freeRADIUS

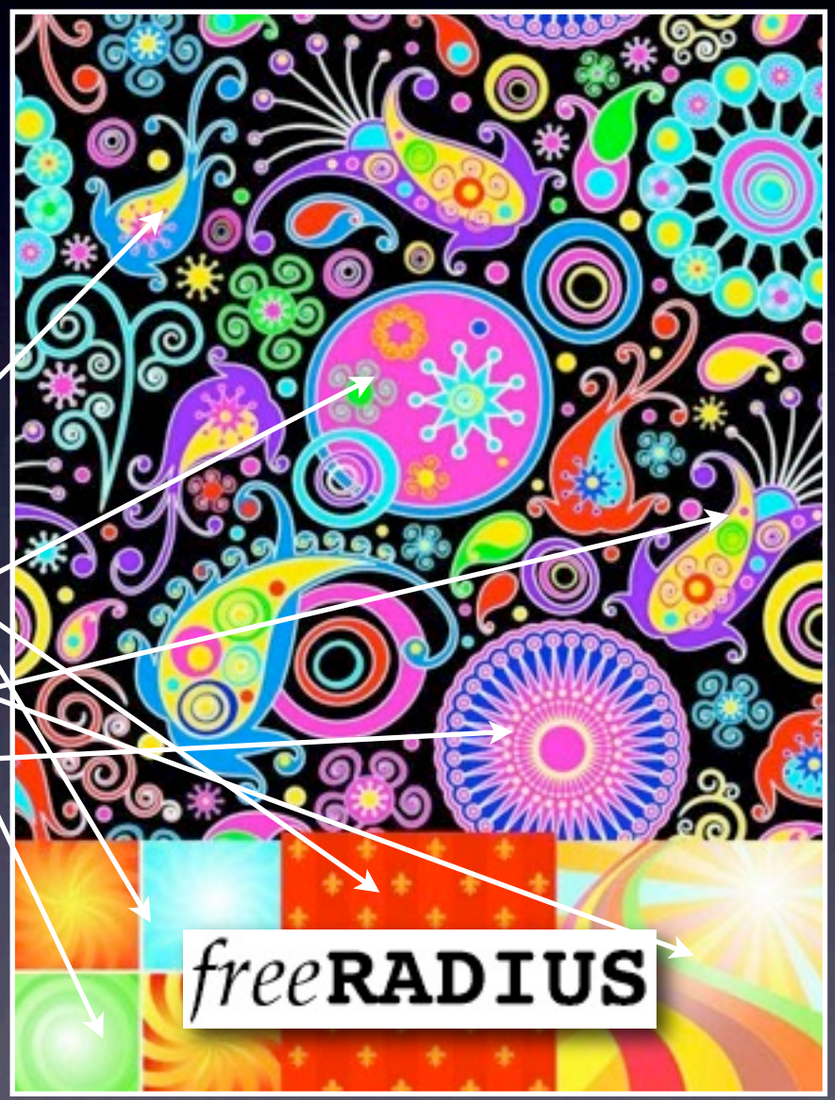
Serveur RADIUS libre, performant et modulaire

mais pas vraiment simple

Aurélien Geron, Wifirst, 7 janvier 2011

Plan

- Plusieurs protocoles : RADIUS, EAP...
- Un serveur sous GPLv2
- Un système de configuration puissant
- Une multitude de modules
- Comment créer un module ?



Source image: http://crshare.com/abstract-backgrounds-vector-clipart/

Modules déjà mentionnés

- **`preprocess`** : nettoyage d'attributs bizarres + `hints` + `huntgroups`
- **`files`** : ajout d'attributs dans les listes `control` et `reply`
- **`pap`** : authentification PAP
- **`chap`** : authentification CHAP
- **`mschap`** : authentification MS-CHAP et MS-CHAP-v2
- **`ldap`** : authentification via un annuaire LDAP
- **`sql`** : authentification via une base de données relationnelle
- **`realm`** : extraction du *realm* à partir du `User-Name` (préfixe/suffixe)
- **`attr_filter`** : filtre les attributs (notamment pour le *roaming*)
- **`exec`** : faire une substitution avec la sortie d'un programme
- **`expr`** : faire une substitution avec le résultat d'un calcul
- **`perl`** : définition d'un module en perl (ex: pour des substitutions)
- **`python`** : définition d'un module en python
- **`always`** : donne toujours le même résultat (`ok`, `fail`...)
- ~~**`policy`**~~ : module à ne plus utiliser, remplacé par `policy.conf`

Module `sql`

- Ce module permet de stocker dans une base de données SQL :
  - la liste des NAS (en complément de `clients.conf`), lue uniquement au démarrage de freeRADIUS
  - la liste des utilisateurs, avec leur mot de passe, les groupes auxquels ils appartiennent, et les attributs à rajouter pour chaque utilisateur et chaque groupe
  - la liste détaillée des sessions (il permet d'ailleurs de limiter le nombre de sessions par utilisateur, si on le souhaite)
- Il permet aussi (nous l'avons vu) de faire des substitutions (`xlat`) reposant sur des requêtes SQL quelconques
- On peut se contenter d'utiliser uniquement les fonctionnalités de son choix (par exemple, juste la liste des NAS et les substitutions)
- Il repose, au choix, sur MySQL, PostgreSQL, Oracle, DB2, SqlLite, Sybase, Firebird, iODBC, UnixODBC ou FreeTDS

