freeRADIUS

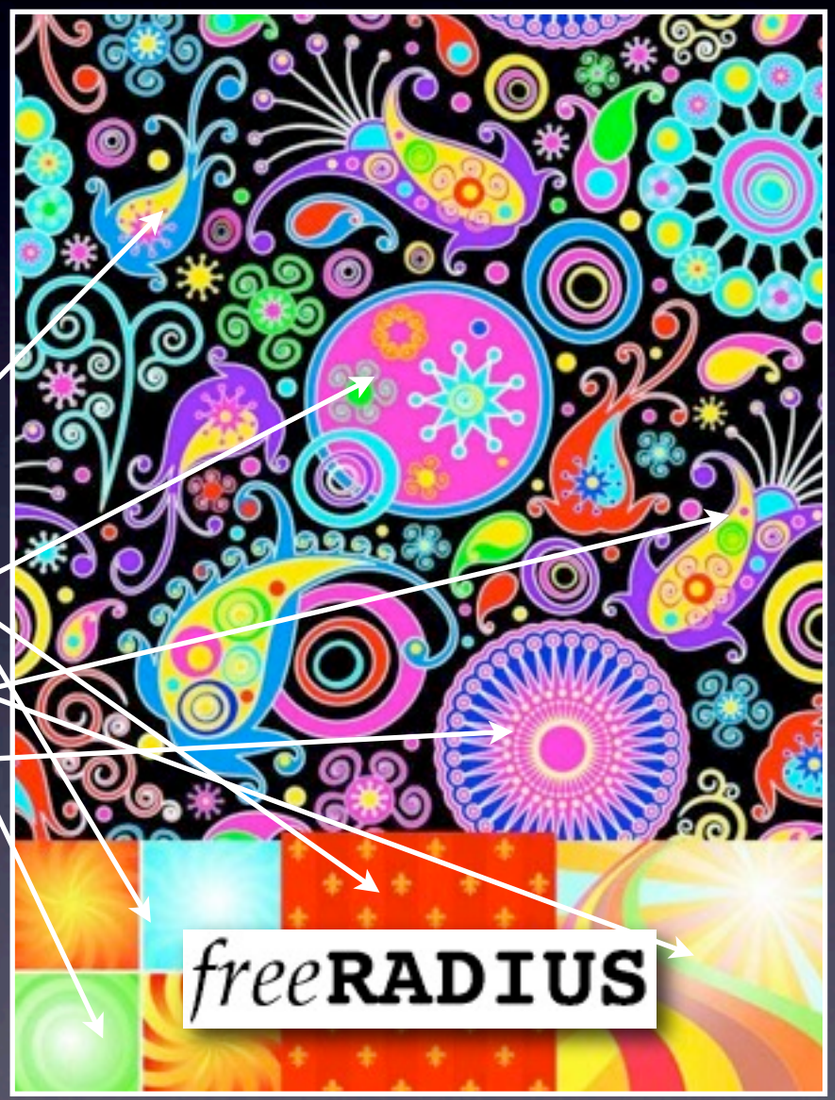
Serveur RADIUS libre, performant et modulaire

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 écrire un module ?



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

Organisation

- La configuration se trouve dans les fichiers situés dans `/etc/freeradius` et ses sous-répertoires
- Pour cette présentation, on distinguera plusieurs parties dans la configuration :
  - La configuration du dictionnaire RADIUS
  - La configuration de base du serveur
  - La configuration des politiques de gestion des requêtes
  - La configuration des modules
  - La configuration du roaming

Organisation

- La configuration se trouve dans les fichiers situés dans `/etc/freeradius` et ses sous-répertoires
- Pour cette présentation, on distinguera plusieurs parties dans la configuration :
  - La configuration du dictionnaire RADIUS
  - La configuration de base du serveur
  - La configuration des politiques de gestion des requêtes
  - La configuration des modules
  - La configuration du roaming

Dictionnaire RADIUS

- Rappel : le nom et le type des attributs ne transitent *pas* dans les paquets RADIUS, seulement leur numéro et leur valeur
- Il serait pénible de configurer le serveur freeRADIUS en faisant référence aux attributs au travers de leur numéro
- C'est pour cela que freeRADIUS utilise un dictionnaire qui permet d'associer un nom et un type à chaque attribut

Dictionnaire RADIUS

- Le fichier `/etc/freeradius/dictionary` est le point d'entrée de la définition du dictionnaire RADIUS utilisé dans la configuration du serveur freeRADIUS
- Il ne contient par défaut qu'une seule ligne (hormis les commentaires), pour inclure le dictionnaire standard :

```
$INCLUDE /usr/share/freeradius/dictionary
```

- Ce fichier inclut lui-même de nombreux autres dictionnaires :

```
$INCLUDE dictionary.rfc2865
$INCLUDE dictionary.rfc2866
$INCLUDE dictionary.rfc2867
...
$INCLUDE dictionary.cisco.bbsm
$INCLUDE dictionary.clavister
...
```

- Pour rajouter des définitions d'attributs personnalisés, il faut modifier `/etc/freeradius/dictionary`, et *jamais* les dictionnaires `/usr/share/freeradius/dictionary.*`

Dictionnaire RADIUS

- Voici pour exemple le début du dictionnaire qui contient les attributs de la RFC 2865 :

```
# -*- text -*-
#
#   Attributes and values defined in RFC 2865.
#   http://www.ietf.org/rfc/rfc2865.txt
#
ATTRIBUTE   User-Name                   1   string
ATTRIBUTE   User-Password                2   string    encrypt=1
ATTRIBUTE   CHAP-Password                3   octets
ATTRIBUTE   NAS-IP-Address               4   ipaddr
ATTRIBUTE   NAS-Port                     5   integer
ATTRIBUTE   Service-Type                 6   integer
ATTRIBUTE   Framed-Protocol              7   integer
ATTRIBUTE   Framed-IP-Address            8   ipaddr
ATTRIBUTE   Framed-IP-Netmask           9   ipaddr
ATTRIBUTE   Framed-Routing               10  integer
ATTRIBUTE   Filter-Id                    11  string
ATTRIBUTE   Framed-MTU                   12  integer
ATTRIBUTE   Framed-Compression           13  integer
ATTRIBUTE   Login-IP-Host                14  ipaddr
ATTRIBUTE   Login-Service                15  integer
ATTRIBUTE   Login-TCP-Port               16  integer
# Attribute 17 is undefined
ATTRIBUTE   Reply-Message                18  string
ATTRIBUTE   Callback-Number              19  string
...
```

Type de chiffrement

Dictionnaire RADIUS

- Pour certains attributs, les valeurs possibles sont elles-mêmes numérotées, et ce qui transite dans un paquet RADIUS est ce numéro (et non une chaîne de caractères).
- Là encore, pour éviter de faire référence aux valeurs possibles avec leur numéro dans la configuration, le dictionnaire permet d'établir la correspondance numéro ↔ nom
- Par exemple, `dictionary.rfc2865` prévoit les valeurs suivantes pour l'attribut `Framed-Compression` (attribut numéro 13) :

```
...
#    Framed Compression Types
VALUE    Framed-Compression    None                    0
VALUE    Framed-Compression    Van-Jacobson-TCP-IP     1
VALUE    Framed-Compression    IPX-Header-Compression  2
VALUE    Framed-Compression    Stac-LZS                3
...
```

Dictionnaire RADIUS

- Enfin, dans le cas des attributs propriétaires (c'est à dire `Vendor-Specific`), le nom de la société n'est *pas* transmis, mais plutôt son numéro délivré par l'organisme IANA
- Le dictionnaire permet là aussi d'établir la correspondance entre le numéro et le nom de la société
- Voici par exemple à quoi ressemble le dictionnaire de Cisco (`dictionary.cisco`), dont le numéro IANA est le **9** :

```
VENDOR  Cisco  9

BEGIN-VENDOR  Cisco

ATTRIBUTE   Cisco-AVPair     1   string
ATTRIBUTE   Cisco-NAS-Port   2   string
...
VALUE   Cisco-Disconnect-Cause      Session-End-Callback    102
VALUE   Cisco-Disconnect-Cause      Invalid-Protocol        120

END-VENDOR  Cisco
```

Organisation

- La configuration se trouve dans les fichiers situés dans `/etc/freeradius` et ses sous-répertoires
- Pour cette présentation, on distinguera plusieurs parties dans la configuration :
  - La configuration du dictionnaire RADIUS
  - La configuration de base du serveur
  - La configuration des politiques de gestion des requêtes
  - La configuration des modules
  - La configuration du roaming

Syntaxe de la config

- Le fichier **`/etc/freeradius/radiusd.conf`** est le point d'entrée de tout le reste de la configuration de freeRADIUS
- Sa syntaxe est assez simple, il est juste composé :
  - de définitions de variables (ex : `prefix = /usr`)
  - de noms de modules (ex : `ldap`), seuls sur une ligne
  - et de sections (ex : `authenticate { ... }`) qui peuvent elles-mêmes contenir tout ce qui précède, ainsi que des sous-sections (de façon récursive)
  - ...plus les commentaires qui peuvent apparaître n'importe où : `# exemple de commentaire`

$INCLUDE

- On peut inclure un fichier à tout point de la configuration avec la directive `$INCLUDE`
- On peut aussi inclure un répertoire : seront inclus tous les fichiers de ce répertoire dont le nom ne contient que des chiffres, des lettres, des points et des soulignés (_)

- C'est ainsi que la configuration de freeRADIUS est éparpillée dans une multitude de fichiers, avec notamment tous les fichiers des répertoires **`/etc/freeradius/modules`** et **`/etc/freeradius/sites-enabled`**, ainsi qu'un certain nombre des fichiers situés dans **`/etc/freeradius`**
- Cette organisation est bien plus claire que dans la version 1

Les variables

- Les valeurs des variables peuvent être précisées avec ou sans guillemets (simples ou doubles) :

```
exec_prefix = /usr
exec_prefix = '/usr'   # équivalent
exec_prefix = "/usr"   # encore équivalent
```

- La définition doit tenir sur une ligne, ou bien on doit terminer la ligne par un antislash :

```
nom = "mon nom est vrai\
ment long" # nom = "mon nom est vraiment long"
```

- La valeur d'une variable peut être utilisée plus loin dans la définition d'une autre variable avec la syntaxe `${var}` :

```
sbindir = ${exec_prefix}/sbin
```

- Cette évaluation a lieu uniquement au moment du lancement de freeRADIUS (donc il n'y a pas de coût à l'exécution)

Les sections

- La syntaxe est la suivante :

```
nom_de_la_section { # retour à la ligne obligatoire ici
   ...
} # obligatoirement seul sur une ligne, hormis les espaces
```

- Dans certains cas prédéfinis que nous verrons plus loin, un deuxième nom peut (ou doit) suivre le premier, par exemple :

```
...
authenticate {
    ...
    Auth-Type CHAP {
        ...
    }
    ...
}
...
```

radiusd.conf

- Voici le début du contenu par défaut de `radiusd.conf` :

```
prefix = /usr
exec_prefix = /usr
sysconfdir = /etc
localstatedir = /var
sbindir = ${exec_prefix}/sbin
logdir = /var/log/freeradius
raddbdir = /etc/freeradius
radacctdir = ${logdir}/radacct
name = freeradius
confdir = ${raddbdir}
run_dir = ${localstatedir}/run/${name}
db_dir = ${raddbdir}
libdir = /usr/lib/freeradius
pidfile = ${run_dir}/${name}.pid
#chroot = /path/to/chroot/directory

user = freerad
group = freerad

max_request_time = 30
cleanup_delay = 5
max_requests = 1024

#...
```

Chemin des répertoires et fichiers principaux (généralement à ne pas changer)

Utilisateur et groupe Un*x sous lequel le serveur tournera (généralement à ne pas changer)

Quelques paramètres de performances que l'on peut régler selon la charge du serveur (voir `radiusd.conf` pour plus de détails)

Sections `listen`

- Par défaut, freeRADIUS écoute sur toutes les IP du serveur (s'il en a plusieurs), et sur les ports RADIUS par défaut (c'est à dire 1812 pour le service d'authentification, et 1813 pour le service d'accounting)
- On peut changer ceci dans les sections `listen` de `radiusd.conf`

```
#...
listen {
      type = auth
      ipaddr = 10.1.2.3
      port = 0    # port standard pour l'authentification (c'est à dire 1812)
}
listen {
      ipaddr = 10.1.2.3
      port = 2001 # port non standard pour l'accounting
      type = acct
}
#...
```

- On peut aussi rajouter des sections `listen` si on le souhaite

Sections `listen`

- Options possibles pour une section `listen` :

```
listen {
      type = auth                       # Type du service offert
      ipaddr = *                        # Pour l'IPv4 (ici on écoute sur toutes les IP)
#     ipv6addr = ::                     # Pour l'IPv6 (idem, écoute sur toutes les IP)
      port = 0                          # Utiliser le port par défaut du service
#     interface = eth0                  # On peut préciser l'interface d'écoute
#     clients = per_socket_clients      # On peut limiter l'écoute à certains NAS
#     virtual_server = ma_policy        # On peut traiter les paquets selon une politique
                                        # particulière via un serveur virtuel donné
}
```

- Les types de services possibles sont les suivants :

```
#    auth     Service d'authentification (n'accepte que les paquets d'authentif.)
#    acct     Service d'accounting
#    proxy    Permet de préciser l'IP et le port source utilisés lorsque le serveur
#             sert de proxy vers un autre serveur RADIUS
#    detail   Utilisé pour synchroniser des serveurs RADIUS redondants. Cette
#             fonctionnalité remplace l'ancien démon «radrelay» de la version 1.
#    status   Ecoute les paquets Status-Server qui sont envoyés par l'outil radadmin
#    coa      Pour les requêtes CoA-Request et Disconnect-Request (cf. plus loin)
```

