eap_eduroam2024
    }
    Auth-Type EAP {
        eap_eduroam2024
    }
}
```


eduroam Inner-Tunnel (5)

```
Auth-Type MS-CHAP {
    mschap
}
Auth-Type PAP {
    pap
}
}

post-auth {
    eduroam_reply_log

    update {
        # Echte Nutzernamen (nicht anonyme) für WLAN Controller
        &outer.session-state:User-Name := &request:User-Name
    }
}
```

eduroam Inner-Tunnel (6)

```
# post-auth
  Post-Auth-Type REJECT {
    eduroam_reply_log
    attr_filter.access_reject
  }
}

} # server eduroam-inner
```

eduroam EAP

```
eap eap_eduroam2024 {  
    # PEAP wird per CAT-Eduroam verteilt, spart einen Roundtrip  
    default_eap_type = peap  
  
    ttls {  
        tls = tls-common  
        default_eap_type = mschapv2  
        virtual_server = "eduroam-inner"  
    }  
    peap {  
        tls = tls-common  
        default_eap_type = mschapv2  
        virtual_server = "eduroam-inner"  
    }  
    mschapv2 { }  
    ...  
}
```

```
ldap ldap_eduroam {
    # Hier nur begrenzt Unlang möglich
    update {
        # Für MSCHAPv2 und PAP
        control:NT-Password := 'ntPassword'
        # Zuordnung Mitarbeiter vs. Student
        control:Aruba-User-Group := \
            'radiusTunnelPrivateGroupID'
    }
    ...
}
```

```
attr_filter eduroam_attr_filter.pre-proxy {
    key = "%{Realm}"
    filename = ${modconfdir}/${.:name}/eduroam-pre-proxy
}
attr_filter eduroam_attr_filter.post-proxy {
    key = "%{Realm}"
    filename = ${modconfdir}/${.:name}/eduroam-post-proxy
}
```

```
# Keine unnötigen Attribute an Homeserver schicken
DEFAULT
    User-Name =* ANY,
    EAP-Message =* ANY,
    Message-Authenticator =* ANY,
    State =* ANY,
    Operator-Name =* ANY,
    Calling-Station-Id =* ANY,
    Proxy-State =* ANY
```

```
# Keine unnötigen Attribute von Homeservern akzeptieren
DEFAULT
    Reply-Message =* ANY,
    Proxy-State =* ANY,
    EAP-Message =* ANY,
    Message-Authenticator =* ANY,
    # Schlüssel für den WLAN-Controller
    MS-MPPE-Recv-Key =* ANY,
    MS-MPPE-Send-Key =* ANY,
    State =* ANY
```

eduroam Proxy

```
home_server radsecproxy {
    ...
    # Falls Homeserver down, nicht radsecproxy als down markieren
    status_check = status-server
}
home_server_pool radsecproxy-pool {
    type = fail-over
    home_server = radsecproxy
}

realm fau.de { }      # Leere Realms werden lokal behandelt
realm LOCAL { }      # Um Proxying zu verhindern
realm NULL { }       # Ohne Realm
realm DEFAULT {
    auth_pool = radsecproxy-pool
    nostrip        # Realm nicht entfernen
}
```


eduroam Clients

```
client radsecproxy {
    ipaddr          = 127.0.0.1
    virtual_server = eduroam
    ...
}

client wlan-controller {
    ipaddr          = 192.0.2.23
    virtual_server = eduroam
    ...
}