Module `sql`

- Pour utiliser le module `sql`, il faut commencer par installer le pilote (*driver*) correspondant à la base de données

  ➡ Par exemple, sous Debian, le package `freeradius-mysql` doit être installé pour utiliser le module `sql` avec une base de données MySQL

- Il faut ensuite éditer le fichier `/etc/freeradius/sql.conf` et renseigner le type de base de données utilisé (ex: `mysql`), le nom du driver (ex: `rlm_sql_mysql`), l'hôte et le port du serveur de bases de données, les identifiants à utiliser pour s'y connecter, le nom des tables, etc.

- Puis créer les tables qui seront nécessaires dans la base de données, en exécutant les requêtes SQL des fichiers `*.sql` dans le répertoire `/etc/freeradius/sql/mysql` (ou `postgresql`, ou `oracle`...)

Module sql

- Voici par exemple la commande qui permet de créer la table **nas** dans une base de données MySQL :

```
CREATE TABLE nas (
  id int(10) NOT NULL auto_increment,
  nasname varchar(128) NOT NULL,
  shortname varchar(32),
  type varchar(30) DEFAULT 'other',
  ports int(5),
  secret varchar(60) DEFAULT 'secret' NOT NULL,
# server varchar(64),
  community varchar(50),
  description varchar(200) DEFAULT 'RADIUS Client',
  PRIMARY KEY (id),
  KEY nasname (nasname)
);
```

/etc/freeradius/sql/mysql/nas.sql

- La colonne `nasname` contient l'adresse ou le sous-réseau IP du NAS

  ➡ Exemples : "`10.1.2.3`" ou "`10.1.2.0/24`"

- La colonne `community` est utilisée par l'outil `checkrad` pour interroger le NAS via SNMP (pour les NAS dont le type le permet)

Module `sql`

Le fichier SQL principal est `schema.sql` ; il contient les commandes qui créent les tables suivantes :

- `radacct` : historique des sessions
- `radpostauth` : historique des authentifications
- `radcheck` : filtres sur les attributs de la liste **`request`** et opérations sur les attributs de la liste **`control`** (comme dans le fichier `users` du module `files`)
- `radreply` : opérations sur les attributs de la liste **`reply`** (idem)
- `radusergroup` : liste des groupes de chaque utilisateur
- `radgroupcheck` : comme `radcheck` mais s'applique à des groupes (plutôt qu'à des utilisateurs)
- `radgroupreply` : comme `radreply` mais s'applique à des groupes

Module sql

```
CREATE TABLE radacct (
  radacctid bigint(21) NOT NULL auto_increment,
  acctsessionid varchar(64) NOT NULL default '',
  acctuniqueid varchar(32) NOT NULL default '',
  username varchar(64) NOT NULL default '',
  groupname varchar(64) NOT NULL default '',
  realm varchar(64) default '',
  nasipaddress varchar(15) NOT NULL default '',
  nasportid varchar(15) default NULL,
  nasporttype varchar(32) default NULL,
  acctstarttime datetime NULL default NULL,
  acctstoptime datetime NULL default NULL,
  acctsessiontime int(12) default NULL,
  acctauthentic varchar(32) default NULL,
  connectinfo_start varchar(50) default NULL,
  connectinfo_stop varchar(50) default NULL,
  acctinputoctets bigint(20) default NULL,
  acctoutputoctets bigint(20) default NULL,
  calledstationid varchar(50) NOT NULL default '',
  callingstationid varchar(50) NOT NULL default '',
  acctterminatecause varchar(32) NOT NULL default '',
  servicetype varchar(32) default NULL,
  framedprotocol varchar(32) default NULL,
  framedipaddress varchar(15) NOT NULL default '',
  acctstartdelay int(12) default NULL,
  acctstopdelay int(12) default NULL,
  xascendsessionsvrkey varchar(10) default NULL,
  PRIMARY KEY  (radacctid),
  KEY username (username),
  KEY framedipaddress (framedipaddress),
  KEY acctsessionid (acctsessionid),
  KEY acctsessiontime (acctsessiontime),
  KEY acctuniqueid (acctuniqueid),
  KEY acctstarttime (acctstarttime),
  KEY acctstoptime (acctstoptime),
  KEY nasipaddress (nasipaddress)
) ;
```

```
CREATE TABLE radpostauth (
  id int(11) NOT NULL auto_increment,
  username varchar(64) NOT NULL default '',
  pass varchar(64) NOT NULL default '',
  reply varchar(32) NOT NULL default '',
  authdate timestamp NOT NULL,
  PRIMARY KEY  (id)
) ;
```

```
CREATE TABLE radcheck (
  id int(11) unsigned NOT NULL auto_increment,
  username varchar(64) NOT NULL default '',
  attribute varchar(64)  NOT NULL default '',
  op char(2) NOT NULL DEFAULT '==',
  value varchar(253) NOT NULL default '',
  PRIMARY KEY  (id),
  KEY username (username(32))
) ;
```

```
CREATE TABLE radreply ( # comme radcheck
  ...
) ;
```

```
CREATE TABLE radusergroup (
  username varchar(64) NOT NULL default '',
  groupname varchar(64) NOT NULL default '',
  priority int(11) NOT NULL default '1',
  KEY username (username(32))
) ;
```

```
CREATE TABLE radgroupcheck ( # comme radcheck
  ... # avec groupname plutôt que username
) ;
```

```
CREATE TABLE radgroupreply ( # comme radreply
  ... # avec groupname plutôt que username
) ;
```