Sections `listen`

- Voir `sites-available/copy-acct-to-home-server` pour un exemple d'utilisation du type `detail`
- Voir `sites-available/status` pour un exemple d'utilisation du type `status`
- Voir `sites-available/originate-coa` pour un exemple d'utilisation du type `coa`

radiusd.conf (suite)

```
# ...

hostname_lookups = no
allow_core_dumps = no
regular_expressions     = yes
extended_expressions    = yes

security {
      max_attributes = 200
      reject_delay = 1
      status_server = no
}

thread pool {
      start_servers = 5
      max_servers = 32
      min_spare_servers = 3
      max_spare_servers = 10
      max_requests_per_server = 0
}

log {
      destination = files
      file = ${logdir}/radius.log
      syslog_facility = daemon
      stripped_names = no
      auth = no
      auth_badpass = no
      auth_goodpass = no
}

# ...
```

Activer ou désactiver le reverse DNS (pour logs), les *core dumps* et les expressions régulières (cf. plus loin)

Pour contrer quelques attaques connues

Gestion des threads

Gestion des logs

radiusd.conf (suite et fin)

```
#...

checkrad = ${sbindir}/checkrad

proxy_requests  = yes
$INCLUDE proxy.conf

$INCLUDE clients.conf

modules {
        $INCLUDE ${confdir}/modules/
        $INCLUDE eap.conf
#       $INCLUDE sql.conf
#       $INCLUDE sql/mysql/counter.conf
#       $INCLUDE sqlippool.conf
}

instantiate {
        exec
        expr
#       daily
        expiration
        logintime
}

$INCLUDE policy.conf

$INCLUDE sites-enabled/
```

Outil qui peut interroger un NAS pour vérifier si un utilisateur est connecté ou non

Configuration du roaming

Configuration des NAS

Configuration des modules

Forcer l'instanciation des modules (cf. plus loin)

Définition des politiques de traitement des requêtes

clients.conf

- Config de tous les NAS qui se connecteront à ce serveur

```
client localhost {
  ipaddr = 127.0.0.1
  secret = testing123
}
```

Pour pouvoir faire des tests depuis le serveur lui-même.

```
client salle-de-reunion.wifi.wifirst.fr {
  shortname = wifi_sdr
```

Le shortname est utilisé pour faire référence à ce NAS dans le reste de la configuration. C'est par défaut le nom précisé en début de section.

```
  ipaddr = 10.1.9.4
# ipv6addr = ::
# netmask = 32
```

L'adresse IP du NAS, ou tout un sous-réseau contenant un ou plusieurs NAS

```
  secret = "hEin/geo9c$be3Eet.ugh3le0eH"
```

Un excellent secret est indispensable

```
  require_message_authenticator = yes
```

Par défaut «no». Il vaut mieux le passer à «yes» si le NAS le permet.

```
  nastype = cisco
```

Utilisé par l'outil `checkrad` pour déterminer comment interroger le NAS

```
# virtual_server = politique_stricte
```

Pour que les requêtes de ce NAS soient traitées d'une façon particulière

```
# coa_server = coa
}
```

cf. plus loin

Organisation

- La configuration se trouve dans les fichiers situés dans `/etc/freeradius` et ses sous-répertoires
- Pour cette présentation, on distinguera plusieurs parties dans la configuration :
  - La configuration du dictionnaire RADIUS
  - La configuration de base du serveur
  - La configuration des politiques de gestion des requêtes
  - La configuration des modules
  - La configuration du roaming

Traitement des requêtes

Liste d'attributs

Requête

Ce qui suit est légèrement simplifié, on détaillera après

NAS

freeRADIUS

Access-Accept

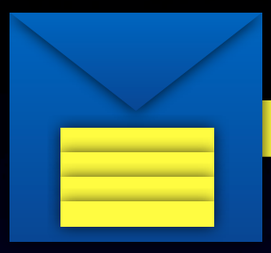
ou

Access-Reject

Réponse

Liste d'attributs

Recherche du NAS

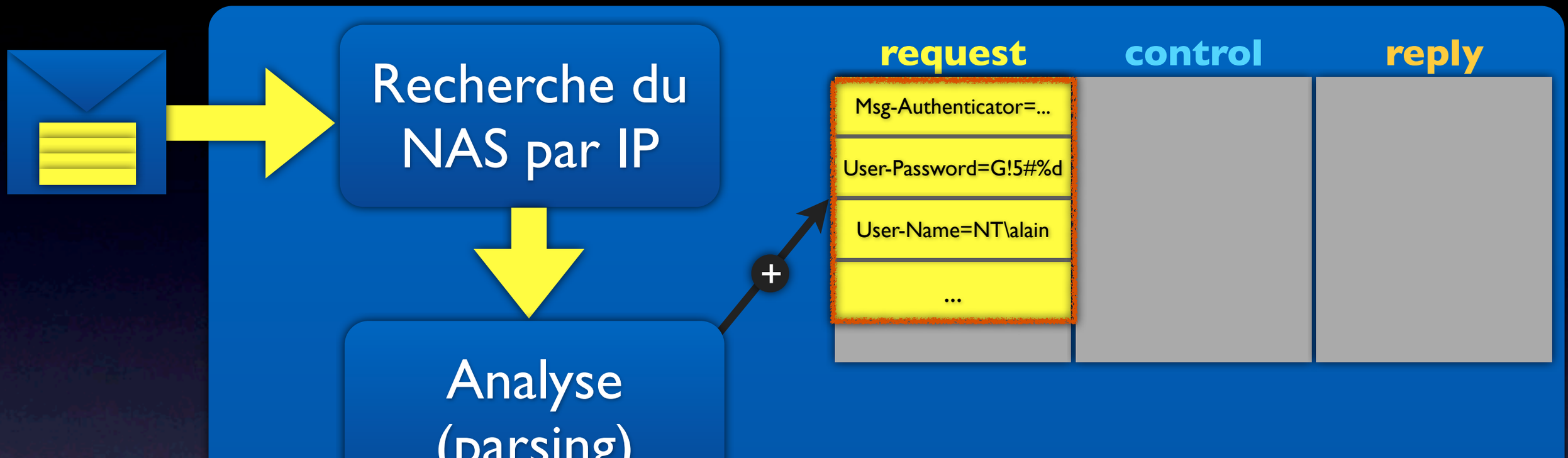


Recherche du NAS par IP

- Un NAS est toujours recherché par l'IP source du paquet RADIUS
- Si le NAS n'est pas trouvé, le paquet est ignoré
- La configuration de tous les NAS est chargée entièrement au démarrage de freeRADIUS, et est statique par la suite

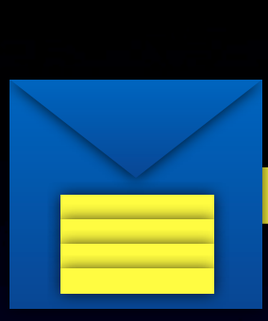
Pour rajouter ou modifier un NAS, il faut redémarrer freeRADIUS

Listes internes d'attributs



- Remarques :
  - Les attributs de contrôle sont parfois appelés des «config items»
  - Il y a d'autres listes : proxy-request, proxy-response, outer.request, outer.reply, coa, etc.

Phase d'autorisation



Recherche NAS par IP

Analyse (parsing)

request	control	reply
Msg-Authenticator=...		
User-Password=G!5#%d		
User-Name=NT\alain		
...		

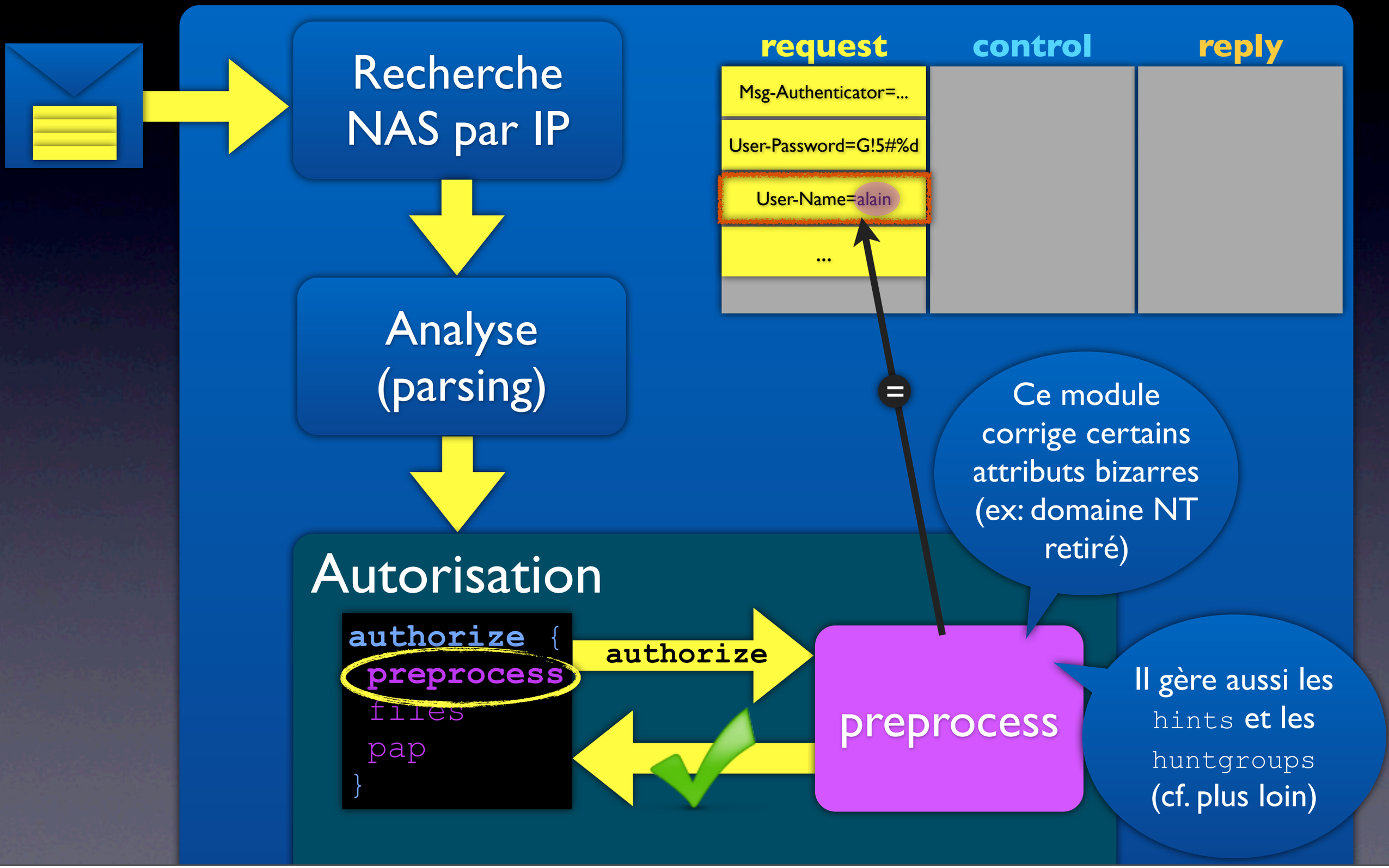
Autorisation

```
authorize {
 preprocess
 files
 pap
}
```

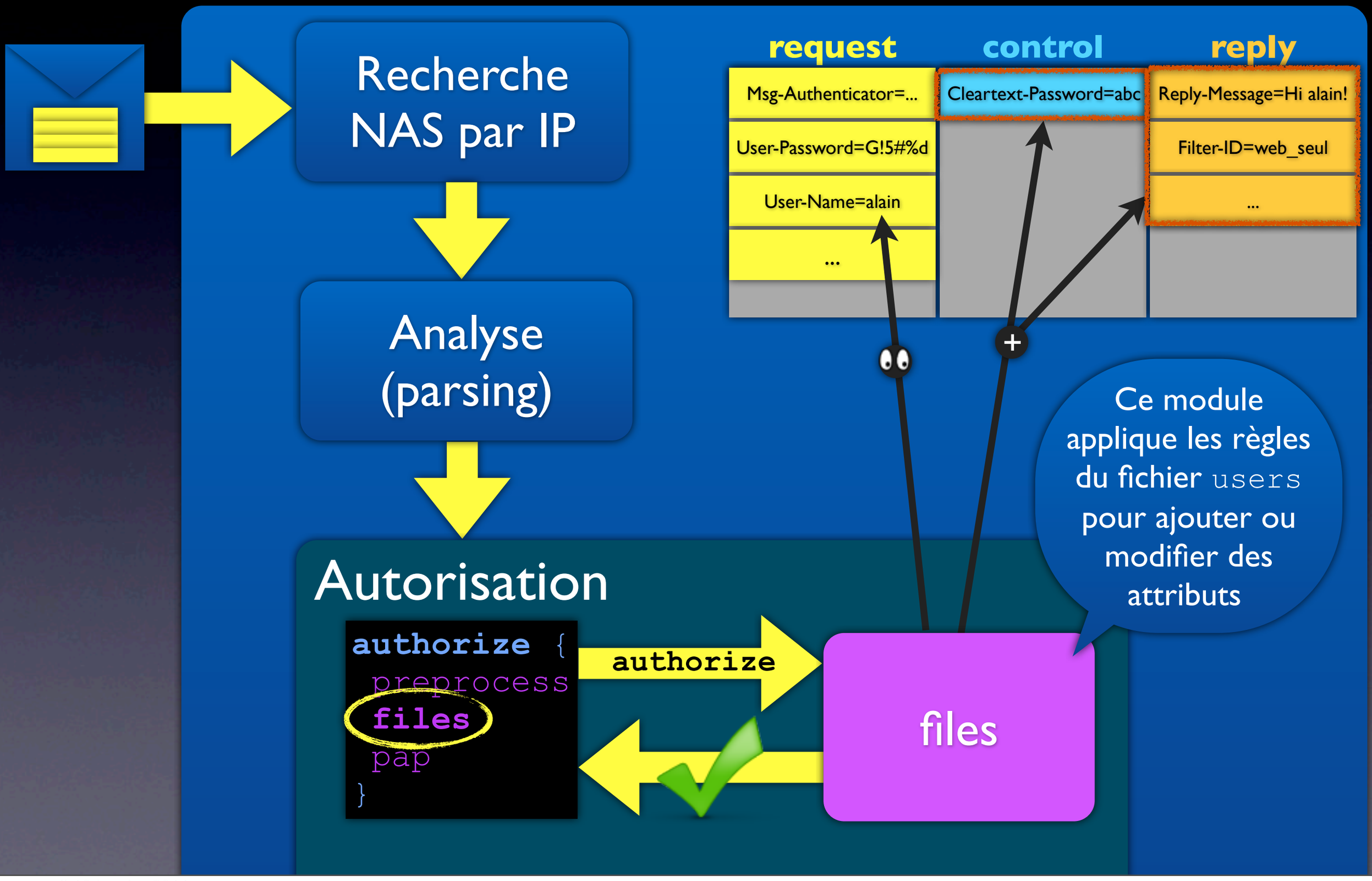
Liste de modules

/etc/freeradius/sites-enabled/default

Module preprocess



Module `files`



Un instant !