...
```

Debugging

Debugging

- **Nur** auf Testserver: `radiusd -X`
(RDEBUG3-Nachrichten mit `-Xx`)
- Im Betrieb: `raddebug`
 - Nutzer filtern: `-u <name>`
 - Bedingungen mit Unlang: `-c <bedingung>`
bspw. `-c "(&Client-Shortname == 'radsecproxy')"`
 - Debugzeit ändern: `-t <timeout>`

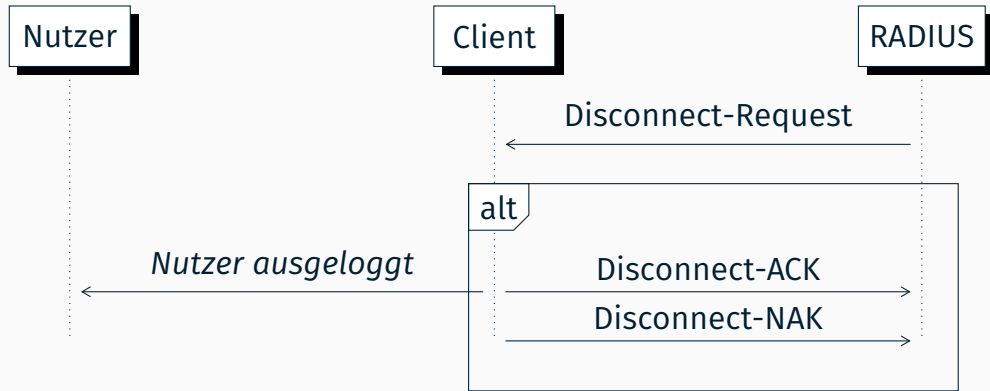
Debugging

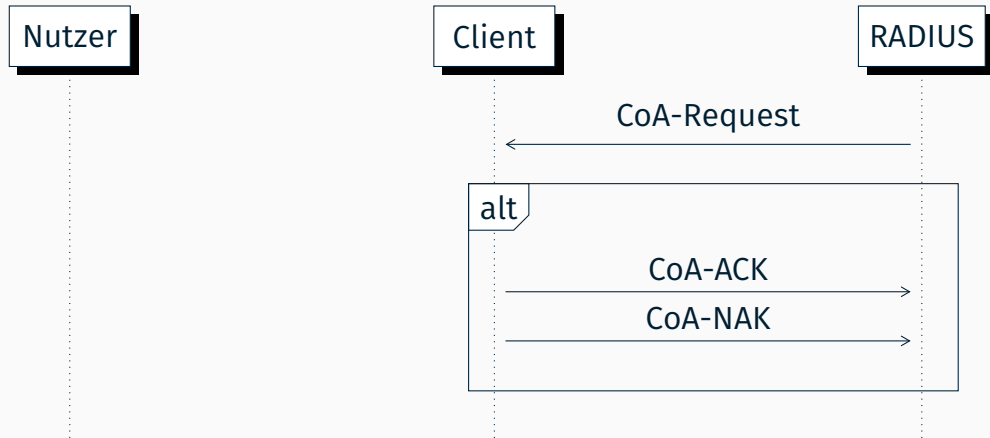
- **Nur** auf Testserver: `radiusd -X`
(RDEBUG3-Nachrichten mit `-Xx`)
- Im Betrieb: `raddebug`
 - Nutzer filtern: `-u <name>`
 - Bedingungen mit Unlang: `-c <bedingung>`
bspw. `-c "(&Client-Shortname == 'radsecproxy')"`
 - Debugzeit ändern: `-t <timeout>`
 - Braucht Control-Socket, Konfiguration für `radiusd.conf`:

```
listen {  
    type = control  
    socket = ${run_dir}/${name}.sock  
    mode = rw  
}
```

- <https://www.freeradius.org/documentation/freeradius-server/>
- <https://networkradius.com/doc/FreeRADIUS-Technical-Guide.pdf>
- <https://download.rrze.fau.de/noc/eduroam.conf.txt>
(Beispielkonfiguration als Textdatei)

Appendix





Beispiel: Nutzer ablehnen

```
authorize {  
  if (&User-Name && &User-Name == "simon") {  
    update request {  
      &Module-Failure-Message += \  
        "Du (%{User-Name}) nicht!"  
    }  
    reject  
  }  
}
```

Beispiel: Anfrage ohne Check erlauben

```
authorize {  
  if (...) {  
    update {  
      &control:Auth-Type := Accept  
    }  
    return  
  }  
}
```

Beispiel: users-Datei (Modul files)

```
bob Huntgroup-Name == EXAMPLE, Cleartext-Password := "pass"  
    Reply-Message = "Hey ;-)",  
    Framed-IP-Address = 192.0.2.0
```

- Erste Zeile enthält Vergleiche (==) und Zuweisungen (= oder :=)
- **Falls** Vergleiche zutreffen, wird Block ausgewertet
- Dann werden Zuweisungen der ersten Zeile an `&control` durchgeführt
- Sowie Zuweisungen an `&reply` in den folgenden Zeilen

Debugging: Fehlermeldungen

- Nutzer nicht in LDAP/Datenbank gefunden:
No Auth-Type found: rejecting the user via
Post-Auth-Type = Reject
- Dito:
No NT-Password. Cannot perform authentication
- Falsches Passwort:
NT digest does not match "known good" digest