Module `ldap`

Le module LDAP effectue des tâches différentes selon les sections où on l'active :

- dans la section `authorize`, il se connecte (on parle de `bind`) à l'annuaire LDAP, il recherche l'entrée correspondant à l'utilisateur, il lit tous les attributs LDAP de cet utilisateur, et enfin il traduit certains de ces attributs en attributs RADIUS ajoutés dans la liste **`control`** ou **`reply`** (les règles de traduction sont définies dans le fichier `/etc/freeradius/ldap.attrmap`)

- dans la section `authenticate`, il essaie de se connecter à l'annuaire LDAP avec l'identité (`User-Name`) et le mot de passe (`User-Password`) de l'utilisateur (on parle de «*bind-as-user*», bind-en-tant-qu'utilisateur) : l'utilisateur est rejeté si le `bind` échoue

- dans la section `post-auth`, si l'annuaire est un eDirectory de Novell, le module vérifie les politiques de sécurité de l'eDirectory et l'informe en cas d'échec de l'authentification

Module `ldap`

Il y a donc deux façons totalement différentes d'utiliser le module LDAP pour authentifier les utilisateurs :

1. soit on l'utilise pour récupérer l'attribut LDAP correspondant au mot de passe de l'utilisateur (ou à un *hash* du mot de passe) pendant la phase `authorize` puis on utilise un module différent pour vérifier ce mot de passe (ex: `pap` ou `ms-chap`)

   - La méthode d'authentification utilisée devra être compatible avec le format du mot de passe : par exemple, on ne pourra utiliser la méthode PEAP/MS-CHAP-v2 que si l'annuaire LDAP possède le *hash* `NT-Password` de l'utilisateur

2. soit on fait un *bind-as-user* dans la phase `authenticate`

   - Le serveur freeRADIUS devra avoir accès au mot de passe non chiffré envoyé par l'utilisateur, ce qui n'est le cas qu'avec les méthodes d'authentification PAP ou TTLS/PAP

Module `ldap`

```
ldap {
     server = "ldaps://ldap.ma-societe.fr"

     identity = "cn=admin,o=ma-societe,c=FR"
     password = "F5138F$SD~4FKD3!4cFNb"

     basedn = "ou=people,dc=ma-societe,dc=net"
     filter = "(uid=%{%{Stripped-User-Name}:-%{User-Name}})"
     #base_filter = "(objectclass=radiusprofile)"

     ldap_connections_number = 5
     timeout = 4
     timelimit = 3
     net_timeout = 1

     tls {
          start_tls = no
          cacertfile     = /etc/ssl/certs/ca-du-serveur.pem
          #cacertdir     = /etc/ssl/certs/
          #certfile      = /etc/ssl/certs/client-ldap.pem
          #keyfile       = /etc/ssl/private/client-ldap.key
          randfile       = /dev/urandom
          require_cert   = "demand" # vérifier le certificat !
     }

     # default_profile = "cn=radprofile,ou=dialup,o=ma-societe,c=FR"

     # ...
}
```

Adresse de l'annuaire LDAP

Identifiants de connexion (*bind*) utilisés pour la phase `authorize` (sinon, c'est un *bind* anonyme)

Paramètres pour la recherche de l'entrée LDAP de l'utilisateur

Paramètres de connexion

Paramètres TLS pour les connexions LDAP**S** (sécurisées)
Note: il vaut mieux toujours vérifier que le certificat du serveur est valide et correspond bien au nom du serveur

Entrée LDAP par défaut contenant des attributs à appliquer à tous les utilisateurs

suite à la page suivante

/etc/freeradius/modules/ldap

Module ldap

```
ldap {
    # ...   suite de la page précédente

    dictionary_mapping = ${confdir}/ldap.attrmap

    #password_attribute = userPassword

    # groupname_attribute = cn
    # groupmembership_filter = "..."
    # groupmembership_attribute = radiusGroupName

    # compare_check_items = yes

    # do_xlat = yes


    # access_attr_used_for_allow = yes
    # access_attr = "dialupAccess"

    # chase_referrals = yes
    # rebind = yes

    edir_account_policy_check = no

    set_auth_type = no


    #ldap_debug = 0x0028
}
```

Correspondance entre attributs LDAP et RADIUS (voir ci-après)

Attribut LDAP contenant le mot de passe de l'utilisateur (un peu inutile depuis qu'on peut définir `ldap.attrmap`)

Pour chercher l'entrée LDAP du groupe de l'utilisateur, et rajouter les attributs correspondants

Comparer la valeur des attributs de la requête avec les attributs correspondant dans l'entrée LDAP de l'utilisateur

Appliquer les éventuelles substitutions contenues dans les valeurs des attributs récupérés depuis l'annuaire LDAP

Si **`access_attr_user_for_allow`** vaut `yes`, alors la présence de l'attribut `dialupAccess` dans l'entrée LDAP de l'utilisateur indique qu'il doit être autorisé, sinon il doit être rejeté. S'il vaut `no`, alors c'est l'inverse.