Les modules `files` et `preprocess` sont essentiels...
arrêtons-nous un instant pour les regarder de plus près

Fichier `users`

- Le module `files` consulte le fichier `users` dans le répertoire `/etc/freeradius` pour rajouter ou modifier des attributs dans les listes **`control`** et **`reply`** de freeRADIUS
- Ce fichier est composé d'une suite de règles, chaque règle ayant le format suivant :

```
login condition1, condition2, ..., operation_control1, operation_control2,...
        operation_reply1,
        operation_reply2,
        ...
```

Tabulation

une opération par ligne

Sur une même ligne

- Exemple :

```
alain Huntgroup-Name == "switch7_ports_1_a_12", Cleartext-Password := "abc"
        Reply-Message = "Hi alain!",
        Filter-ID = "web_seul"
```

Ne pas oublier les virgules

Parcours du fichier `users`

- Le fichier est parcouru dans l'ordre, jusqu'à trouver une règle dont le **`login`** corresponde à l'identifiant de l'utilisateur (indiqué par l'attribut **`User-Name`**) **et** dont les **`conditions`** soient toutes satisfaites (sinon freeRADIUS passe à la règle suivante)
- Dès qu'une règle satisfaisante est trouvée, ses opérations sont exécutées pour rajouter ou modifier des attributs dans la liste interne **`control`** et dans la liste **`reply`**, puis le module s'arrête là
- Note : dans la documentation de freeRADIUS, les **`conditions`** et les opérations sur les attributs de la liste **`control`** sont regroupées sous l'appellation «*check items*»

Format des conditions

- Chaque condition s'applique à un attribut de la liste `request` (ou de la liste `control` si l'attribut n'est pas trouvé dans la liste `request`)
- Les conditions sont exprimées selon la syntaxe `attribut opérateur valeur`
- Les opérateurs autorisés sont les suivants :

`==`	égal à
`!=`	différent de
`>`	strictement supérieur à (uniquement pour les attributs entiers)
`>=`	supérieur ou égal à (attributs entiers)
`<`	strictement inférieur à (attributs entiers)
`<=`	inférieur ou égal à (attributs entiers)
`=~`	correspond à l'expression régulière (attributs chaînes de caractères)
`!~`	ne correspond pas à l'expression régulière (attributs chaînes de caractères)
`=*`	est présent (la valeur est ignorée)
`!*`	n'est pas présent (la valeur est ignorée)

Format des opérations

- Pour rajouter ou modifier un attribut dans la liste `control` ou la liste `reply`, la syntaxe est encore : `attribut opérateur valeur`
- Les opérateurs autorisés sont les suivants :

`=` ajoute l'attribut avec la valeur indiquée, ou ne fait rien s'il existe déjà

`:=` ajoute l'attribut avec la valeur indiquée, ou écrase sa valeur s'il existe déjà

`+=` ajoute l'attribut avec la valeur indiquée, même s'il existe déjà

- **Attention** : on se mélange souvent entre `=`, `:=` et `==`

  Il faut bien comprendre que les opérations (`=`, `:=` et `+=`) ne sont exécutées que si le `login` correspond et si les conditions (`==`, `>=`...) sont toutes remplies

L'attribut `Fall-Through`

- Par défaut, dès que le module `files` a trouvé une règle satisfaisante, il applique les opérations de cette règle puis s'arrête
- Si l'on souhaite qu'il continue à parcourir les règles, il suffit de rajouter **`Fall-Through=Yes`** à la fin de la règle, c'est à dire à la fin des opérations sur les attributs **`reply`** (cet attribut n'est pas rajouté dans la liste d'attributs **`reply`**)
- Exemple :

```
alain Cleartext-Password := "abc"
        Reply-Message = "Hi alain!",
        Fall-Through = Yes

alain Huntgroup-Name == "switch7_ports_1_a_12"
        Filter-ID = "web_seul"

alain Huntgroup-Name == "switch7_ports_13_a_24"
        Filter-ID = "voip_seule"
```

Dans tous les cas le mot de passe est «abc» et le message de bienvenue «Hi alain!»

Condition sur le lieu et le port de connexion (cf. plus loin)

Identifiant du filtrage que le NAS devra appliquer

Le login DEFAULT

- Si l'on écrit **DEFAULT** à la place du **login**, alors la règle est appliquée pour tous les utilisateurs, pourvu que toutes les conditions soient remplies
- Exemple:

```
DEFAULT
        Reply-Message = "Hi %{User-Name}!",
        Fall-Through = Yes

DEFAULT Huntgroup-Name == "switch7_ports_1_a_12"
        Filter-ID = "web_seul",
        Fall-Through = Yes

DEFAULT Huntgroup-Name == "switch7_ports_13_a_24"
        Filter-ID = "voip_seule",
        Fall-Through = Yes

alain   Cleartext-Password := "abc"

pierre Cleartext-Password := "def"

jean    Cleartext-Password := "ghi"

DEFAULT Auth-Type := Reject
        Reply-Message = "Non! Login inconnu."
```

Ceci sera dynamiquement substitué par la valeur de l'attribut **User-Name**

Tout le monde aura donc un message personnalisé avec son propre identifiant

Pour ceux qui se connectent sur les ports 1 à 12 du switch 7, l'accès sera limité au Web

Et sur les ports 13 à 24 l'accès sera limité à la voix sur IP

Si on arrive ici, c'est que l'utilisateur est inconnu : on le rejette donc, en écrasant le message défini plus haut

Substitutions dynamiques

- La syntaxe `%{attribut}` sera substituée (on parle de *translation*, noté *xlat*) par la valeur de l'attribut en question

  ➡ Exemple : `"abc%{User-Name}def"` sera substitué par `"abcjoedef"` pour l'utilisateur `joe`

- A ne pas confondre avec la syntaxe `${variable}` que l'on a vue précédemment, et qui n'est utilisée qu'au démarrage

- On peut aussi utiliser : `"%{%{attribut}:-pardefaut}"` qui est substituée par la valeur de l'attribut, ou par la chaîne `"pardefaut"` si l'attribut n'est pas défini

- On peut imbriquer ces substitutions, par exemple : `"%{%{Stripped-User-Name}:-%{%{User-Name}:-inconnu}}"`

  Sera substitué par la valeur de `Stripped-User-Name`, ou s'il n'est pas défini, par la valeur de `User-Name`, ou s'il n'est pas défini par `inconnu`

Substitutions dynamiques

- Certains modules offrent une fonction de substitution que l'on peut utiliser avec la syntaxe `%{nom_du_module:parametres}`
- C'est le cas notamment des modules `sql`, `ldap`, `expr`, `exec` et `perl`. Exemples :
  - `%{sql:select credit from credits where login='%{User-Name}'}`
  - `%{ldap:ldap:///dc=company,dc=com?uid?sub?uid=%u}`
  - `%{expr:2*%{Session-Timeout}+10}`
  - `%{exec:/usr/bin/mon_prog %{User-Name} %{Ses`
  - `%{perl:%{User-Name} %{Session-Timeout}}`

Fonction perl personnalisable

- Les paramètres peuvent eux-mêmes contenir des substitutions : chaque module se charge de ces substitutions en les échappant si nécessaires. Par exemple, si le `User-Name` est `joe's`, alors le module `sql` substitue `%{User-Name}` par `joe=27s` (encodage MIME) avant d'exécuter la requête SQL.

Les *huntgroups*

- Un *huntgroup* est un ensemble de lieux et/ou de ports de connexion (le module `preprocess` doit être activé pour pouvoir les utiliser)
- On peut ensuite filtrer sur **`Huntgroup-Name == "..."`** pour appliquer des règles différentes selon les *huntgroups*
- Ils sont définis dans le fichier `/etc/freeradius/huntgroups`
- Exemple de configuration:

```
switch7_ports_1_a_12  NAS-IP-Address == 10.4.3.2, NAS-Port-Id == 1-12
switch7_ports_13_a_24 NAS-IP-Address == 10.4.3.2, NAS-Port-Id == 13-24
switchs_1_a_3 NAS-IP-Address == 10.4.3.2
switchs_1_a_3 NAS-IP-Address == 10.4.3.3
switchs_1_a_3 NAS-IP-Address == 10.4.3.4
ports_voip NAS-IP-Address == 10.4.3.2, NAS-Port-Id == 13-24
ports_voip NAS-IP-Address == 10.4.3.3, NAS-Port-Id == 1,3-7,9
ports_voip NAS-IP-Address == 10.4.3.4, NAS-Port-Id == 1,10-15
```

Un *huntgroup* peut être composé de plusieurs NAS

Et même de plusieurs ensembles de ports sur plusieurs NAS différents

Les *hints*

- Le module `preprocess` permet également de définir des *hints* (indices) : il s'agit de préfixes ou suffixes que l'utilisateur peut rajouter à son identifiant pour indiquer le service souhaité
- Ils sont définis dans le fichier `/etc/freeradius/hints` selon un format identique à celui du fichier `/etc/freeradius/users`
- Exemple de configuration:

```
DEFAULT Suffix == ".ppp", Strip-User-Name = Yes
    Hint = "PPP",
    Service-Type = Framed-User,
    Framed-Protocol = PPP
```

Le **User-Name** sera modifié dans la requête pour supprimer le suffixe

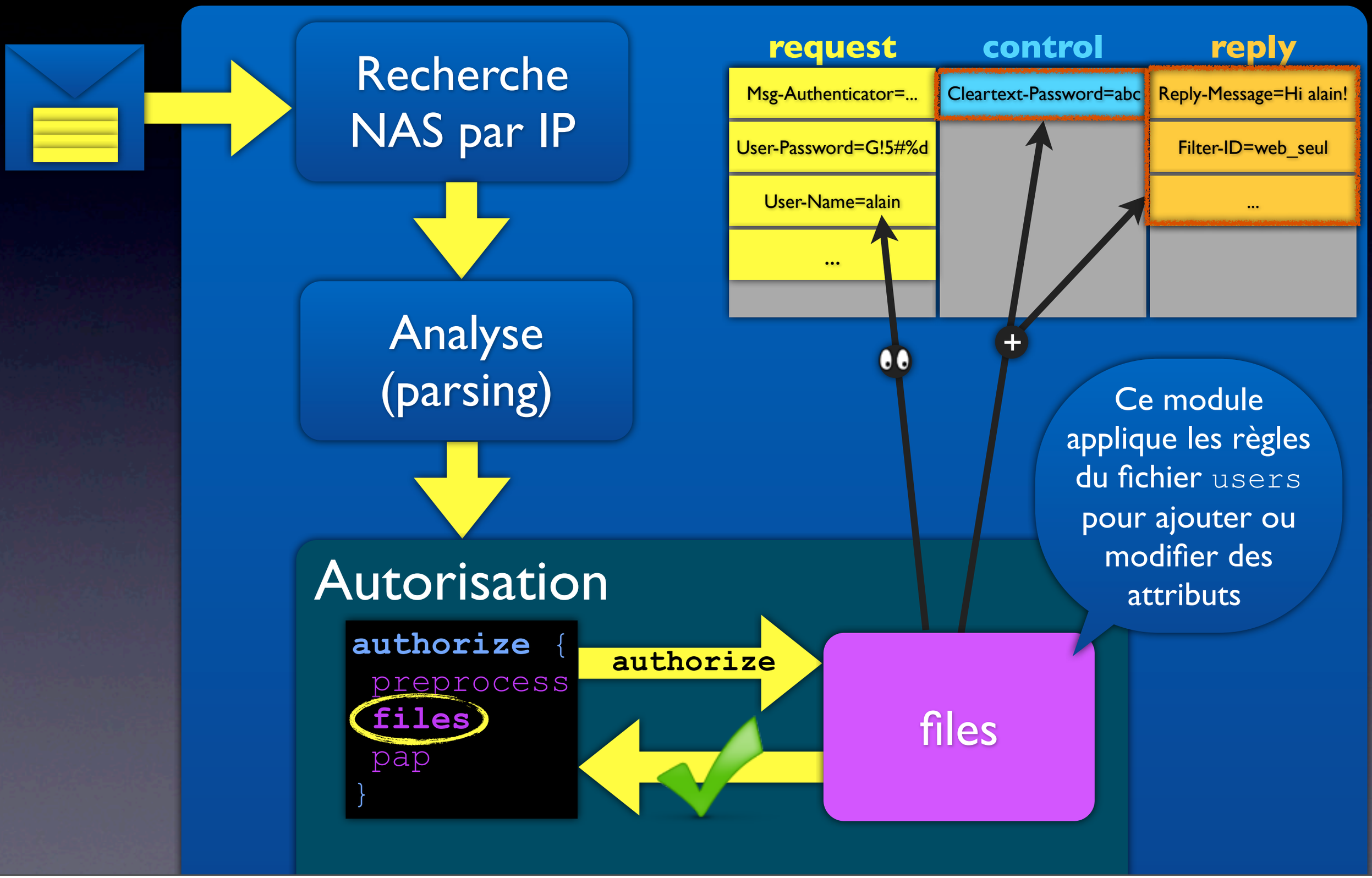
- Dans cet exemple, si le **User-Name** de l'utilisateur se termine par "**.ppp**" alors les attributs **Hint**, **Service-Type** et **Framed-Protocol** sont rajoutés à la liste interne **request**

Attention : contrairement au fichier `users`, le fichier `hints` modifie la requête !

