

**UNIVERSITÉ DE MONS-HAINAUT**  
**FACULTÉ DES SCIENCES**

**Architecture AAA et protocole RADIUS**

Vince Stéphane

Centre Informatique Audio-visuel et Multimédia  
Directeur : Alain Buys  
Année académique 2004-2005

à Alphonse Vince

## Remerciements

---

A toute l'équipe de l'AWT qui m'a permis de réaliser ce travail, et plus particulièrement à mon cher collègue Carmelo Zaccone pour son soutien et sa disponibilité. Je tiens aussi à remercier André Delacharlerie et François Bodart pour m'avoir encouragé à entreprendre des études universitaires.

A Catherine, ma compagne et relectrice pour sa patience...

# Table des matières

---

## Partie Théorique

1	Introduction.....	5
2	Architecture AAA .....	6
2.1	Concepts de base.....	6
2.1.1	Authentification .....	6
2.1.2	Autorisation .....	6
2.1.3	Comptabilité .....	7
2.2	Le framework d'autorisation.....	7
2.2.1	Entités d'autorisations .....	7
2.2.2	Séquences de bases.....	8
2.2.3	Roaming.....	10
2.2.4	Services distribués.....	13
2.2.5	Droits d'accès (Policies) .....	14
2.2.6	Gestions des ressources et des sessions.....	15
3	RADIUS.....	16
3.1	Historique .....	16
3.2	Etude du fonctionnement .....	16
3.2.1	Généralités .....	16
3.2.2	Utilisation d'UDP .....	17
3.2.3	Format des paquets .....	18
3.2.4	Secret Partagé .....	19
3.2.5	Types de paquets.....	20
3.2.6	Méthodes d'authentification.....	22
3.2.7	Attributs, valeurs et dictionnaires.....	23
3.3	EAP.....	25
3.4	Problèmes de sécurité, attaques et contre-mesure.....	27
3.4.1	Les paquets de type <i>Access-Request</i> .....	27
3.4.2	Attribut <i>User-Password</i> .....	27
3.4.3	Secret partagé .....	28
3.5	Quelques mots sur Diameter.....	28
4	Autres méthodes d'identification .....	30
4.1	Kerberos.....	30
4.1.1	Description .....	30
4.1.2	Principes de fonctionnement.....	30
4.1.3	Cas d'utilisation .....	31
4.2	Infrastructure à clé publique.....	31
4.2.1	Description .....	31
4.2.2	Principes de fonctionnement.....	31
4.2.3	Cas d'utilisation .....	32
4.3	Biométrie.....	32
4.3.1	Description .....	32
4.3.2	Principes de fonctionnement.....	32
4.3.3	Cas d'utilisation .....	32

## Partie Pratique

5	Description et choix d'un serveur .....	34
5.1	Introduction .....	34
5.2	Généralités.....	34
5.3	Divers Serveurs .....	37
5.4	Plus en avant dans FreeRADIUS .....	41
5.4.1	Installation .....	41
5.4.2	Architecture et configuration de base.....	41
5.4.3	Gestion des types d'authentification .....	42
5.4.4	Gestion des Utilisateurs et de leurs droits.....	44
5.4.5	Gestion de l'Accounting .....	52
5.4.6	Configuration des serveur d'accès .....	53
5.4.7	Interface graphique d'administration .....	53
5.4.8	Tester sa configuration .....	54
6	Plan de travail .....	57
6.1	Environnement .....	57
6.2	Rationaliser la gestion des utilisateurs.....	58
6.3	Description de l'existant et plan d'évolution .....	58
6.3.1	Présentations des éléments .....	58
6.3.2	Description du réseau.....	59
6.3.3	Périphérie du serveur RADIUS .....	61
7	Mise en oeuvre.....	63
7.1	Configuration du serveur.....	63
7.1.1	Installation .....	63
7.1.2	Configuration générale : .....	63
7.1.3	Configuration du module LDAP.....	65
7.1.4	Configuration du module MySQL .....	65
7.1.5	Interaction avec les NAS .....	66
8	Conclusion.....	75
9	Annexes .....	76
10	Références.....	83

# PARTIE THÉORIQUE

# 1 Introduction

---

Les systèmes informatiques actuels ne cessent d'échanger des informations de toutes formes et en tous genres. Mais qu'en est-il de la véracité de ces informations ?

Dans un monde idéal, tout le monde pourrait se faire confiance et l'accès aux ressources ne devrait pas être restreint en fonction d'une identité. Dans un tel monde, personne n'essayerait d'avoir accès à des données qui ne le concernent pas, et il ne serait pas nécessaire de garder une trace des opérations qu'effectue une personne ou un système automatisé, car on ne commettrait pas d'erreurs et on ne tenterait pas d'effectuer des opérations préjudiciables à autrui.