/etc/freeradius/modules/ldap

Module ldap

```
ldap {
      # ...
      dictionary_mapping = ${confdir}/ldap.attrmap

      #password_attribute = userPassword

      # groupname_attribute = cn
      # groupmembership_filter = "..."
      # groupmembership_attribute = radiusGroupName

      # compare_check_items = yes

      # do_xlat = yes


      # access_attr_used_for_allow = yes
      # access_attr = "dialupAccess"

      # chase_referrals = yes
      # rebind = yes

      edir_account_policy_check = no

      set_auth_type = no


      #ldap_debug = 0x0028
}
```

suite de la page précédente

Paramètres de connexion pour les annuaires Active Directory de Microsoft

Appliquer les politiques de sécurité définies dans l'annuaire s'il est de type eDirectory de Novell ?

Est-ce que l'attribut **`Auth-Type`** doit être fixé à **`ldap`** pendant la phase `authorize` (pour forcer le mode *bind-as-user*) ?

Clé de déboggage pour OpenLDAP (`0x0000` = pas de messages)

/etc/freeradius/modules/ldap

Module ldap

```
checkItem $GENERIC$                 radiusCheckItem
replyItem $GENERIC$                 radiusReplyItem

checkItem Auth-Type                 radiusAuthType
checkItem Simultaneous-Use          radiusSimultaneousUse
checkItem LM-Password               lmPassword
checkItem NT-Password               ntPassword
checkItem LM-Password               sambaLmPassword
checkItem NT-Password               sambaNtPassword
checkItem LM-Password               dBCSPwd
checkItem SMB-Account-CTRL-TEXT     acctFlags
checkItem Expiration                radiusExpiration

checkItem Called-Station-Id         radiusCalledStationId
checkItem Calling-Station-Id        radiusCallingStationId
checkItem NAS-IP-Address            radiusNASIpAddress

replyItem Service-Type              radiusServiceType
replyItem Framed-Protocol           radiusFramedProtocol
replyItem Framed-IP-Address         radiusFramedIPAddress
replyItem Framed-IP-Netmask         radiusFramedIPNetmask
replyItem Framed-Route              radiusFramedRoute
replyItem Framed-Routing            radiusFramedRouting
replyItem Filter-Id                 radiusFilterId
replyItem Framed-MTU                radiusFramedMTU
replyItem Framed-Compression        radiusFramedCompression
replyItem Login-IP-Host             radiusLoginIPHost
...
replyItem Port-Limit                radiusPortLimit
replyItem Login-LAT-Port            radiusLoginLATPort
replyItem Reply-Message             radiusReplyMessage
replyItem Tunnel-Type               radiusTunnelType
replyItem Tunnel-Medium-Type        radiusTunnelMediumType
replyItem Tunnel-Private-Group-Id   radiusTunnelPrivateGroupId
```

/etc/freeradius/ldap.attrmap

Ces attributs peuvent contenir des opérations sur tout attribut, par exemple :
« `Filter-ID += "juste-web"` »

Ces attributs sont automatiquement rajoutés à la liste **`control`**

Ces attributs sont comparés à ceux de la liste **`request`** si `compare_check_items = yes` et l'utilisateur est rejeté s'ils diffèrent

Note: les précédents aussi, mais comme ce sont des attributs internes à freeRADIUS, on ne les trouvera normalement jamais dans la requête

Ces attributs sont automatiquement rajoutés à la liste **`reply`**

Exemple concret

On souhaite :

- mettre en place un réseau WiFi sécurisé par WPA dans de nombreux sites, avec une authentification centralisée
- utiliser la méthode d'authentification PEAP/MS-CHAP-v2
- consulter un annuaire LDAP qui contient les comptes des utilisateurs, pour la vérification des mots de passe
- lire la liste des NAS dans une base de données MySQL

Exemple concret

- Il faut activer le WPA Enterprise dans tous les points d'accès WiFi (ils doivent donc être compatibles avec le WPA Enterprise) : choisir de préférence le WPA2 avec chiffrement AES
- Il faut configurer chaque AP en indiquant l'adresse IP et le port de notre serveur freeRADIUS, et désactiver l'*accounting* (pas besoin)
- Installer le serveur MySQL et créer la base de données (à moins qu'ils existent déjà)
- Créer la table `nas` en appliquant le fichier `nas.sql`, puis remplir cette table avec les paramètres de chaque NAS
- S'assurer que l'annuaire LDAP contient un attribut `ntPassword` (ou `sambaNtPassword`) rattaché à chaque utilisateur
- Installer les packages `freeradius` et `freeradius-mysql` (sous Debian) sur le serveur