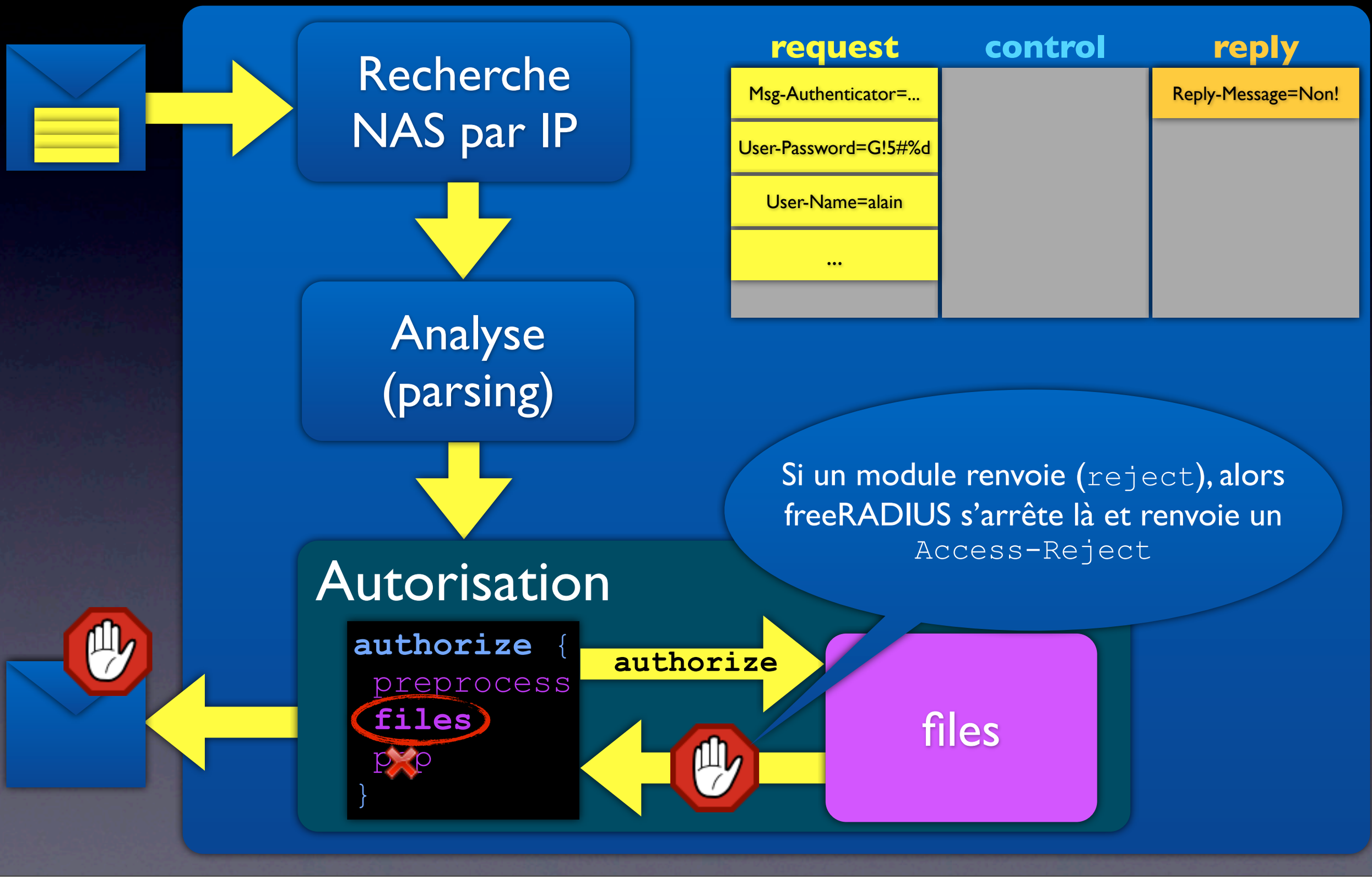
On reprend...

Nous en étions au traitement d'une requête par le **module** `files` **dans la section** `authorize`

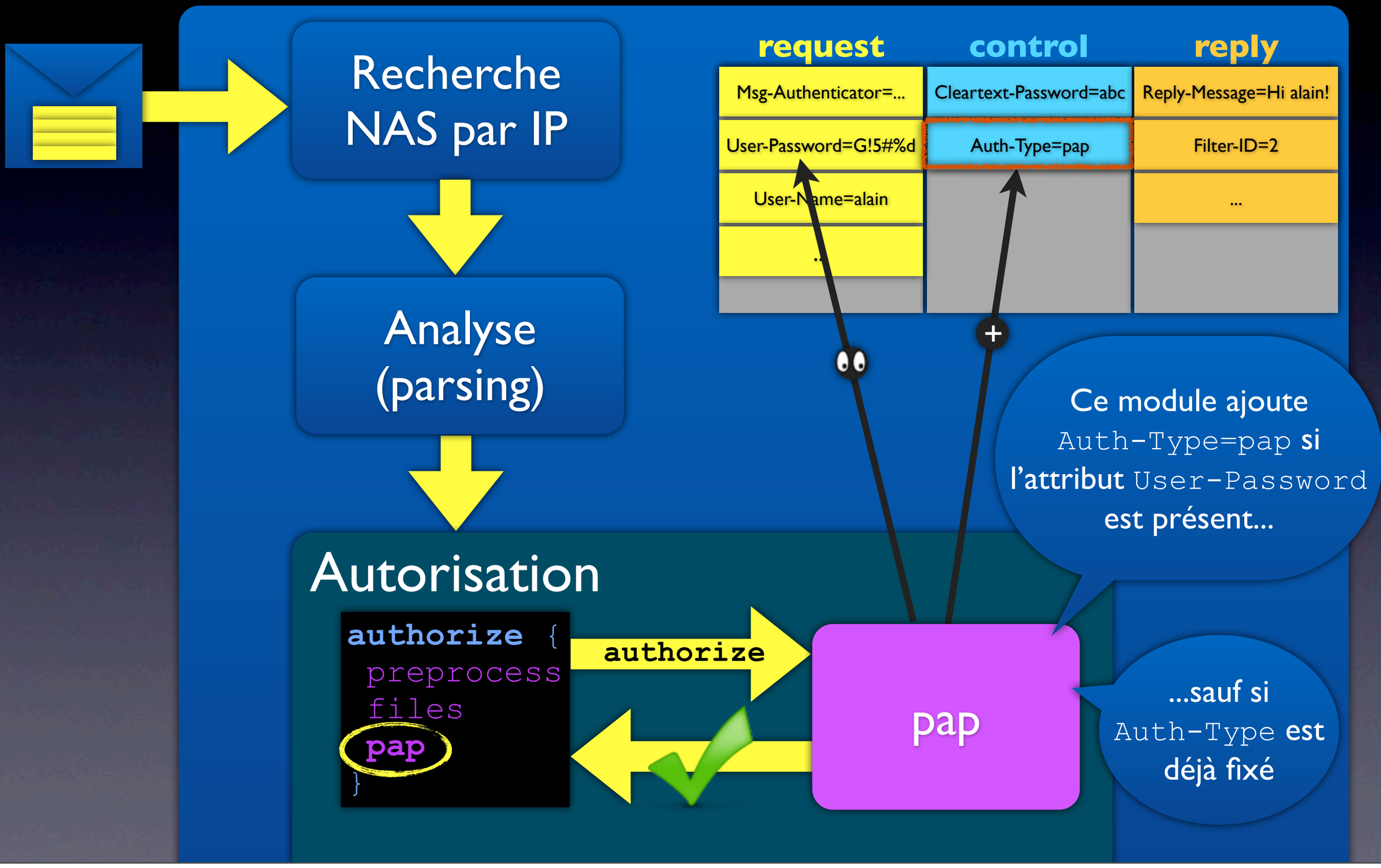
Module `files`



Rejet d'un utilisateur



Module pap

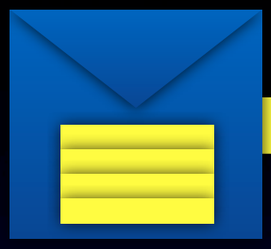

Recherche NAS par IP
Analyse (parsing)
Autorisation
authorize {
preprocess
files
pap
}
authorize
pap
request
control
reply
Msg-Authenticator=...
User-Password=G!5#%d
User-Name=alain
...
Cleartext-Password=abc
Auth-Type=pap
Reply-Message=Hi alain!
Filter-ID=2
...
+
Ce module ajoute Auth-Type=pap si l'attribut User-Password est présent...
...sauf si Auth-Type est déjà fixé

Sous-section `Autz-Type`

- Dans la section `authorize`, des sous-sections au format «`Autz-Type` *XXX*» peuvent être définies
- La section `authorize` est d'abord exécutée *sans* ces sous-sections
- Si l'attribut `Autz-Type` est défini à l'issue de la section `authorize`, et s'il n'y a pas eu de rejet, alors la sous-section correspondante est exécutée seule
- Cela permet de choisir dynamiquement une politique particulière d'autorisation

`Autz-Type` = **Auth**ori**z**ation Type
`Auth-Type` = **Auth**entication Type

Authentification



Recherche
NAS + analyse

Autorisation

request	control	reply
Msg-Authenticator=...	Cleartext-Password=abc	Reply-Message=Hi alain!
User-Password=G!5#%d	Auth-Type=pap	Filter-ID=2
User-Name=alain		...
...		

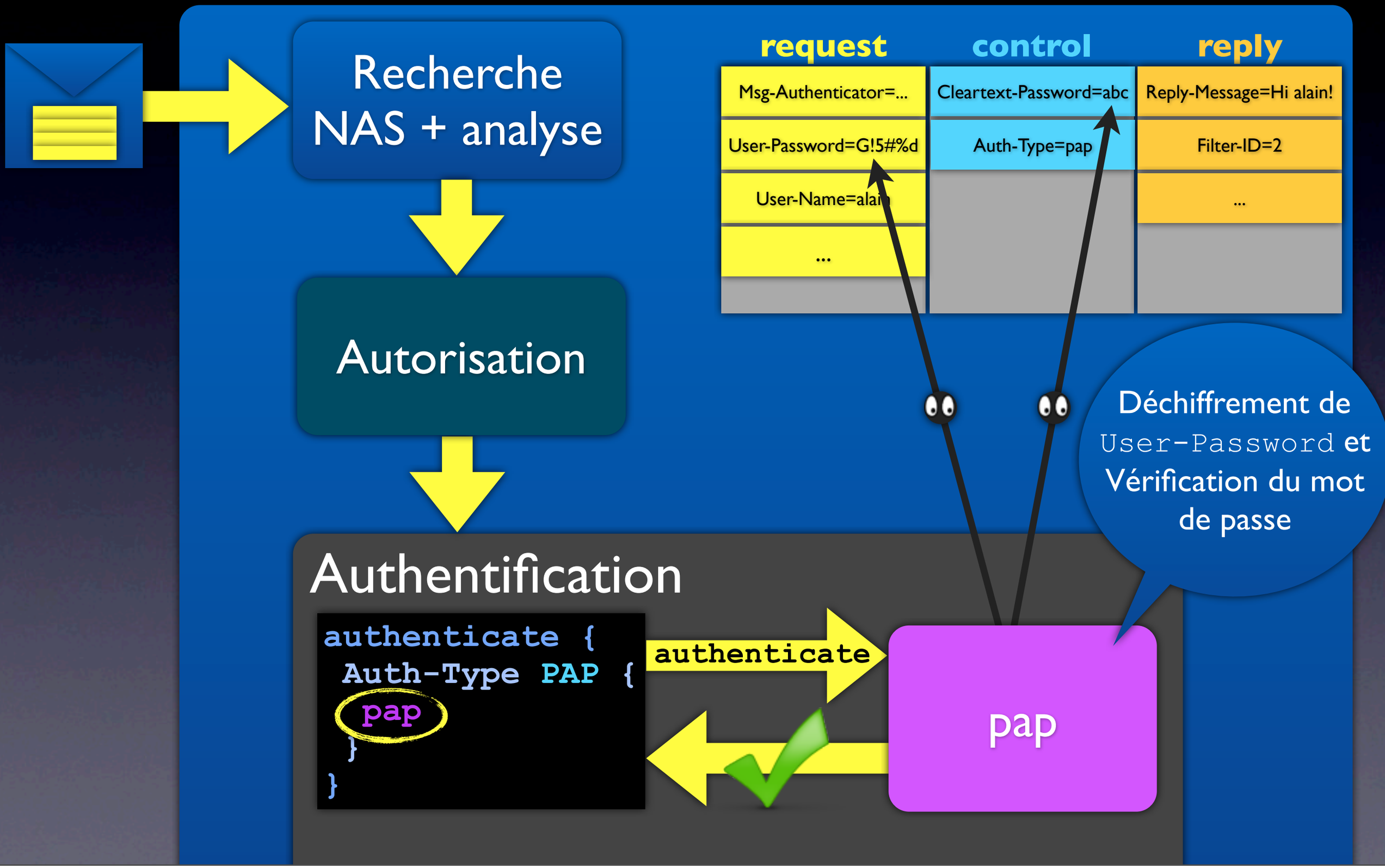
Authentification

```
authenticate {
 Auth-Type PAP {
  pap
 }
}
```