Pourtant, la réalité est tout autre. Les tentatives d'intrusion dans les systèmes informatiques et les usurpations d'identités sont monnaie courante. La règle aujourd'hui est : « N'ayez confiance en personne ». C'est probablement pour cela que la sécurité informatique est devenue une des clés de voûte des systèmes actuels. En effet, il semble impensable, pour les entreprises usant (bon gré ou mal gré) de l'outil informatique, d'aujourd'hui négliger cet aspect des choses...

Deux des composantes de cette sécurité informatique sont la gestion des identités et la gestion des autorisations d'accès aux ressources. Il faut définir « qui peut faire quoi » et dans quelles limites. Ces deux assertions peuvent paraître simples, mais les techniques mises en œuvre pour les concrétiser sont aussi variées que complexes. Ce travail a pour but de présenter une de ces techniques de manière détaillée dans sa partie théorique et d'aborder un cas concret de l'exploitation de cette technique dans sa partie pratique.

## 2 Architecture AAA

---

### 2.1 Concepts de base

Constitué afin de standardiser les opérations d'accès aux fournisseurs de services Internet nécessitant une identification, le groupe de travail AAA a été créé au sein de l'Internet Engineering Task Force (IETF). Ce groupe, composé d'intervenants de divers horizons (universités, entreprises, ...), est à l'origine de plusieurs publications appelées RFC (Request For Comment) considérées comme des standards. Ces documents décrivent les différentes fonctions de base nécessaires aux fournisseurs de services, fonctions regroupées autour d'un même thème baptisé AAA.

Les fonctions principales sont au nombre de trois : l'authentification des utilisateurs, les autorisations qui leur sont accordées, et la comptabilité des opérations effectuées. En anglais, ces trois notions s'appellent *authentication*, *authorization* et *accounting* (d'où le choix du patronyme « AAA » donné à cette architecture).

#### 2.1.1 Authentification

L'authentification est le processus par lequel une entité prouve son identité. Plus simplement, il a pour but de répondre à la question : « Qui êtes-vous ? ».

Ce procédé, très courant en informatique, consiste souvent en l'introduction d'un nom d'utilisateur (ou login) et d'un mot de passe. Ce login est unique et identifie l'entité de manière non équivoque au sein d'un système. La connaissance du mot de passe associé à cet identifiant représente la preuve que l'on est qui l'on prétend être. La divulgation ou la découverte du mot de passe cassant ce schéma d'authentification...

Un des aspects clés de l'authentification est qu'elle établit une relation de confiance entre les deux entités. Ce point sera développé ci-après.

#### 2.1.2 Autorisation

Après avoir identifié avec succès un utilisateur, il convient de l'affecter correctement aux services auxquels il a préalablement souscrit. Plus largement, le concept d'autorisation a pour but de déterminer ce à quoi l'utilisateur a accès. Diverses règles seront donc appliquées souvent

en concordance avec des profils types d'utilisateurs permettant leur regroupement.

Par exemple, un abonné haut débit aura le droit de se connecter aussi via une connexion bas débit à son opérateur, tandis qu'un autre client n'ayant pas ce type d'abonnement sera restreint à un accès bas débit.

### 2.1.3 Comptabilité

Pour cette dernière notion, on s'intéresse à ce que fait l'utilisateur des ressources auxquelles il a accès. Le choix de baptiser cette fonction « comptabilité » n'est pas un hasard, les opérateurs de services Internet facturant fréquemment leurs utilisateurs en fonction de ce qu'ils font.

La comptabilité permet de déterminer entre autres :

- à quel moment un utilisateur s'est connecté,
- la raison d'une déconnexion ou d'un refus de connexion,
- le temps de connexion,
- le type de connexion employé,
- le volume de données transféré.

Le traitement de ces données n'est bien sûr pas limité aux seuls aspects de facturation. Les opérations statistiques que ces données permettent d'effectuer sont nombreuses.

## 2.2 Le framework d'autorisation

Défini par le RFC 2904, ce cadre de travail représente l'une des pièces maîtresses de l'architecture AAA. Il définit le comportement des diverses entités intervenant lors d'une procédure de connexion.

### 2.2.1 Entités d'autorisations

On peut définir quatre types d'entités qui interagissent lors d'une demande de connexion à un FAI<sup>1</sup> :