Exemple concret

- L'authentification PEAP/MS-CHAP-v2 fonctionnera également si l'annuaire LDAP contient le mot de passe en clair, mais c'est fortement déconseillé, pour des questions de sécurité
- Si l'annuaire ne contient ni le mot de passe en clair, ni le *hash* `NT-Password`, alors il faudra :
  - ajouter le schéma Samba à l'annuaire LDAP pour autoriser les entrées `ntPassword` (ou `sambaNtPassword`)
  - puis demander à chaque utilisateur de saisir à nouveau son mot de passe pour que le *hash* `NT-Password` soit bien créé

    ➡ Autre option : utiliser la méthode TTLS/PAP plutôt que le PEAP/MS-CHAP-v2, mais cela suppose d'installer un logiciel *supplicant* TTLS sur les postes des utilisateurs sous Windows
- S'assurer que le fichier `ldap.attrmap` contient bien les lignes :

```
checkItem NT-Password ntPassword
checkItem NT-Password sambaNtPassword
```

Exemple concret

Ensuite il faut installer et configurer freeRADIUS :

- `eap.conf` : activer et configurer le `tls` et le `peap` pour établir le tunnel `peap`, et activer `mschap` pour activer le MS-CHAP-v2 à l'intérieur du tunnel
- `sql.conf` : indiquer l'adresse et les identifiants de connexion au serveur SQL
- `modules/ldap` : configurer les paramètres d'accès à l'annuaire, et les paramètres de recherche des entrées des utilisateurs
- `sites-available/default` : activer `preprocess` et `eap` dans la phase `authorize`, et `eap` uniquement dans la phase `authenticate`
- `sites-available/inner-tunnel` : activer `preprocess`, `eap` et `ldap` dans la phase `authorize`, et `mschap` uniquement dans la phase `authenticate`
- s'assurer qu'il y a bien un lien symbolique vers ces deux fichiers dans le répertoire `sites-enabled`

Exemple concret

```
eap {
     default_eap_type = peap
     timer_expire     = 60
     ignore_unknown_eap_types = no
     cisco_accounting_username_bug = no
     max_sessions = 4096
     tls {
          certdir = ${confdir}/certs
          cadir = ${confdir}/certs
          private_key_file = ${certdir}/server.key
          certificate_file = ${certdir}/server.pem
          dh_file = ${certdir}/dh
          random_file = ${certdir}/random
          cipher_list = "DEFAULT"
          cache {
               enable = no
               lifetime = 24
               max_entries = 255
          }
     }
     peap {
          default_eap_type = mschapv2
          copy_request_to_tunnel = yes
          use_tunneled_reply = no
          virtual_server = "inner-tunnel"
     }
     mschapv2 {
     }
}
```

/etc/freeradius/eap.conf

Exemple concret

- Pour créer un certificat TLS autosigné, installer OpenSSL et exécuter les trois commandes suivantes :

```
#openssl genrsa -out ma-societe.key 1024
Generating RSA private key, 1024 bit long modulus
..++++++
..++++++
e is 65537 (0x10001)

#openssl req -new -key ma-societe.key -out ma-societe.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Ma société
Organizational Unit Name (eg, section) []:RD
Common Name (eg, YOUR name) []:Ma societe, WiFi WPA
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

#openssl x509 -req -days 3650 -in ma-societe.csr -signkey ma-societe.key -out ma-societe.crt
Signature ok
subject=/C=FR/ST=France/L=Paris/O=Ma soci\xC3\xA9t\xC3\xA9/OU=RD/CN=Ma societe, WiFi WPA
Getting Private key
```

Exemple concret

```
sql {
      database = "mysql"
      driver = "rlm_sql_${database}"
      server = "mysql.ma-societe.fr"
      login = "radius"
      password = "NAB3kFp$FDk.11F9kC"
      radius_db = "radius"
      readclients = yes
      nas_table = "nas"

      num_sql_socks = 5
      connect_failure_retry_delay = 60
      lifetime = 0
      max_queries = 0
      $INCLUDE sql/${database}/dialup.conf

      sqltrace = no
      sqltracefile = ${logdir}/sqltrace.sql

      acct_table1 = "radacct"
      acct_table2 = "radacct"
      postauth_table = "radpostauth"
      authcheck_table = "radcheck"
      authreply_table = "radreply"
      groupcheck_table = "radgroupcheck"
      groupreply_table = "radgroupreply"
      usergroup_table = "radusergroup"
      deletestalesessions = yes
}
```

Inutilisé dans notre exemple

/etc/freeradius/sql.conf

Exemple concret

Attention : dans notre exemple, nous n'utilisons le module `sql` nul part dans la configuration des politiques de gestion des paquets, du coup le module ne sera pas instancié automatiquement

La solution est simple : il suffit de rajouter le module `sql` dans la section `instantiate` dans le fichier `radiusd.conf`,
ainsi la liste des NAS sera bien lue dans la base de données MySQL au démarrage de freeRADIUS

Exemple concret

```
ldap {
     server = "ldap.private.ma-societe.fr"
     identity = "cn=admin,dc=ma-societe,dc=fr"
     password = "df0fkNS$2.2F,SC1"
     basedn = "ou=people,dc=ma-societe,dc=fr"
     filter = "(uid=%{%{Stripped-User-Name}:-%{User-Name}})"
     ldap_connections_number = 5
     timeout = 4
     timelimit = 3
     net_timeout = 1
     dictionary_mapping = ${confdir}/ldap.attrmap
     edir_account_policy_check = no
     set_auth_type = no
}
```