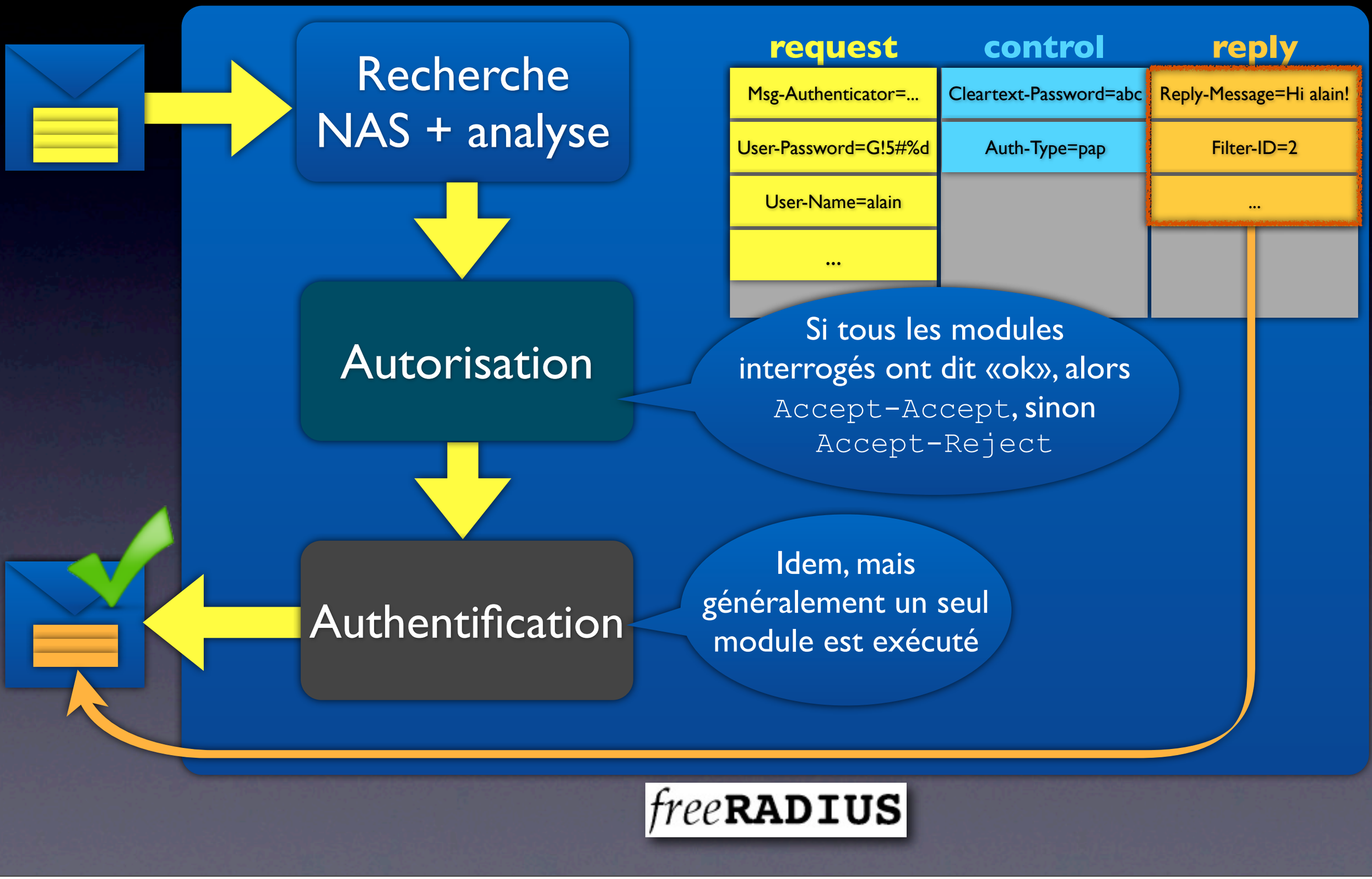
/etc/freeradius/sites-enabled/default

Si l'attribut `Auth-Type` est défini et qu'une sous-section correspondant à sa valeur existe, alors elle est exécutée seule, sinon le contenu de la section `authenticate` (sans les sous-sections) est exécuté dans l'ordre.

Module pap (encore)



Réponse !



freeRADIUS

Info d'authentification

- Selon la méthode d'authentification choisie, une information différente est envoyée au serveur par le NAS, par exemple :
    - PAP : mot de passe chiffré avec le secret RADIUS
    - CHAP : *hash* MD5 du mot de passe + défi + ID-CHAP
    - EAP/MD5 : idem
    - TTLS/PAP : mot de passe chiffré dans un tunnel TLS
    - PEAP/MS-CHAP-v2 : *hash* de *hash* NT dans un tunnel TLS
    - EAP/TLS : certificat TLS du client
    - ...

Vérification du mot de passe

- Si on renseigne le mot de passe de chaque utilisateur en clair dans le fichier `users` (avec `Cleartext-Password:="a3d$G4"`) alors freeRADIUS est en mesure de vérifier le mot de passe de l'utilisateur :
  - pour PAP et TTLS/PAP : le mot de passe transmis est déchiffré et simplement comparé au mot de passe du fichier `users`
  - pour CHAP et EAP/MD5 : freeRADIUS calcule le *hash* MD5 du mot de passe du fichier `users` + le défi et l'ID-CHAP reçus, et il compare le résultat au *hash* MD5 transmis
  - pour PEAP/MS-CHAP-v2 : freeRADIUS applique l'algorithme MS-CHAP-v2 pour calculer le *hash* adéquat à partir du mot de passe du fichier MD5 et des informations reçues dans la requête RADIUS (défi MS-CHAP-v2), et il compare le résultat au *hash* reçu

Mot de passe en clair ?

Par souci de sécurité, il est fortement **déconseillé** de stocker les mots de passe des utilisateurs en clair

Stockage du mot de passe

- Dans le fichier `users`, on peut stocker un hash du mot de passe plutôt que le mot de passe lui-même :
  - `Crypt-Password` : hash crypt Unix
  - `MD5-Password` : hash MD5
  - `SMD5-Password` : hash MD5 avec sel
  - `SHA-Password` : hash SHA1
  - `SSHA-Password` : hash SHA1 avec sel
  - `NT-Password` : hash Windows NT
  - `LM-Password` : hash Windows Lan Manager

Incompatibilité des *hash*

- Malheureusement, un *hash* est par définition une fonction irréversible, c'est à dire qu'on ne peut pas deviner le mot de passe si l'on ne connaît que le *hash* (à moins d'essayer tous les mots de passe possibles)
- Du coup, si l'on stocke par exemple le *hash* SHA1 du mot de passe des utilisateurs, alors on ne pourra pas utiliser la méthode d'authentification CHAP car freeRADIUS ne pourra pas savoir si le *hash* SHA1 du fichier `users` et le *hash* MD5 transmis ont été calculés à partir du même mot de passe
- D'ailleurs, puisque le *hash* MD5 transmis avec la méthode CHAP n'est pas un *hash* du mot de passe seul (mais un *hash* du mot de passe plus un défi et un identifiant CHAP), on ne peut pas utiliser la méthode CHAP en stockant un *hash* MD5 du mot de passe !

Tableau de compatibilité

- Le tableau suivant indique, pour chaque méthode d'authentification, les formats de stockage compatibles :

Méthode \ Stockage	Clair	Crypt	MD5	SHA1	SMD5	SSHA1	NT	LM
PAP	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
CHAP	Oui	Non	Non	Non	Non	Non	Non	Non
EAP/MD5	Oui	Non	Non	Non	Non	Non	Non	Non
TTLS/PAP	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
PEAP/MS-CHAP-v2	Oui	Non	Non	Non	Non	Non	Oui	Oui

Méthodes sûres ?

Si un pirate a accès aux échanges entre le NAS et le serveur RADIUS, alors non seulement il a accès aux informations non chiffrées des paquets RADIUS, mais il peut aller plus loin, selon la méthode utilisée :

- La méthode PAP est la moins sûre de toutes, car elle est vulnérable à une attaque de dictionnaire hors-ligne et elle est également vulnérable à une attaque de redite

➡ Attaque de dictionnaire hors ligne : un pirate saisit un mot de passe quelconque, et capte l'échange RADIUS qui en résulte, puis hors connexion, il essaie des millions de secrets RADIUS jusqu'à trouver celui qui produit le même mot de passe chiffré. Il connait alors le secret RADIUS et peu déchiffrer tous les mots de passe suivants.

➡ Attaque de redite (*replay*) : le pirate capte un échange RADIUS réussi et le répète ultérieurement. Il parvient à se connecter avec l'identité d'un autre utilisateur (sans connaître son mot de passe).

Méthodes sûres ?

- La méthode CHAP est également vulnérable aux attaques de redite et aux attaques de dictionnaire hors ligne. Toutefois, lorsque le pirate est parvenu à trouver le secret RADIUS, il ne peut que déchiffrer les mots de passe des utilisateurs qui s'authentifient avec la méthode PAP (ainsi que les autres attributs chiffrés avec le secret RADIUS).
- La méthode EAP/MD5 a les mêmes vulnérabilités que la méthode CHAP.

- Les méthodes qui reposent sur un tunnel TLS sont invulnérables à la fois aux attaques de dictionnaire hors ligne et aux attaques de redite : elles sont donc beaucoup plus sûres.
- Leur seul inconvénient est qu'elles exigent que l'utilisateur vérifie le certificat du serveur... et tous les utilisateurs ne le font pas.

Utilisez de préférence une méthode PEAP ou TTLS

EAP/TLS, EAP/SIM, EAP/GTC

- La méthode EAP/TLS ne transmet aucun mot de passe : lors de l'authentification, l'utilisateur vérifie le certificat du serveur, et le serveur vérifie le certificat de l'utilisateur

➡ Cette méthode est invulnérable aux attaques de dictionnaire hors-ligne et même en-ligne (car aucun mot de passe n'est échangé) et est invulnérable aux attaques de redite. C'est donc une méthode encore plus sûre que les méthodes PEAP et TTLS...

➡ ...mais elle est pénible à mettre en oeuvre car il faut au préalable installer un certificat TLS sur le poste de chaque utilisateur

- On aura un niveau de sécurité encore supérieure avec une carte à puce, car l'utilisateur possède alors à la fois un objet physique et un secret (le code PIN) : on parle d'authentification à deux facteurs (2FA). Cela suppose une infrastructure matérielle et logicielle plus complexe, donc c'est réservé généralement aux opérateurs et grandes entreprises. La méthode EAP utilisée est soit EAP/SIM (opérateurs mobiles) soit EAP/GTC (pour les autres cartes de sécurité)

Tableau de compatibilité

- Révisons donc le tableau de compatibilité, pour plus de sécurité :

Méthode \ Stockage	Clair	Crypt	MD5	SHA1	SMD5	SSHA1	NT	LM
PAP	Stockage peu sûr	Méthodes peu sûres						
CHAP								
EAP/MD5								
TTLS/PAP		Oui	Oui	Oui	Oui	Oui	Oui	Oui
PEAP/MS-CHAP-v2		Non	Non	Non	Non	Non	Oui	Oui

Réponses des modules

Jusqu'à présent, nous avons considéré qu'un module renvoyait soit «**succès**» soit «**échec**», et qu'en cas d'échec, le traitement de la requête s'arrêtait immédiatement. En réalité, chaque module peut renvoyer l'un des codes suivants :

Code	Signification	Que faire ?
`notfound`	L'utilisateur n'a pas été trouvé	**Continuer**
`noop`	Module non applicable (il n'a donc rien fait)	**Continuer**
`ok`	Utilisateur accepté	**Continuer**
`updated`	Utilisateur accepté et listes d'attributs mises à jour	**Continuer**
`fail`	Le module a échoué (ex: problème d'accès à la BD)	**Stop + Rejeter**
`reject`	Utilisateur rejeté	**Stop + Rejeter**
`userlock`	Utilisateur rejeté car son compte a été bloqué	**Stop + Rejeter**
`invalid`	Utilisateur rejeté car sa configuration est incorrecte	**Stop + Rejeter**
`handled`	Le module a géré la requête tout seul	**Stop + Rien**

Priorité des résultats

- Que faire, par exemple, si dans la section `authorize` un module a répondu **`noop`**, puis le suivant a répondu **`notfound`** et le dernier a répondu **`noop`** ? La logique veut que le résultat de la section `authorize` soit **`notfound`** (et freeRADIUS doit donc rejeter l'utilisateur).
- Si l'un des modules avait répondu **`ok`**, il aurait fallu que le résultat de la section soit lui-même **`ok`** (et freeRADIUS aurait alors dû poursuivre le traitement de la requête *via* la section `authenticate`)
- On voit donc que, si aucun module n'a renvoyé un échec immédiat, il faut établir une échelle de priorité entre les résultats obtenus pour déterminer le résultat de la section elle-même
- Par défaut, l'échelle de priorité est la suivante : **`updated`** > **`ok`** > **`notfound`** > **`noop`**

Priorité des réponses

Le tableau suivant indique le niveau de priorité par défaut des réponses possibles des modules (la priorité **`return`** indique que la section s'arrête immédiatement avec le code en question) :

Code	Signification	Priorité
`notfound`	L'utilisateur n'a pas été trouvé	1
`noop`	Module non applicable (il n'a donc rien fait)	2
`ok`	Utilisateur accepté	3
`updated`	Utilisateur accepté et listes d'attributs mises à jour	4
`fail`	Le module a échoué (ex. problème réseau)	**return**
`reject`	Utilisateur rejeté	**return**
`userlock`	Utilisateur rejeté car son compte a été bloqué	**return**
`invalid`	Utilisateur rejeté car sa configuration est incorrecte	**return**
`handled`	Le module a géré la requête tout seul	**return**