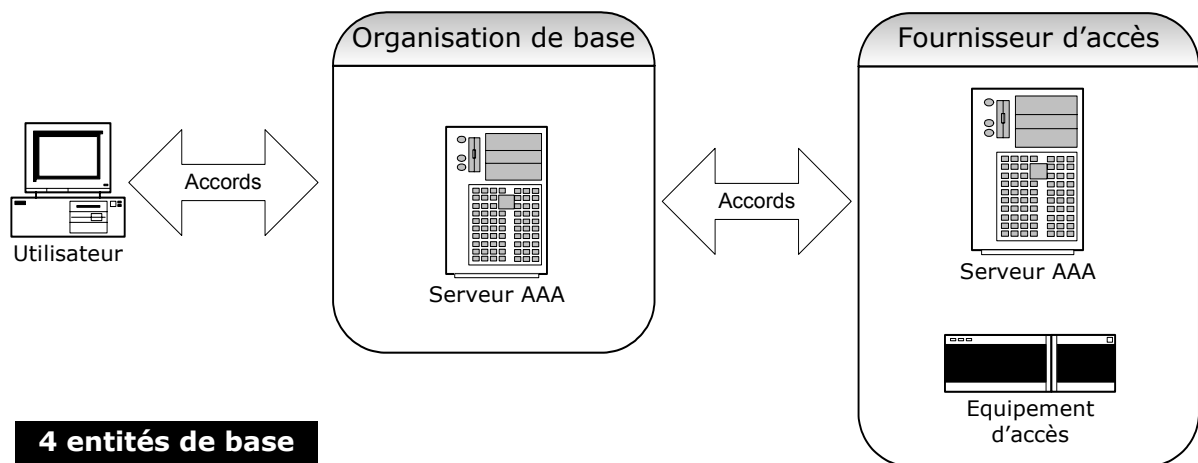
- L'utilisateur qui souhaite accéder à une ressource.
- L'organisation de base de cet utilisateur<sup>2</sup> : c'est l'organisation avec laquelle l'utilisateur a un accord (par ex : un contrat). C'est

---

<sup>1</sup> FAI : Fournisseur d'Accès à Internet

cette entité qui déterminera les autorisations accordées à l'utilisateur (par ex : une limite d'utilisation quotidienne).

- Le serveur AAA du fournisseur d'accès : c'est lui qui procède au provisionnement de l'équipement d'accès. Il est en relation avec le serveur AAA de l'organisation de base. Il ne connaît rien de l'utilisateur, il ne valide pas son accès.
- L'équipement réseau du fournisseur d'accès : c'est le matériel qui fournit le service à proprement parler. Souvent appelé NAS (Network Access Server – Serveur d'accès réseau). Il peut s'agir d'une batterie de modem, d'un routeur, d'un pare-feu, ...



#### 4 entités de base

Trois acteurs établissent ainsi deux à deux des « accords » bilatéraux. Ces accords sont fréquemment de deux natures. D'une part contractuelle (au sens formel du terme), définissant par exemple les tenants et aboutissants financiers. D'autre part relationnelle, établissant une relation de confiance (au sens informatique) entre chacun de ses acteurs.

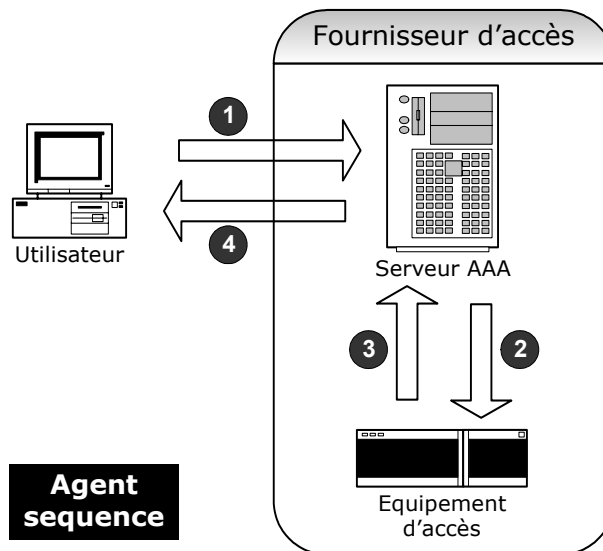
### 2.2.2 Séquences de bases

Afin de simplifier les choses, nous allons tout d'abord considérer que l'organisation de base est le fournisseur d'accès, réduisant ainsi le nombre d'acteurs à deux. Au point suivant, nous nous replacerons dans un contexte à plusieurs acteurs.

---

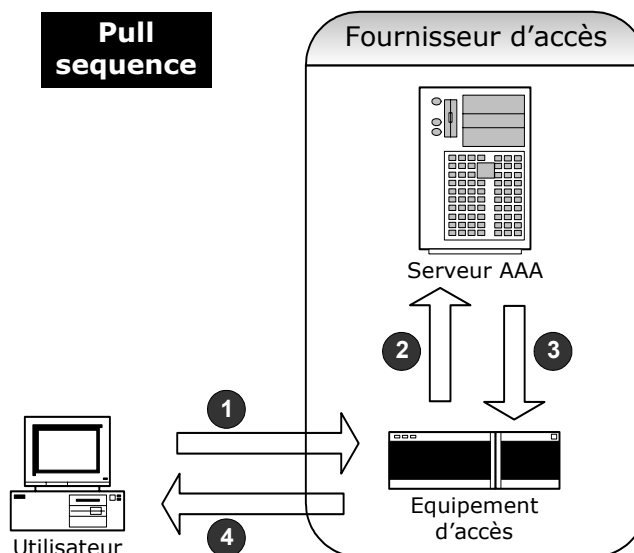
<sup>2</sup> La littérature étant uniquement anglophone, il s'agit de la traduction de ce que le RFC définit comme UHO (User Home Organization).