/etc/freeradius/modules/ldap

```
checkItem NT-Password ntPassword
checkItem NT-Password sambaNtPassword
```

/etc/freeradius/ldap.attrmap

Exemple concret

1. Négociation EAP et établissement du tunnel PEAP

4. Réponse finale

```
authorize {
        preprocess
        eap
}
authenticate {
        eap
}
```

Le tunnel `peap` est établi ici (cela suppose plusieurs échanges EAP dans des paquets RADIUS distincts)

Autorise l'utilisateur si la méthode interne à PEAP réussit

`/etc/freeradius/sites-available/default`

`/etc/freeradius/sites-enabled/default`

2. EAP

3. MS-CHAP-v2

```
server inner-tunnel {
  authorize {
    preprocess
    mschap
    eap {
      ok = return
    }
    ldap
  }
  authenticate {
    Auth-Type MS-CHAP {
      mschap
    }
  }
}
```

Permet d'éviter l'interrogation de l'annuaire LDAP lors de la négociation EAP (choix de la méthode d'authentification)

Récupération du `NT-Password` de l'utilisateur dans l'annuaire LDAP

Authentification MS-CHAP-v2 en utilisant le `NT-Password`

`/etc/freeradius/sites-available/inner-tunnel`

`/etc/freeradius/sites-enabled/inner-tunnel`

Autres modules

```
#ls /etc/freeradius/modules
acct_unique                 logintime
always                      mac2ip
attr_filter                 mac2vlan
attr_rewrite                mschap
chap                        ntlm_auth
checkval                    otp
counter                     pam
cui                         pap
detail                      passwd
detail.example.com          perl
detail.log                  policy
digest                      preprocess
echo                        python
etc_group                   radutmp
exec                        realm
expiration                  smbpasswd
expr                        smsotp
files                       sqlcounter_expire_on_login
inner-eap                   sql_log
ippool                      sradutmp
krb5                        unix
ldap                        wimax
linelog
```

Suivi des sessions

- **radutmp** : si ce module est placé dans la section `session`, alors il enregistre dans un fichier la liste des sessions ouvertes actuellement, avec le `User-Name` et le lieu de connexion associé (le programme `radwho` affiche cette liste). Si l'attribut `Simultaneous-Use` a été rajouté (par un autre module) dans la liste **control** et si l'utilisateur a déjà un nombre de sessions ouvertes égal à la valeur de l'attribut `Simultaneous-Use`, alors ce module rejette la requête.
  - Note : le module `sql` offre la même fonctionnalité mais il stocke bien sûr les sessions ouvertes dans une base de données... et il est bien plus rapide.
- **sradutmp** : variante «**s**écurisée» du module `radutmp` qui n'enregistre pas l'identifiant de la machine du client (généralement son adresse MAC), car cette information est parfois considérée comme «sensible» dans certains contextes.

Historisation détaillée

- **`detail`** : enregistre toutes les requêtes d'accounting dans un fichier.

  ➡ Si on le souhaite, ces requêtes peuvent en parallèle être lues régulièrement par freeRADIUS (il faut une section `listen` de type `detail` dans `radiusd.conf` pour cela) et renvoyées vers un autre serveur (c'était le rôle du démon `radrelay` dans la version 1 de freeRADIUS). Ceci permet de synchroniser les bases d'accounting de plusieurs serveurs freeRADIUS, par exemple un serveur primaire et un serveur secondaire.

- **`detail.log`** : variante du module `detail` qui enregistre un historique détaillé dans un fichier. Il peut s'agir, au choix, du détail des requêtes d'authentification, des réponses, ou encore des requêtes avant ou après leur redirection vers un *Home-Server*. On peut configurer le module pour supprimer tel ou tel attribut de l'historique, par sécurité (par exemple le mot de passe de l'utilisateur).

Historisation détaillée

- `linelog` : permet d'écrire une ligne de *log* pour chaque requête.
- `sql_log` : écrit dans un fichier une requête SQL à chaque requête.

➡ On peut installer un démon `radsqlrelay` (disponible dans le package `freeradius-utils` sous Debian), qui se charge de lire le fichier généré par le module `sql_log` et d'exécuter les requêtes au fur et à mesure qu'elles arrivent

Compteurs

- **`counter`** : ce module conserve dans un fichier le décompte total du temps de connexion pour chaque `User-Name` pour la journée en cours. Si l'attribut `Max-Daily-Session` est rajouté (par un autre module) dans la liste `control`, alors il indique la durée maximale cumulée des sessions de l'utilisateur pour la journée. Si cette durée maximale est déjà atteindre, ce module rejette l'utilisateur. Sinon, il rajoute l'attribut `Session-Timeout` dans la liste `reply`, indiquant le temps maximal restant (le NAS se charge alors de couper la session au bout du temps indiqué).

➡ Tout est paramétrable : on peut limiter le temps de connexion par heure, par jour, par semaine ou par mois. On peut aussi limiter le volume téléchargé ou encore le nombre de sessions.

➡ Si on utilise ce module, il faut aussi le rajouter dans la section `instantiate` avant tous les autres modules, car il définit l'attribut `Max-Daily-Session` (ou autre, le cas échéant).