Modifier les priorités

- Dans certains cas, les règles de priorité par défaut ne sont pas satisfaisantes
- Par exemple, on peut souhaiter arrêter immédiatement le traitement d'une section si un module répond **ok**
- Pour cela, il suffit d'accoler une section au nom du module, avec les règles de priorité que l'on souhaite modifier, par exemple :

```
authorize {
  preprocess
  sql
  ldap
}
```

```
authorize {
  preprocess
  sql {
    ok = return
    updated = return
  }
  ldap
}
```

- Dans cet exemple, si le module `sql` répond **ok** (ou **updated**), alors la section `authorize` s'arrête immédiatement et répond elle-même **ok** (ou **updated**) : le module `ldap` n'est pas interrogé

Regrouper des modules

- On peut regrouper plusieurs modules dans une section `group`
- Les modules de cette section sont appelés les uns après les autres, chacun renvoie un code, et le plus prioritaire d'entre eux devient le code de retour du groupe (le traitement de la section `group` s'interrompt si un code a pour priorité **`return`**)
- Ceci peut être utile pour mettre en place un *fail-over* entre plusieurs modules (bascule automatique vers un ou plusieurs module de secours en cas d'échec d'un module), par exemple :

```
authorize {
  preprocess
  sql_principal
  ldap
}
```

```
authorize {
  preprocess
  group {
    sql_principal {
      fail = 1
      default = return
    }
    sql_de_secours
  }
  ldap
}
```

On ne fait appel au serveur SQL de secours *que* si le serveur principal est injoignable

Priorités du groupe

- On peut également modifier les règles de priorité de la réponse du groupe lui-même
- Par exemple, imaginons que nous ne souhaitions pas interroger le serveur LDAP si l'un des serveurs SQL a répondu **ok** (ou **updated**) :

```
authorize {
 preprocess
 sql_principal {
  ok = return
  updated = return
 }
 ldap
}
```

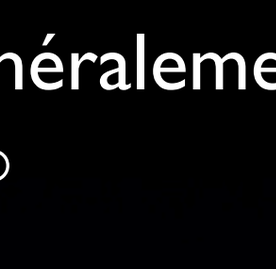
```
authorize {
 preprocess
 group {
  sql_principal {
   fail = 1
   default = return
  }
  sql_de_secours
  ok = return
  updated = return
 }
 ldap
}
```

Sections redundant

- Pour le *fail-over*, on utilise généralement une section `redundant` plutôt qu'une section `group`
- C'est la même chose, sauf que les règles de priorité par défaut dans une section `redundant` sont **`fail = 1`** et **`default = return`**

```
authorize {
 preprocess
 group {
  sql_principal {
   fail = 1
   default = return
  }
  sql_de_secours1 {
   fail = 1
   default = return
  }
  sql_de_secours2

  ok = return
  updated = return
 }
 ldap
}
```



```
authorize {
 preprocess
 redundant {
  sql_principal
  sql_de_secours1
  sql_de_secours2

  fail = return
  ok = return
  updated = return
 }
 ldap
}
```

Répartion de charge

- Pour répartir la charge entre plusieurs modules (par exemple trois bases de données), on peut utiliser une section `load-balance` :

```
authorize {
 preprocess
 load-balance {
  sql1
  sql2
  sql3
 }
}
```

- L'un des modules est choisi au hasard et exécuté, et son résultat devient le résultat de la section `load-balance` elle-même (même si le module échoue avec le code **`fail`**)

- Si l'on souhaite basculer vers l'un des autres modules en cas de **`fail`**, alors on peut utiliser une section `redundant-load-balance` :

➡ La section n'échoue que si *tous* ses modules échouent

Le «langage» `unlang`

- Avant la version 2 de freeRADIUS, si l'on souhaitait exprimer une condition dans la configuration des politiques de gestion des requêtes, on n'avait généralement pas d'autre choix que d'écrire un module
- Désormais, si l'on souhaite exprimer des conditions assez simples, on peut le faire à l'aide d'expressions de type `if`, `else`, `elsif`, etc., par exemple :

```
authorize {
 preprocess
 if (User-Name == "joe") {
  ldap1
 }
 elsif (User-Name == "jack") {
  ldap2
 }
 else {
  sql
 }
}
```

Dans cet exemple, si l'utilisateur est `joe`, alors on fait appel au module `ldap1`, sinon si c'est `jack`, alors on utilise le module `ldap2`, et sinon on utilise le module `sql` (pour tous les autres utilisateurs, donc)

Note: une section `load-balance` ou `redundant-load-balance` ne doit pas contenir de section `else` ou `elseif`. Une section `redundant` ne doit pas contenir de section `if`, `else` ou `elseif`.

Le «langage» `unlang`

- Le langage `unlang` n'est pas un langage complet, et n'a pas vocation à le devenir (d'où son nom, que l'on peut traduire par «non langage») : il ne sert qu'à exprimer des règles simples (pour appliquer une logique complexe, il faut écrire un module, cf. plus loin)
- La condition d'un bloc `if` peut être :
  - `(attribut opérateur valeur)`

    Exemple: `(Session-Timeout >= 3600)`
  - `(valeur)`

    Vrai si la valeur est non vide et non nulle, ex: `(Idle-Timeout)` ou encore `("%{Idle-Timeout}")`
  - `(code_retour)`

    Vrai si c'est le code retour du dernier module appelé, ex: `(fail)`
- Comme avec le langage C, on peut utiliser `!a` pour «non a», `a && b` pour «a et b», et enfin `a || b` pour «a ou b»
- On peut imbriquer les conditions, par exemple `(a && !(b || c))`

Le «langage» unlang

- On peut également utiliser l'instruction `switch` / `case`, avec le même sens que dans le langage C, par exemple :

```
authorize {
 preprocess
 switch "%{User-Name}" {
  case "joe" {
    ldap1
  }
  case "jack" {
    ldap2
  }
  case {
    sql
  }
 }
}
```

Cette chaîne de caractères peut contenir des substitutions dynamiques...

...mais pas les valeurs des sections `case`

Cette section est la section par défaut

- Cet exemple aura le même comportement que celui que nous avons vu précédemment avec les instructions `if`, `elsif` et `else`

Le «langage» `unlang`

- Par défaut, les attributs sont recherchés dans la liste **`request`**
- On peut indiquer une autre liste d'attributs avec la syntaxe : **`%{liste:attribut}`** par exemple **`%{control:Auth-Type}`**
- Outre les listes **`request`**, **`control`** et **`reply`**, que nous avons vues, on a aussi :
  - **`proxy-request`** et **`proxy-reply`** qui contiennent respectivement les attributs envoyés à un *Home-Server*, et reçu en réponse, lorsque le serveur freeRADIUS sert de *Proxy-Server*
  - **`outer.request`**, **`outer.reply`**, **`outer.control`**, **`outer.proxy-request`**, et **`outer.proxy-reply`** qui donnent accès aux différentes liste d'attributs de la requête externe depuis l'intérieur d'un tunnel PEAP ou TTLS

Le «langage» `unlang`

- Il existe encore d'autres options de substitution:
  - `%{#chaine}` : longueur de la chaîne
  - `%{attribut[n]}` : (n+1)ème attribut de ce type
  - `%{attribut[#]}` : nombre d'attributs de ce type
  - `%{attribut[*]}` : tous les attributs de ce type séparés par un retour chariot (`\n`) et regroupés en une seule chaîne
  - `%{0}` : la chaîne identifiée par la dernière expression régulière (avec `=~` ou `!~`)
  - `%{1}`, `%{2}`, ..., `%{8}` : les groupes identifiés par la dernière l'expression régulière
- Cependant, s'il on a besoin de cette complexité, il faut se poser la question de développer plutôt un nouveau module (en python, en perl ou en C, comme nous le verrons plus loin)

Le «langage» unlang

- On peut enfin définir une section `update`, qui permet simplement de mettre à jour une liste d'attributs, par exemple :

```
update reply {
  Reply-Message := "Bonjour %{User-Name}"
  Session-Timeout <= 3600
  Filter-Id !* ALL
}
```

- Toutes les listes peuvent être modifiées (**`request`**, **`control`**...)

- Les opérateurs **`=`**, **`:=`** et **`+=`** ont le même sens que précédemment, et l'on peut également utiliser les opérateurs suivants :

`-=` : supprime tous les attributs de ce type qui ont la valeur indiquée
`==` : supprime tous les attributs de ce type, sauf ceux qui ont la valeur indiquée
`!*` : supprime tous les attributs de ce type (quelle que soit la valeur indiquée)
`<=` : remplace les valeurs supérieures par la valeur indiquée (attributs entiers uniquement)
`>=` : remplace les valeurs inférieures par la valeur indiquée (attributs entiers uniquement)

Pour **`<=`** et **`>=`** l'attribut est rajouté avec la valeur indiquée s'il n'existe pas

Module `always`

- Le module `always` répond toujours le même résultat, et ce résultat est configurable. Voici la configuration par défaut de ce module :

```
always fail {
        rcode = fail
}
always reject {
        rcode = reject
}
always noop {
        rcode = noop
}
always handled {
        rcode = handled
}
always updated {
        rcode = updated
}
always notfound {
        rcode = notfound
}
always ok {
        rcode = ok
        simulcount = 0
        mpp = no
}
```

Voici un exemple d'utilisation de la variante **`reject`** du module **`always`**

```
authorize {
  preprocess
  if (User-Name=="mechant") {
    reject
  }
  ...
}
```

policy.conf

- Si l'on pense utiliser un morceau de code *unlang* à plusieurs endroits dans la configuration, alors il est bon de définir une section avec le code en question dans la section `policy` qui se trouve dans le fichier `/etc/freeradius/policy.conf`, par exemple :

```
policy {
  ajout_message_accueil {
    update reply {
      Reply-Message := "Bonjour %{User-Name}"
    }
  }
}
```

- Cette «fonction» peut ensuite être utilisée dans la configuration :

```
authorize {
  preprocess
  ajout_message_accueil
  ...
}
```

Module `policy`

- Il existe un module `policy` qui offre une fonctionnalité un peu similaire, mais plus limitée
- **ATTENTION** : ce module n'a rien à voir avec le fichier `policy.conf` et ce que nous venons de voir
- Il est maintenant conseillé d'utiliser de préférence `policy.conf`, et d'ignorer complètement le module `policy`
- Note: le module `policy` repose sur un fichier de paramétrage que l'on peut également ignorer : `/etc/freeradius/policy.txt`

Politique par défaut

- La politique (*policy*) de traitement des paquets utilisée par défaut est définie dans `/etc/freeradius/sites-enabled/default`
- C'est un lien symbolique vers le fichier `/etc/freeradius/sites-available/default`
- Ce fichier contient les sections de modules décrits précédemment : `authorize` et `authenticate`, ainsi que plusieurs autres sections similaires...

Autres sections de modules

Il y a plusieurs autres points d'entrées pour les modules, selon le même principe que pour **`authorize`** et **`authenticate`** :

- **`session`** : si l'attribut `Simultaneous-Use` a été rajouté à la liste de contrôle (par exemple par le module `files` pendant la phase d'autorisation), alors les modules listés ici vont s'assurer que le nombre maximum de sessions simultanées précisé par cet attribut n'est pas atteint, et rejeter l'utilisateur si c'est le cas
- **`post-auth`** : actions à lancer après l'authentification

Autres sections de modules

- Si un paquet est rejeté à tout moment de la phase d'autorisation ou d'authentification, alors la sous-section '`Post-Auth-Type REJECT`' de la section `post-auth` est exécutée

- Ceci est souvent utilisé pour ajouter des attributs dans le paquet de réponse `Access-Reject`
  - par exemple ajouter l'attribut `Reply-Message` pour renvoyer un message d'erreur au NAS (qu'il pourra par exemple afficher à l'utilisateur)

Autres sections de modules

Deux sections de modules sont appelées lors du traitement d'une requête de type `Accounting-Request` :

- `preacct` : liste de modules appelés avant l'accounting
- `accounting` : liste de modules qui gèrent l'accounting

Autres sections de modules

Et enfin deux sections de modules sont appelées par le serveur freeRADIUS avant et après le «proxying» d'un paquet vers un Home-Server, en cas de roaming :

- `pre-proxy` : modules appelés avant le *proxying* d'un paquet
- `post-proxy` : modules appelés lorsque la réponse arrive

Serveurs virtuels

Toutes les sections de *policy* que nous avons vues peuvent également être définies à l'intérieur d'une section '**`server nom_du_serveur`**' :

```
server politique-ldap {
 authorize {
  preprocess
  ldap
 }
 authenticate {
  Auth-Type LDAP {
   ldap
  }
 }
 ...
}
```

On appelle cela un «serveur virtuel»

Serveurs virtuels

- On peut ainsi définir plusieurs serveurs virtuels et configurer pour chacun une politique de traitement des paquets différente

- On configure généralement chaque serveur virtuel dans un fichier à part dans le répertoire `sites-available`...