### 2.2.2.1 Séquence agent (*Agent sequence*)



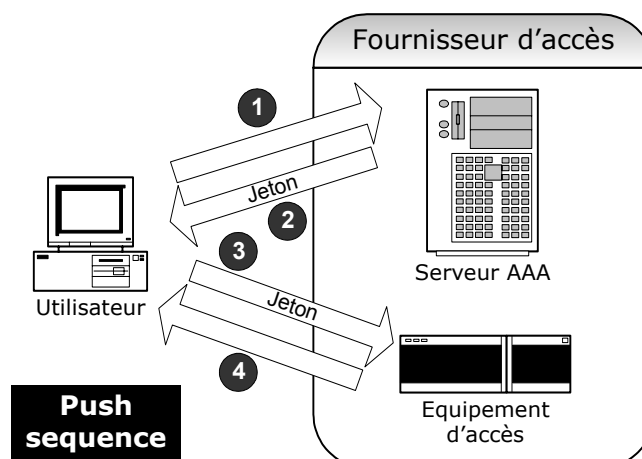
Dans cette séquence, le serveur AAA joue le rôle d'intermédiaire (d'agent) entre l'utilisateur et l'équipement d'accès. L'utilisateur contacte d'abord le serveur qui va à son tour contacter le NAS, ce dernier effectuant la liaison vers l'utilisateur en informant le serveur au passage.

### 2.2.2.2 Séquence tirée (*Pull sequence*)



Ici, ce n'est plus le serveur qui est contacté, mais l'équipement d'accès. Celui-ci contacte ensuite le serveur AAA qui, après validation, commande au NAS de rendre la connexion effective. C'est cette séquence qui est employée lors d'un accès réseau distant.

### 2.2.2.3 Séquence poussée (*Push sequence*)



Dans cette dernière possibilité, c'est le client qui joue le rôle d'intermédiaire. Un 1<sup>er</sup> contact est établi avec le serveur AAA qui valide l'accès de l'utilisateur en retournant un jeton<sup>3</sup>. Ce dernier est ensuite utilisé pour valider l'accès auprès de l'équipement réseau.

Il est intéressant de noter que cette séquence implique une relation de confiance entre chaque intervenant. Premièrement entre l'utilisateur et le serveur AAA, deuxièmement entre ce même serveur et le NAS, et troisièmement entre ce NAS et l'utilisateur de départ.

### 2.2.3 Roaming

Un des points principaux de l'architecture AAA est la capacité des serveurs à pouvoir se relayer dans les diverses requêtes (RFC 2903). Cette opération de relais (*proxy* en anglais) permet une très grande souplesse de déploiement.

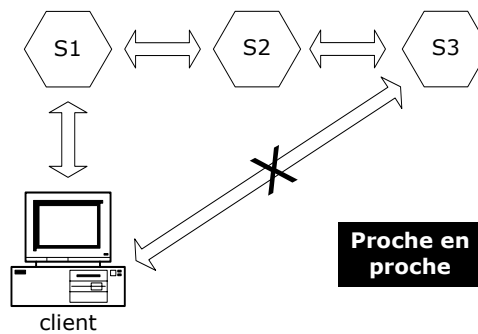
Le *proxying* permet de relayer des informations entre plusieurs entités<sup>4</sup> (voir schéma p 12). A un bout de la chaîne, un client transmet une information à un serveur. Le serveur ayant reçu l'information devient alors client d'un autre serveur et lui transmet l'information précédemment reçue. Cette méthode peut être reproduite le nombre de fois nécessaire à atteindre le serveur de destination (capable de traiter l'information).

<sup>3</sup> Ce jeton peut, par exemple, être un ticket comme dans Kerberos, ou un certificat.

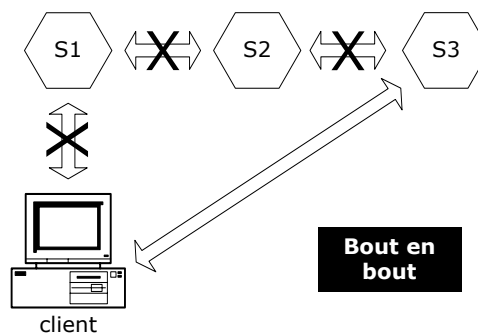
<sup>4</sup> Avec : nombre d'entités  $\geq 3$ .

Avec cette méthode, les relations de confiance deviennent problématiques. Deux types de transactions peuvent être envisagées :

- De proche en proche : dans cette configuration, chaque intervenant dans la transaction possède une relation de confiance avec son voisin immédiat. Les relations de confiance n'étant pas transitives, le client d'origine et le serveur en bout de chaîne ne peuvent pas prétendre à une relation de confiance.