Compteurs

- **`sqlcounter`** : ce module repose sur le module `sql` pour accéder à une base de données (le module `sql` doit donc être configuré) et il exécute une requête SQL qui calcule la durée cumulée de toutes les sessions de l'utilisateur pour la journée en cours. Cette durée totale est comparée à la valeur de l'attribut `Max-Daily-Session` (qui doit avoir été rajouté au préalable par un autre module dans la liste `control`). Si cette valeur est dépassée, l'utilisateur est rejeté.

➡ Ici aussi, tout est paramétrable : on peut limiter le temps de connexion par heure, par jour, par semaine ou par mois, ou encore mesurer tout autre paramètre en changeant la requête SQL.

➡ Bien sûr, pour pouvoir faire une requête pertinente dans la base de données, il faut que le module `sql` soit utilisé pour l'accounting.

Compteurs

- **`expire_on_login`** : ce module est une variante du module `sqlcounter` qui lit l'attribut `Expire-After` (pourvu que celui-ci ait été rajouté au préalable dans la liste `control` par un autre module), et s'assure que le nombre de secondes écoulées depuis la 1ère connexion de l'utilisateur ne dépasse pas la valeur de l'attribut.
- **`expiration`** : permet de définir une date d'expiration pour chaque utilisateur. Passée cette date, l'utilisateur ne pourra plus se connecter.
- **`logintime`** : permet de définir des créneaux horaires pendant lesquels l'utilisateur peut se connecter ou non.

Modules d'authentification

- **`krb5`**(1) : authentification Kerberos
- **`unix`**(1) : authentification Unix
- **`pam`**(1) : authentification PAM (Pluggable Authentication Module)
- **`ntlm_auth`**(1) : authentification NT/LM
- **`otp`**(2) : authentification OTP (*One-Time Password*)
- **`smbpasswd`**(2) : variante du module `passwd` pour l'authentification sur un fichier de mots de passes au format de Samba :
  - `format = "*User-Name::LM-Password:NT-Password:SMB-Account-CTRL-TEXT::"`
- **`smsotp`**(1) : authentification OPT par SMS (un mot de passe unique est envoyé par SMS)

(1) Uniquement avec les méthodes PAP et TTLS/PAP
(2) Uniquement avec les méthodes PAP, TTLS/PAP et PEAP/MS-CHAP-v2

Analyse de fichiers

- **`passwd`** : module qui analyse (*parse*) un fichier selon un format configurable (voir page suivante), et en extrait des attributs.
- **`etc_group`** : variante du module `passwd` qui trouve les groupes auxquels appartient l'utilisateur à partir d'un fichier du type de `/etc/group` et rajoute pour chaque groupe un attribut `Etc-Group-Name` dans la liste `control`

Module passwd

- Les lignes d'un fichier analysé par le module **passwd** doivent être composées de champs séparés par un caractère `delimiter`
- On indique ensuite à quel attribut correspond chaque champ grâce au paramètre `format` (les champs du `format` sont délimités par le caractère «:», même si le `delimiter` est différent)

```
passwd etc_group {
    filename = /etc/group
    format = "=Etc-Group-Name:::*,User-Name"
    delimiter = ":"
    ...
}
```

- Chaque nom d'attribut peut être précédé d'un ou plusieurs symboles qui indiquent comment traiter l'attribut :
  - `*` Le module analyse le fichier et ne prend en compte que les lignes pour lesquelles ce champ correspond à la valeur de l'attribut dans la requête
  - `,` Ce champ peut être composé d'une suite de valeurs pour l'attribut
  - `~` Cet attribut est rajouté à la liste **request**
  - `=` Cet attribut est rajouté à la liste **reply**

Sans `=`, `~` ou `*` l'attribut est rajouté à la liste `control`

Adressage IP et VLAN

- **`mac2ip`** : variante du module `passwd` qui recherche dans un fichier l'adresse IP que l'on doit attribuer à l'utilisateur, en fonction de son adresse MAC (l'adresse IP est renvoyée dans un attribut au NAS).
- **`mac2vlan`** : idem mais recherche le VLAN que l'on doit attribuer à l'utilisateur en fonction de son adresse MAC (le nom du VLAN est renvoyé dans un attribut au NAS).
- **`ippool`** : permet de définir un pool d'adresses IP, et de distribuer une adresse IP non utilisée à un utilisateur lors de sa connexion.

CUI

- **`cui`** : ce module est une variante du module `sql` qui permet de générer l'attribut **`C`**`hargeable-`**`U`**`ser-`**`I`**`dentity` et de le conserver dans une base de données.

  ➡ Rappel : avec les méthodes d'authentification PEAP et TTLS, l'utilisateur peut cacher son identité à l'intérieur du tunnel TLS, en ne laissant voir qu'une identité telle que «`anonyme`» dans la requête externe...

  ➡ ...malheureusement, c'est cette identité externe qui est envoyée par le NAS pour l'accounting.

  ➡ L'attribut `Chargeable-User-Identity` (CUI) a été défini (dans un draft IETF, bientôt RFC) pour permettre au serveur de fournir au NAS (ou à un serveur proxy) un alias pour l'utilisateur.