- ...et on crée un lien symbolique vers chacun des fichiers (que l'on veut activer) dans le répertoire `sites-enabled`

Serveurs virtuels

Une fois les serveurs virtuels définis, on peut configurer freeRADIUS pour qu'il sélectionne dynamiquement tel ou tel serveur virtuel (autrement dit telle ou telle politique de gestion des paquets), en fonction de :

- l'adresse IP et le port auxquels le paquet a été envoyé : dans la section `listen` correspondante, ajouter `virtual-server=nom_du_serveur_virtuel`
- le NAS d'où provient la requête, en procédant de la même façon dans la section `client` adéquate
- le Home-Server vers lequel un paquet est redirigé, en cas de *roaming*, toujours de la même façon dans la configuration du Home Server (utile pour définir des sections `pre-proxy` et `post-proxy` spécifiques à chaque partenaire de roaming)

Organisation

- La configuration se trouve dans les fichiers situés dans `/etc/freeradius` et ses sous-répertoires
- Pour cette présentation, on distinguera plusieurs parties dans la configuration :
  - La configuration du dictionnaire RADIUS
  - La configuration de base du serveur
  - La configuration des politiques de gestion des requêtes
  - La configuration des modules
  - La configuration du roaming

Configuration des modules

- Sauf exception (`eap.conf`, `sql.conf`...), la configuration des modules se trouve dans les fichiers du répertoire `/etc/freeradius/modules`
- La configuration d'un module a le format suivant :

```
nom_du_module {
 un_param = 23
 autre_param = "blabla"
 ...
}
```

OU

```
nom_du_module autre_nom {
 un_param = 23
 autre_param = "blabla"
 ...
}
```

- Si l'on précise un autre nom, alors c'est cet autre nom qui doit être utilisé dans le reste de la configuration

Configuration des modules

- Exemple de la configuration du module **`files`** :

```
files {
        # The default key attribute to use for matches.  The content
        # of this attribute is used to match the "name" of the
        # entry.
        #key = "%{Stripped-User-Name:-%{User-Name}}"

        usersfile = ${confdir}/users
        acctusersfile = ${confdir}/acct_users
        preproxy_usersfile = ${confdir}/preproxy_users

        #  If you want to use the old Cistron 'users' file
        #  with FreeRADIUS, you should change the next line
        #  to 'compat = cistron'.  You can the copy your 'users'
        #  file from Cistron.
        compat = no
}
```

/etc/freeradius/modules/files

Configuration des modules

- Exemple de la configuration du module **realm** :

```
realm suffix {
        format = suffix
        delimiter = "@"
}

realm realmpercent {
        format = suffix
        delimiter = "%"
}
```

/etc/freeradius/modules/realm

- On peut alors utiliser ces deux variantes du modules **realm** dans le reste de la configuration en employant les noms **suffix** et **realmpercent**

Configuration des modules

- La configuration de certains modules peut utiliser des sous-sections pour regrouper des paramètres reliés entre eux
- Par exemple la sous-section `tls` dans le module `ldap` :

```
ldap {
     server = "ldap.example.com"
     identity = "cn=admin,dc=example,dc=com"
     ...
     tls {
          start_tls = no
          cacertfile = /path/to/cacert.pem
          ...
     }
     ...
}
```

/etc/freeradius/modules/ldap

Configuration de EAP

- C'est le cas aussi de la configuration de l'EAP :

```
eap {
        default_eap_type = md5
        timer_expire     = 60
        ...
        md5 {
        }
        ...
        tls {
                certdir = ${confdir}/certs
                cadir = ${confdir}/certs
                ...
        }
        ...
        peap {
                default_eap_type = mschapv2
                copy_request_to_tunnel = yes
                use_tunneled_reply = no
        #       proxy_tunneled_request_as_eap = yes
                virtual_server = "inner-tunnel"
        }
        mschapv2 {
        }
}
```

/etc/freeradius/eap.conf

Certains sous-modules n'ont pas de configuration, mais si on souhaite les utiliser, il faut les mentionner

La configuration de l'EAP/TLS est nécessaire pour le PEAP et le TTLS

Si ce paramètre est défini, alors les requêtes EAP internes au tunnel PEAP seront traitées par le serveur virtuel indiqué ici, sinon, ce sera le serveur virtuel qui a traité la requête externe

L'un des rares modules dont la configuration n'est pas dans le répertoire `modules`

Instanciation des modules

- Lorsque freeRADIUS démarre, il analyse les fichiers de configuration et trouve la liste de tous les modules susceptibles d'être utilisés

  ➡ Curieusement, il ignore les modules utilisés uniquement dans des substitutions (ex: `%{expr:2+3}`)

- Une fois la liste établie, il crée une instance de chacun des modules, et appelle leur fonction d'initialisation

- Pour préciser l'ordre d'instanciation ou instantier des modules supplémentaires (notamment les modules utilisés uniquement dans des substitutions), il suffit de les lister dans la section `instantiate` du fichier `radiusd.conf` :

```
instantiate {
  exec
  expr
  sql
}
```

Instanciation des modules

Voici un exemple pour lequel la section `instantiate` est indispensable :

- Dans sa fonction d'instanciation, le module `sql` charge la liste des NAS dans la table `nas` de la base de données (ils viennent se rajouter aux NAS listés dans le fichier `clients.conf`)
- Si l'on souhaite utiliser la base de données uniquement pour gérer la liste des NAS, alors il faudra rajouter le module `sql` dans la section `instantiate`, car il ne sera pas utilisé ailleurs

Modules virtuels

- Si on définit une section dotée d'un nom dans la section `instantiate`, alors c'est la définition d'un «module virtuel», par exemple :

```
instantiate {
  ...
  redundant redundant_sql {
    sql1
    sql2
    sql3
  }
}
```

- On peut utiliser ce module virtuel n'importe où, comme un module normal :

```
authorize {
  preprocess
  redundant_sql
}
```

- C'est une fonctionnalité redondante car on peut déjà faire la même chose avec `policy.conf`, comme on l'a vu

Change of Authorization (CoA)

- Rien n'était prévu à l'origine dans le protocole RADIUS pour demander au NAS de déconnecter un utilisateur ou de changer ses droits d'accès au cours de sa session
- Dans la RFC 3576, deux nouveaux types de requêtes RADIUS ont été définies à cet effet : type `disconnect` pour déconnecter un utilisateur, et type `coa` pour changer ses droits d'accès (quand on parle de CoA au sens large, on parle implicitement de ces deux types)
- Attention : dans ce cas, c'est le NAS qui joue le rôle du serveur RADIUS, et n'importe qui peut jouer le rôle du client, pourvu qu'il ait un secret RADIUS partagé avec le NAS
- Voici par exemple comment demander au NAS qui se trouve à l'adresse `10.2.3.4`, port `1812`, de déconnecter l'utilisateur `alain` :

```
echo "User-Name=alain" | radclient 10.2.3.4:1812 disconnect "s9$G...s!df"
```

On peut fournir une paire *Attribut*=*Valeur* par ligne

IP

port

type

secret

CoA depuis freeRADIUS

- Bien que les requêtes CoA puissent être envoyées par n'importe qui, il est parfois utile que freeRADIUS en envoie lui-même, par exemple :
    - Si on veut déconnecter un utilisateur d'un premier NAS au moment où il se connecte à un second NAS
    - Si on découvre, au moment d'une requête d'accounting de type `Interim-Update`, que les droits de l'utilisateur ont changé
    - Si on veut envoyer une requête à freeRADIUS pour qu'il cherche lui-même le NAS auquel est connecté un utilisateur et lui envoie lui-même la requête CoA
- *Note* : pour l'instant, freeRADIUS ne sait pas recevoir de requête de type CoA, et ne sait pas non plus rediriger des requêtes de type CoA depuis un *Home-Server* vers un NAS

CoA depuis freeRADIUS

- Le CoA est encore peu supporté dans les NAS
- Sa configuration dans freeRADIUS est jeune et risque de changer dans les prochaines versions
- Pour en savoir plus, lire :

  `/etc/freeradius/sites-available/`**`originate-coa`**

Organisation

- La configuration se trouve dans les fichiers situés dans `/etc/freeradius` et ses sous-répertoires
- Pour cette présentation, on distinguera plusieurs parties dans la configuration :
  - La configuration du dictionnaire RADIUS
  - La configuration de base du serveur
  - La configuration des politiques de gestion des requêtes
  - La configuration des modules
  - La configuration du roaming

Exemple de *roaming*

- Rappel : en cas de *roaming*, le serveur freeRADIUS redirige certaines requêtes vers un (ou plusieurs) *Home-Servers*
- Pour savoir quelles requêtes rediriger et vers quel *Home-Server* les envoyer, on utilise en général le **`User-Name`** de l'utilisateur
- On peut, par exemple, configurer freeRADIUS pour qu'il redirige les requêtes dont le **`User-Name`** est **`joe%machin.com`** vers le serveur RADIUS situé à l'adresse `rad1.machin-telecoms.net`.
- Dans notre exemple, on redirigera les requêtes vers un serveur secondaire si le premier est indisponible

Identifier le *realm*

- La première étape du *roaming* consiste à identifier les requêtes qui devront être redirigées vers un *Home-Server*
- Pour cela, le serveur freeRADIUS cherche l'attribut interne **`Realm`** dans la liste **`control`** à la fin de la phase d'autorisation
- Si cet attribut est renseigné, alors le paquet est redirigé vers le *Home-Server* (ou l'un des *Home-Servers*) configuré(s) pour le *realm* en question
- Généralement on utilise le module **`realm`** pour fixer l'attribut **`Realm`** à partir d'un préfixe ou d'un suffixe de l'attribut `User-Name`

Module *realm*

- Comme on l'a vu plus haut, la configuration par défaut du module `realm` définit la variante `realmpercent`
- Dans notre exemple, il nous suffit donc de rajouter `realmpercent` dans la section `authorize` pour que le *realm* soit identifié à partir du suffixe du `User-Name`, après le caractère `%`
- Lorsque le serveur recevra une requête de **`joe%machin.com`**, le module `realmpercent` rajoutera l'attribut **`Realm`** dans la liste **`control`** avec pour valeur **`machin.com`**, pendant la phase d'autorisation
- Ce module rajoutera également l'attribut **`Stripped-User-Name`** dans la liste de **`control`**, avec pour valeur **`joe`**

proxy.conf

- Le coeur de la config du *roaming* est dans `proxy.conf`
- Ce fichier est composé de plusieurs sections :
  - une section **`proxy server`** pour la configuration générale du *roaming*
  - Une section **`home_server`** pour chaque *Home-Server* vers lequel on est susceptible de rediriger des requêtes
  - une ou plusieurs sections **`home_server_pool`** qui permettent de définir des règles de répartition de charge entre des *Home-Servers*
  - Une ou plusieurs sections **`realm`** pour indiquer quel **`home_server_pool`** utiliser pour chaque *realm*

proxy.conf

- Voici un exemple de configuration de `proxy.conf` :

```
proxy server {
   default_fallback = no
}
home_server rad1_machin_telecoms {
   type = auth
   ipaddr = 212.3.4.5
   port = 1812
   secret = testing123
   require_message_authenticator = yes

   response_window = 20
   zombie_period = 40

# revive_interval = 120

   status_check = status-server
   check_interval = 30
   num_answers_to_alive = 3
}
home_server rad2_machin_telecoms
   type = auth
   ...
}
... # plus bas: config des home_server_pools et realms
```

Seul paramètre prévu pour cette section générale (cf. plus loin)

Config du *Home-Server* primaire

Config du *Home-Server* secondaire

proxy.conf

```
}
home_server rad1_machin_te
   type = auth
   ipaddr = 212.3.4.5
   port = 1812
   secret = testing123
   require_message_authenticator = yes

   response_window = 20
   zombie_period = 40

#  revive_interval = 120

   status_check = status-server
   check_interval = 30
   num_answers_to_alive = 3
}
home_server rad2_machin_telecoms
   type = auth
   ...
}
... # plus bas: config des home_server_pools et realms
```

auth ou **acct** ou **auth+acct**

L'adresse IP et le port UDP vers lequel rediriger les requêtes, et le secret partagé avec ce *Home-Server* (pour le *Home-Server*, le *Proxy-Server* n'est qu'un NAS normal)

Il vaut mieux renseigner une IP plutôt qu'un nom de domaine, car si la résolution DNS échoue, freeRADIUS ne démarre pas

Ce *Home-Server* s'attend-il à un attribut **Message-Authenticator** dans chaque requête ? Si oui, freeRADIUS le rajoute.

Ces paramètres permettent d'éviter de rediriger des requêtes vers des serveurs qui ne répondent plus

proxy.conf

```
proxy server {
   default_fallback = no
}
home_server rad1_machin_telecoms {
   type = auth
   ipaddr = 212.3.4.5
   port = 1812
   secret = testing123
   require_message_authentic

   response_window = 20
   zombie_period = 40

#  revive_interval = 120

   status_check = status-server
   check_interval = 30
   num_answers_to_alive = 3
}
home
   ty
   ...
}
...
```

Si ce *Home-Server* ne répond pas pendant 20s, alors il devient un **zombie** (interrogé uniquement s'il n'y a plus de *Home-Server* **vivant**). Après 40s, il est considéré comme vraiment **mort** (plus interrogé du tout). Si l'on active `revive_interval=120`, alors il ressuscitera après 2 min...