- De bout en bout : cette vision différente des choses part du principe qu'une relation de confiance existe entre le serveur à contacter et le client. A contrario, il n'y a pas dans ce modèle de relation de confiance entre le client et son voisin immédiat.

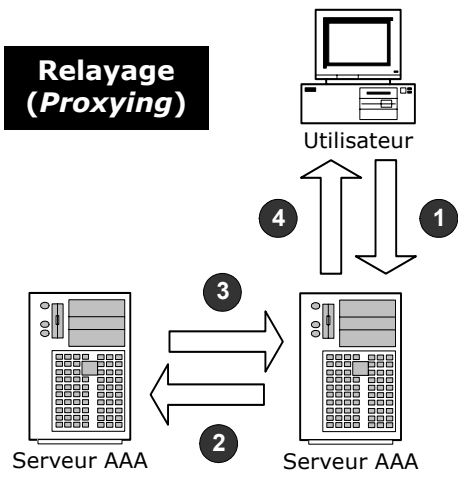


Le proxying est utilisé dans les différentes techniques de roaming ; néanmoins, le framework d'autorisation n'impose pas le type de transaction à effectuer, qui est laissé à la discrétion des implémentations du modèle.

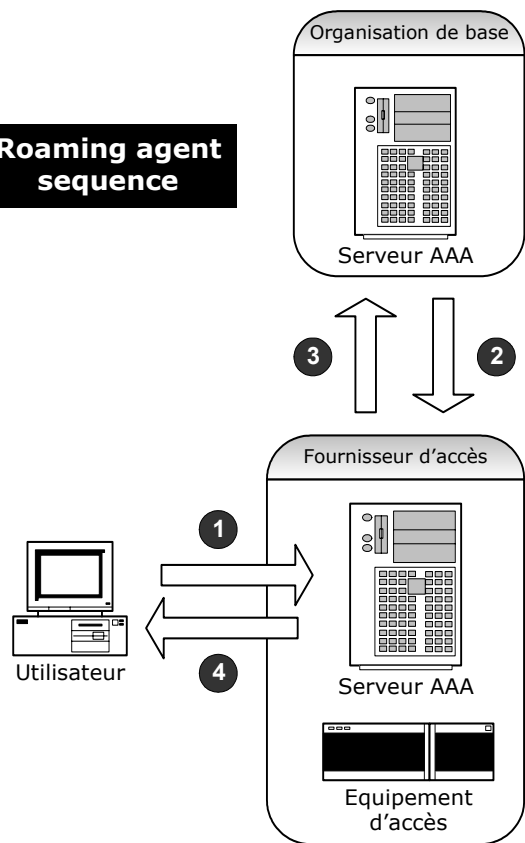
C'est donc afin d'ajouter un niveau supplémentaire d'intégration entre divers acteurs que sont employées les opérations de roaming. Elles permettent, par exemple, à un opérateur important de louer une partie de ses équipements à un plus petit fournisseur d'accès. L'organisation qui fera autorité pour l'utilisateur ne sera donc plus située à la périphérie immédiate du NAS.

Les mêmes séquences agents, tirées et poussées sont disponibles en roaming, elles sont présentées ci-après :