  ➡ Si le NAS respecte cet attribut, il doit le renvoyer avec les requêtes d'accounting, ce qui permet au serveur de savoir de quel utilisateur il s'agit.

Divers

- **`echo`** : ce module donne un exemple de variante du module `exec` qui lance un processus externe (ici `/usr/bin/echo`). Les attributs d'une liste (ex. `request`) sont transmis au programme via des variables d'environnement, puis la sortie standard est lue et les attributs qui y apparaissent sont rajoutés à une liste (ex. `reply`).

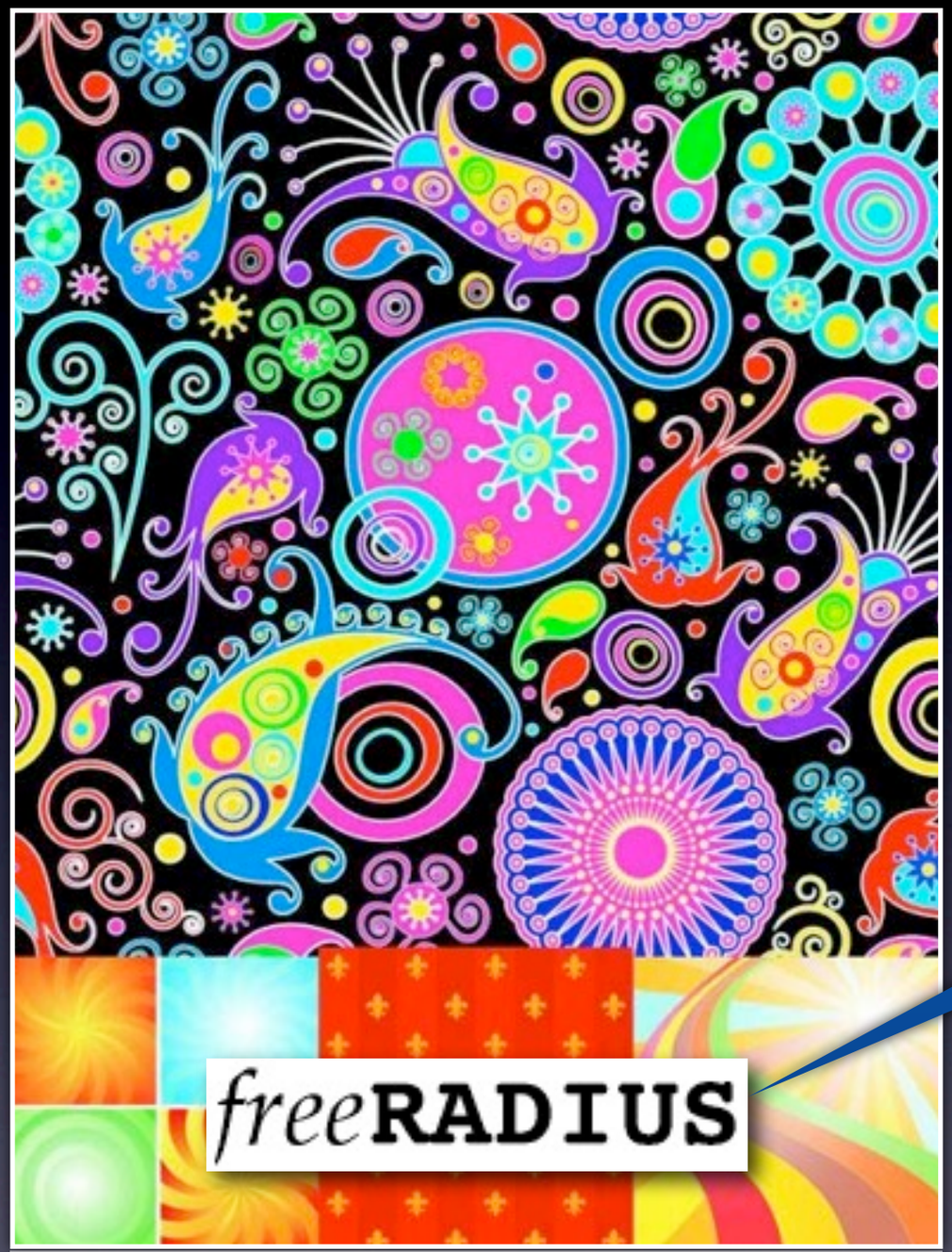
- **`inner-eap`** : une instance du module `eap` que l'on peut utiliser si l'on souhaite pouvoir configurer de façon distincte l'EAP externe et l'EAP interne aux tunnels PEAP et TTLS

- **`acct_unique`** : permet de générer un identifiant de session probablement unique pour remplacer celui qui est envoyé par le NAS. Ceci est parfois utile car certains NAS font l'erreur de répéter les mêmes identifiants de session de temps en temps.

- **`wimax`** : module pour gérer l'échange des clés de chiffrement Wimax

Divers

- ~~`attr_rewrite`~~ : permet de modifier un attribut. On peut désormais faire ça plus simplement avec *unlang*.
- ~~`checkval`~~ : permet de vérifier la valeur d'un attribut. Là-aussi, *unlang* fait mieux.



Questions ?