...mais il est préférable d'interroger régulièrement le *Home-Server* en envoyant des requêtes de statut : ici, on teste ce *Home-Server* toutes les 30s, et il faut 3 succès pour le ressusciter

Le *Home-Server* doit lui-même être configuré pour accepter les requêtes de statut, bien sûr. Si c'est un serveur freeRADIUS, on doit créer (dans *sa* config) une section `listen` avec le type `status`, puis activer `status_server=yes` dans la section `security`, et enfin créer un serveur virtuel dédié (voir l'exemple fourni avec freeRADIUS)

templates.conf

- Les paramètres des *Home-Servers* sont souvent très proches
- Pour éviter les répétitions, on peut donc définir des modèles (*templates* en anglais) de configuration :

```
home_server rad1-pas-cher {
  $template home_server
  ipaddr = 212.3.4.5
  secret = "FRc0...7FL3b8"
}
home_server rad2-pas-cher {
  $template home_server
  ipaddr = 212.3.4.6
  secret = "GDCd...M1$N3z"
}

home_server rad1-machin {
  template = modele-machin
  ipaddr = 212.3.4.7
}
home_server rad2-machin {
  template = modele-machin
  ipaddr = 212.3.4.8
}
```

/etc/freeradius/proxy.conf

```
templates {
  home_server {
      response_window = 20
      zombie_period = 40
      revive_interval = 120
  }

  home_server modele-machin {
      type = auth
      port = 1812
      secret = "ApQj4...3g2sD"
      response_window = 20
  }
}
```

/etc/freeradius/**templates.conf**

templates.conf

- Les modèles peuvent être utilisés dans d'autres endroits de la configuration, pourvu que ce soit dans des sections situées à la racine (c'est à dire pas dans des sous-sections)
- Cela peut être utile aussi, par exemple, pour la définition des NAS dans `clients.conf`
- Pour les sous-sections, on peut parvenir à quelque chose de similaire, mais moins pratique, avec l'instruction **`$INCLUDE`**

proxy.conf

- Voici la suite (et fin) du fichier `proxy.conf` :

```
...
home_server_pool machin_telecoms_pool {
   type = fail-over
   virtual_server = pre_post_proxy_pour_machin
   home_server = rad1_machin_telecoms
   home_server = rad2_machin_telecoms
}
realm machin.com {
   auth_pool = machin_telecoms_pool
   nostrip
}
```

On configure ici un pool composé des deux *Home-Servers* définis plus haut

Tous les *Home-Servers* d'un pool doivent être du même type (`auth` ou `acct` ou `auth+acct`)

Et ici on indique que le *realm* `machin.com` repose sur ce pool

- soit on précise `auth_pool` (*Home-Servers* de type `auth`) et/ou `acct_pool` (*Home-Servers* de type `acct`)
- soit on précise `pool` (*Home-Servers* de type `auth+acct`)
- soit on ne précise aucun pool, auquel cas le *realm* est géré localement (pas de redirection)

proxy.conf

- Voici la suite (et fin) du fichier `proxy.conf` :

Ce pool est de type **`fail-over`**, c'est à dire que le premier *Home-Server* est utilisé en priorité, sauf s'il ne répond plus auquel cas on utilise le *Home-Server* suivant

```
...
home_server_pool machin_telecoms_pool {
   type = fail-over
   virtual_server = pre_post_proxy_pour_machin
   home_server = rad1_machin_telecoms
   home_server = rad2_machin_telecoms
}
realm machin.com {
   auth_pool = machin_telecoms_pool
   nostrip
}
```

On peut indiquer un serveur virtuel dont les sections **`pre-proxy`** et **`post-proxy`** seront exécutées avant et après la redirection de la requête vers le *Home-Server*

Par défaut, si un attribut **`Stripped-User-Name`** est présent dans la liste **`control`**, alors sa valeur est utilisée pour définir l'attribut **`User-Name`** de la requête redirigée vers le *Home-Server*. Par exemple, le *Home-Server* recevra une requête pour **`joe`**, et non pour **`joe%machin.com`**. L'option **`nostrip`** permet d'indiquer que l'on souhaite conserver le **`User-Name`** complet (ce qui peut être utile si le *Home-Server* doit lui-même relayer encore la requête).

Autres types de pools

Dans notre exemple, nous utilisons un pool de type `fail-over`, mais il existe d'autres types :

- `load-balance` : chaque requête est envoyée aléatoirement vers l'un des *Home-Servers* (avec une préférence pour les *Home-Servers* qui répondent bien aux requêtes)
  - ***Attention*** : les méthodes d'authentification EAP ne fonctionneront sans doute pas, car elles requièrent plusieurs échanges vers un même serveur
- `client-balance` : également aléatoire, mais pour un NAS donné, on utilise toujours le même *Home-Server* (tant qu'il est **vivant**)
- `keyed-balance` : aléatoire mais on conserve le même *Home-Server* pour une même valeur de l'attribut interne `Load-Balance-Key`
  - ➡ Un module doit donc au préalable rajouter cet attribut dans la liste `control`, par exemple en copiant la valeur de l'attribut `User-Name`

Realm NULL et LOCAL

- Si l'on définit un *realm* **NULL**, alors il est utilisé pour toutes les requêtes qui n'ont pas de *realm*
- On définit souvent un *realm* **LOCAL** sans *pool* (donc traité localement) : on peut alors forcer l'utilisation de ce *realm* en ajoutant l'attribut `Proxy-To-Realm` dans la liste **control**, avec pour valeur **LOCAL**
- Par exemple, on ne souhaite généralement pas rediriger le contenu d'un tunnel PEAP ou TTLS vers un autre serveur (pour des raisons de sécurité). Pour cela, on peut écrire ce qui suit dans la configuration du serveur virtuel `inner-tunnel` :

```
update control {
    Proxy-To-Realm := LOCAL
}
```

Home-Server virtuel

- Si on omet tous les paramètres d'une section `home_server` et qu'à la place on renseigne le paramètre `virtual-server`, alors les requêtes adressées à ce «*Home-Server* virtuel» seront traitées localement par le serveur virtuel choisi
- Exemple :

```
home_server virtuel_pour_machin {
  virtual_server = server_virtuel_machin
}
```

- Ceci est utile par exemple pour exécuter certaines tâches lorsque tous les *Home-Servers* d'un *pool* ont échoué :

```
home_server_pool machin_telecoms_pool {
  type = fail-over
  home_server = rad1_machin_telecoms
  home_server = rad2_machin_telecoms
  home_server = virtuel_pour_machin
}
```

Puisque le *pool* est de type `fail-over`, on essaie d'abord `rad1`, et s'il échoue on essaie `rad2`, et sinon on appelle le serveur virtuel

Home-Server de secours

- Dans la configuration d'un **`home_server_pool`**, on peut définir un *Home-Server* de repli (*fallback* en anglais), utilisé lorsque tous les *Home-Servers* du *pool* sont morts (on choisit souvent un serveur virtuel) :

```
home_server_pool machin_telecoms_pool {
  type = load-balance
  home_server = rad1_machin_telecoms
  home_server = rad2_machin_telecoms
  fallback = virtuel_pour_machin
}
```

Dans cet exemple, la charge est répartie entre les serveurs `rad1` et `rad2`. Si les deux sont morts, alors le serveur **`virtuel_pour_machin`** est utilisé.

- Si aucun serveur de *fallback* n'est défini, et si le paramètre `default_fallback = yes` dans la section **`proxy server`**, alors le *realm* **`DEFAULT`** est utilisé quand tous les Home-Servers d'un realm sont morts

➡ *On le configure souvent avec un simple pool contenant un seul serveur virtuel dont le rôle est simplement d'historiser l'échec.*

Filtrer les attributs

- Dans un contexte de *roaming*, il faut souvent s'assurer que les attributs renvoyés par le *Home-Server* sont acceptables, et les supprimer dans le cas contraire
- C'est l'objectif du module **`attr_filter`** dont voici un extrait de la configuration par défaut :

```
attr_filter attr_filter.post-proxy {
    attrsfile = ${confdir}/attrs
}
...
```

`/etc/freeradius/modules/attr_filter`

- On peut alors activer **`attr_filter.post-proxy`** dans la section **`post-proxy`** : quand freeRADIUS recevra la réponse d'un *Home-Server*, il appliquera les règles de filtrage définies dans le fichier `/etc/freeradius/attrs`
- Ce fichier indique, pour chaque *realm*, quels attributs sont acceptables et avec quelles valeurs : les autres seront supprimés

Le fichier `attrs`

- Le fichier `attrs` est composé d'une suite de règles, un peu comme pour le fichier `users`, mais avec une logique différente, par exemple :

```
machin.com
        Reply-Message =* ANY,
        Session-Timeout <= 86400,
        Idle-Timeout <= 600,
        Acct-Interim-Interval >= 300,
        Acct-Interim-Interval <= 3600
...
```

/etc/freeradius/attrs

Tabulation obligatoire

Les opérateurs des conditions sont identiques à ceux des conditions du fichier `users` avec en plus l'opérateur `:=` (qui rajoute l'attribut ou le remplace s'il existe déjà)

- Une règle commence par le nom d'un *realm*, suivi à la ligne d'une liste de conditions sur des attributs (une condition par ligne)
- Le module **`attr_filter`** commence par chercher la règle correspondant au **`Realm`** du paquet, puis il supprime tous les attributs qui ne sont pas listés dans la règle, ainsi que tous les attributs pour lesquels aucune condition n'est satisfaite

Le fichier `attrs`

- On peut définir une règle **`DEFAULT`** (et une seulement) : elle est utilisée si aucun *realm* ne correspond
- On peut également rajouter **`Fall-Through = Yes`** à la fin d'une règle pour indiquer que l'on souhaite également appliquer les conditions de la règle **`DEFAULT`**

```
pas-cher-telecoms
    Filter-Id := "service-restreint",
    Fall-Through = Yes

DEFAULT
    Login-TCP-Port <= 65536,
    Framed-MTU >= 576,
    Filter-ID =* ANY,
    Reply-Message =* ANY,
    Proxy-State =* ANY,
    EAP-Message =* ANY,
    Service-Type == Framed-User,
    Service-Type == Login-User,
    ...
    Message-Authenticator =* ANY,
    State =* ANY,
    Session-Timeout <= 28800,
    Idle-Timeout <= 600,
    Port-Limit <= 2
```

Dans cet exemple, pour toutes les réponses des *Home-Servers* du *realm* **`pas-cher-telecoms`**, on rajoute l'attribut **`Filter-Id`** avec la valeur indiquée (si cet attribut est déjà présent, on remplace sa valeur), puis on applique le filtrage d'attributs par défaut

On peut indiquer plusieurs valeurs autorisées pour un attribut

/etc/freeradius/attrs

Filtrer les attributs

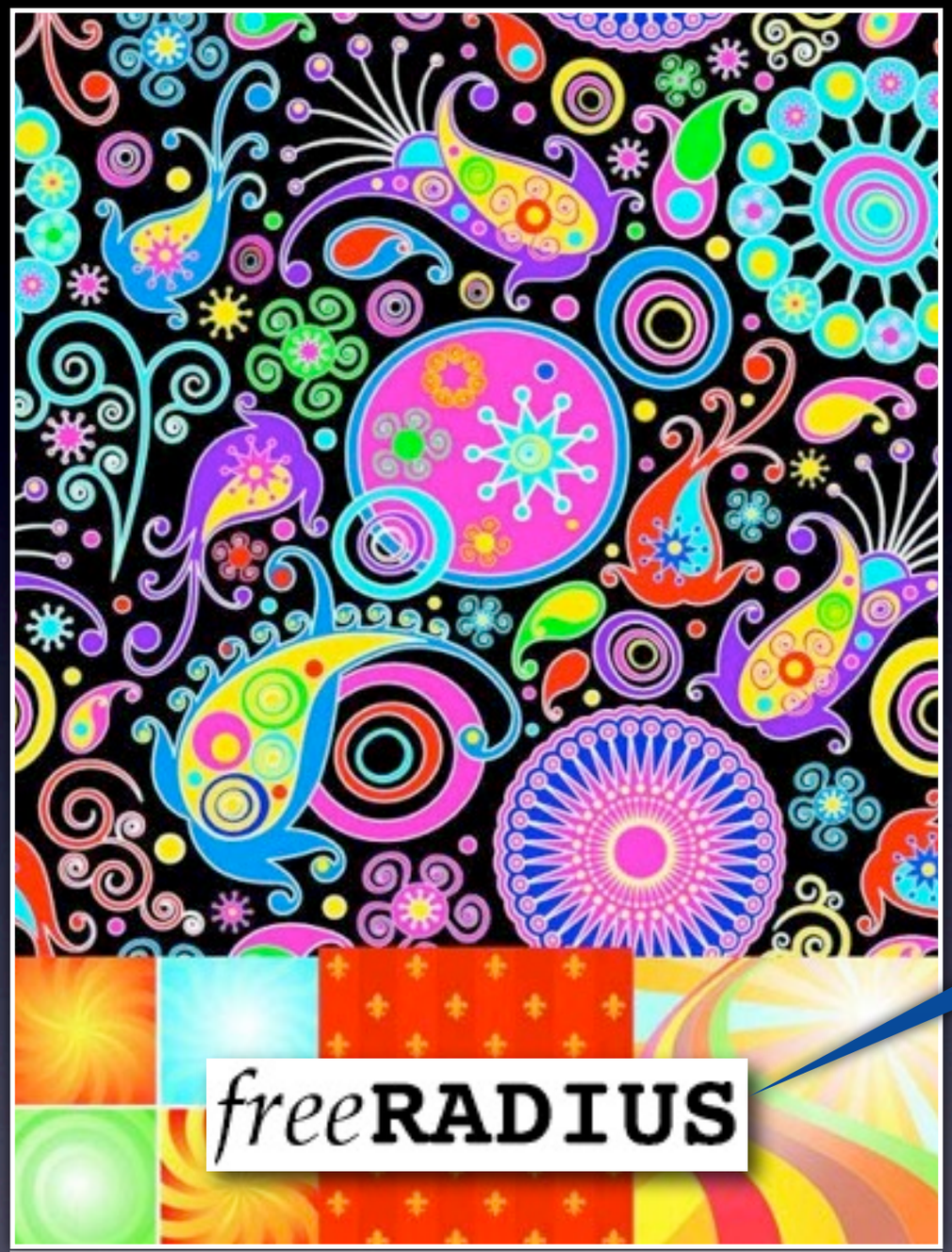
Le module `attr_filter` peut également, selon le même principe, être utilisé pour filtrer les attributs dans d'autres contextes :

- avant la redirection d'une requête vers un *Home-Server* (voir le fichier `attrs.pre-proxy`)
- ou même hors du contexte du *roaming* : par exemple, on peut filtrer les attributs utilisateur par utilisateur (plutôt que *realm* par *realm*) en ajoutant le paramètre `key = %{User-Name}` dans la configuration du module `attr_filter`

  ➡ dans le fichier de règles de filtrage correspondant, on précisera alors le nom d'un utilisateur au début d'une règle plutôt que le nom d'un *realm*

Ouf ! Vous savez tout sur la configuration de freeRADIUS !

Maintenant il reste à étudier quelques modules souvent utiles et voir comment en créer de nouveaux



Questions ?