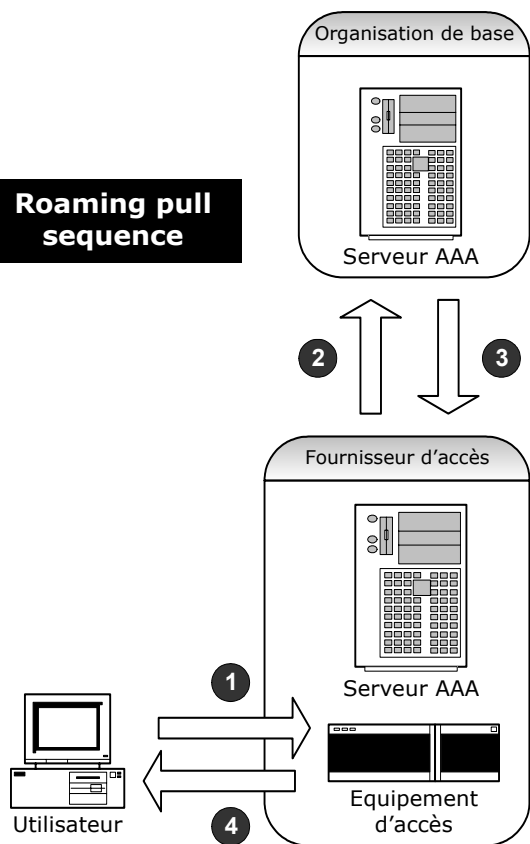
**Relayage (Proxying)**



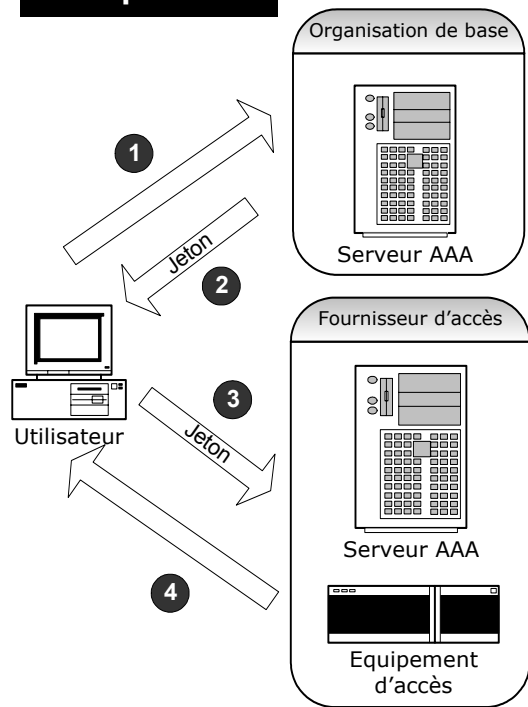
**Roaming agent sequence**



**Roaming pull sequence**



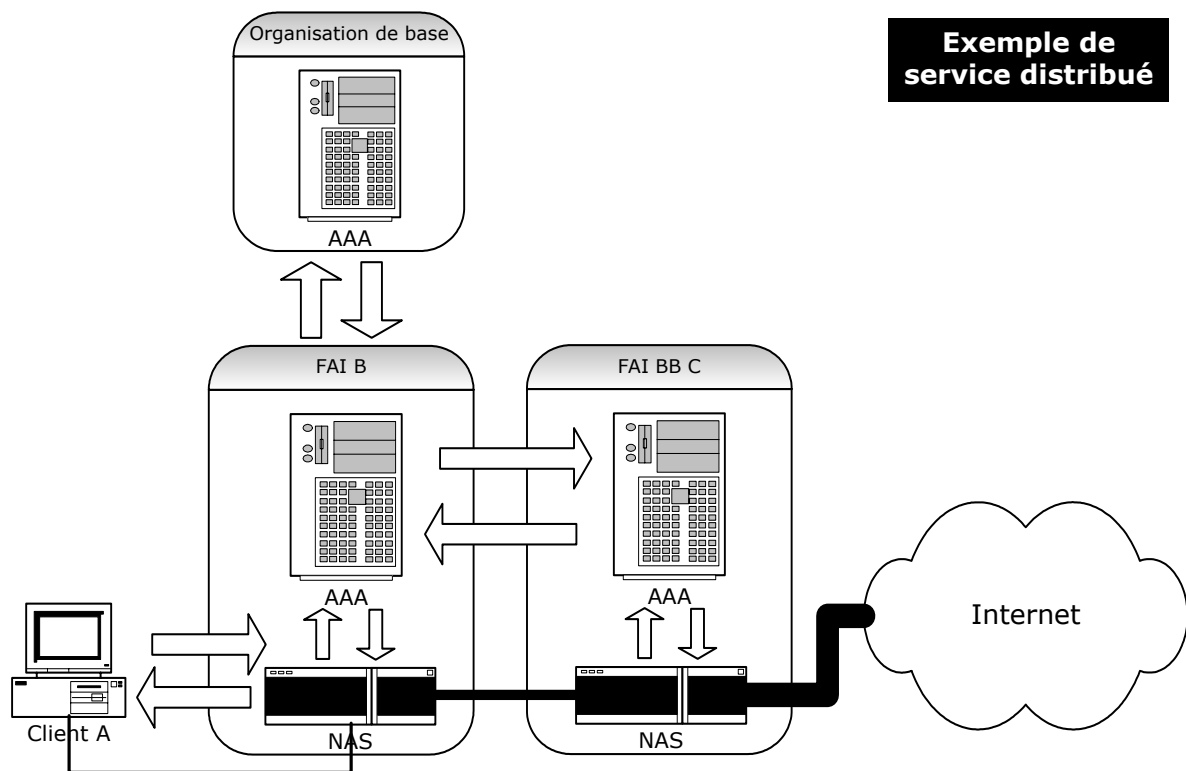
**Roaming push sequence**



## 2.2.4 Services distribués

Ce dernier principe va combiner les séquences de roaming entre elles afin de pouvoir réaliser des opérations plus complexes impliquant un nombre important de fournisseurs de services.

Par exemple : un client A a conclu un contrat avec un fournisseur d'accès B lui garantissant une bande passante minimum d'un mégabit/s vers internet. Le serveur AAA du fournisseur B va donc configurer ses équipements de façon à garantir le méga. Mais, cette bande passante est garantie vers Internet. Le serveur AAA du fournisseur B va donc contacter son homologue chez le fournisseur de backbone C afin qu'il configure de son côté une politique de qualité de service permettant d'honorer le contrat que le fournisseur B a conclu avec le client A.



Une telle implémentation sera possible en combinant par exemple une séquence roaming pull entre A et B suivie d'une séquence agent entre B et C.

## 2.2.5 Droits d'accès (Policies)

Les droits d'accès définissent le comportement d'un serveur en regard d'une demande qui lui est adressée. Tous les serveurs implémentant le protocole AAA sont capables de gérer des droits d'accès ne fût-ce que pour accepter ou rejeter les utilisateurs.

Le framework édicte la loi suivante : « les droits d'accès sont récupérables, évaluables et applicables ». La terminologie définit diverses notions : les droits sont récupérés à un Policy Retrieval Point (PRP), ils sont évalués par un Policy Decision Point (PDP) et rendus effectifs via un Policy Enforcement Point (PEP).

Typiquement, un serveur AAA assume les rôles de PRP et de PDP. Néanmoins, le rôle PRP peut être décentralisé (par exemple en utilisant une base LDAP<sup>5</sup>). Les PEP sont en général représentés par les équipements terminaux.

Les droits doivent, dans certaines circonstances, être échangés (voir le point précédent). A cette fin, le framework définit deux dernières notions que sont les Policy Information Point (PIP) et les Policy Information Block (PIB). Ces notions garantissent la standardisation de l'échange des droits.

---

<sup>5</sup> Plus d'infos sur ce sujet à la page 48

## 2.2.6 Gestions des ressources et des sessions

La gestion des ressources est la capacité qu'a un serveur AAA de connaître les disponibilités des divers équipements qu'il pilote. Un composant du serveur appelé « *resource manager* », doit être capable de recevoir et de restituer les informations transmises en temps réel.

Reprenons l'exemple cité au point 2.2.4, et fixons la capacité du lien entre le fournisseur B et fournisseur C à 10 mégabit/s. Imaginons à présent une nouvelle entité D qui permet d'accéder à Internet au même titre que C. Le lien B<->D ayant une capacité variable, B payant ce lien en fonction de son utilisation.

En heure de pointe, plusieurs clients du même type que A viennent saturer la connexion B <-> C, le serveur AAA de B informé de la situation, modifiera les routes des nouvelles connexions afin qu'elles passent par D plutôt que par C.

La gestion des sessions est l'aptitude d'un équipement terminal à communiquer l'état de ses périphériques à un serveur AAA et d'éventuellement modifier les sessions en cours.

La gestion des sessions peut être représentée par un exemple simple. Un FAI ne souhaite généralement pas voir un de ses abonnés utiliser deux connexions simultanément. Pour que cela soit possible, il faut que les différents NAS auxquels l'utilisateur a accès communiquent au serveur AAA quand une session avec l'utilisateur est en cours. Ainsi, le serveur AAA refusera une deuxième tentative de connexion de ce même utilisateur puisqu'il est au courant qu'une session est déjà en cours.

Les principes de gestion de ressources et de sessions sont directement liés. Le serveur AAA devant être informé en permanence de l'affectation ou de la libération de ressources, afin de pouvoir créer, refuser ou modifier des sessions.

## 3 RADIUS

---

### 3.1 Historique

RADIUS est l'abréviation de Remote Access DialIn User Service. Bien que sa création et sa standardisation soient antérieures aux travaux du working group AAA, les similitudes sont remarquables.

RADIUS fut développé à l'origine par Livingston Entreprises, à la demande de l'opérateur californien Merit Networks qui souhaitait une meilleure intégration de la gestion de ses utilisateurs. En effet, avant cela chaque NAS possédait une liste des utilisateurs et des scripts de synchronisation étaient employés pour répliquer ces mêmes listes. Le nombre d'utilisateurs allant croissant, les employés de Merit firent la proposition d'une standardisation sur la gestion des utilisateurs des systèmes d'accès distants. Livingston fut le premier à répondre à cet appel et implémenta ainsi le premier serveur RADIUS. Les versions se succédant, elles contribuèrent à la standardisation du protocole. Aujourd'hui Livingston a été racheté par Lucent. Ce dernier propose toujours une version gratuite des sources du serveur.

### 3.2 Etude du fonctionnement

#### 3.2.1 Généralités

RADIUS est défini principalement dans le RFC 2138. Le standard possède plusieurs caractéristiques :

##### **3.2.1.1 Architecture de type client/serveur**

Le client typique est un NAS. Il est chargé de transmettre les requêtes de connexion des utilisateurs vers le serveur RADIUS et d'ensuite interpréter la réponse qu'il lui est donnée.

Quant au serveur, il est chargé de recevoir les requêtes des utilisateurs, d'identifier ces utilisateurs et de finalement renvoyer les informations nécessaires au NAS pour qu'il puisse établir les connexions demandées par ces utilisateurs.

Il est également important de noter que, lorsqu'une technique de relais (proxy) est employée, le serveur RADIUS devient client d'un autre serveur AAA.

### **3.2.1.2 Sécurité des messages**

Les messages transmis entre un client et un serveur RADIUS sont authentifiés grâce à l'usage d'un secret partagé qui ne sera jamais transmis sur le réseau. De plus, chaque mot de passe envoyé sur le réseau doit être chiffré de manière à ce que la capture d'un paquet n'en permette pas la découverte.

### **3.2.1.3 Méthode d'authentification flexible**

Un serveur RADIUS supporte diverses méthodes pour identifier un utilisateur. L'une de ces méthodes est l'envoi d'un login et d'un mot de passe. Dans ce cas, les protocoles PPP PAP et PPP CHAP doivent être implémentés. Le serveur n'est pas restreint à ces protocoles, d'autres méthodes peuvent être ajoutées.

### **3.2.1.4 Protocole extensible**

Tous les échanges clients – serveur sont basés sur l'emploi de tuples Attribut – longueur – valeur. De nouveaux attributs peuvent être définis sans perturber les implémentations existantes du protocole.

## **3.2.2 Utilisation d'UDP**

RADIUS utilise le protocole UDP<sup>6</sup> pour transmettre ses messages. La justification de ce choix est expliquée en quatre points dans le RFC, l'idée générale étant que le comportement de RADIUS implique le choix de UDP :

- Si une requête n'aboutit pas au près du serveur primaire, elle doit être renvoyée à un serveur secondaire<sup>7</sup>. Pour ce faire, il faut donc conserver une copie du paquet original avant la couche de transport. L'utilisation d'un temporisateur permettant de déduire que le serveur primaire est inaccessible.
- Les exigences en terme de temps de réponse de TCP<sup>8</sup> ne sont pas adaptée à RADIUS. Celui-ci se basant sur le temps moyen écoulé avant d'effectuer une retransmission paraît trop « agressif » (sic) aux concepteurs du protocole. Ces derniers considèrent que l'utilisateur peut attendre les quelques secondes durant la phase d'identification.
- Par essence, le protocole RADIUS ne maintient pas d'état. En clair, il ne garde pas de trace de ses précédentes actions. UDP non plus n'utilise pas d'état. Cette propriété commune simplifie

---

<sup>6</sup> UDP : User Datagram Protocol

<sup>7</sup> Ce comportement tout à fait similaire à celui du service DNS

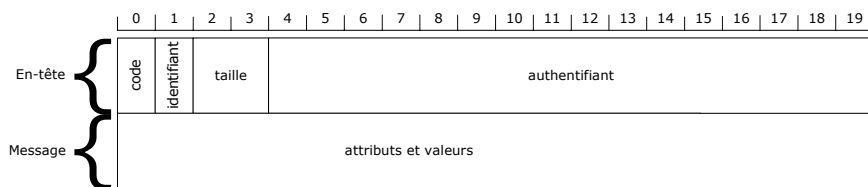
<sup>8</sup> TCP : Transmission Control Protocol

les problèmes de pertes de connexion ou de congestion du réseau qui peuvent devenir problématiques lorsqu'on utilise un protocole tel que TCP.

- Globalement, l'utilisation de UDP simplifie l'implémentation du serveur. La seule complexité avec UDP réside dans la gestion des retransmissions. Elles devront être gérées par l'application elle-même, la couche transport ne les prenant pas en charge.

### 3.2.3 Format des paquets

Un paquet RADIUS est inclus dans un et un seul paquet UDP. Le schéma suivant représente un paquet RADIUS standard, les unités étant exprimées en octets :



**Le code** (1 octet) : permet de distinguer le type de paquet auquel on a affaire. Les codes valides sont :

- 1 : Access-Request
- 2 : Access-Accept
- 3 : Access-Reject
- 4 : Accounting-Request
- 5 : Accounting-Response
- 11 : Access-Challenge
- 12 : Status-Server (experimental)
- 13 : Status-Client (experimental)
- 255 : Réserve

Si le code n'est pas une des 9 valeurs ci-dessus, le paquet est ignoré et aucune réponse n'est renvoyée au destinataire. Le point 3.2.5 couvre les différents types de paquets.

**L'identifiant** (1 octet) : permet de reconnaître une requête RADIUS déterminée. L'identifiant est utilisé pour numéroter une question (request). La réponse à cette question portera le même numéro. Ce champ est très utile par exemple lors de la perte de paquet.

**Taille** (2 octets) : cette valeur comprise entre 20 et 4096 octets représente la longueur de tout le paquet<sup>9</sup>. Si la taille est trop grande par

<sup>9</sup> taille = 1 + 1 + 2 + 16 +  $\sum$  length(Paire A/V)

rapport au nombre de données envoyées, le protocole préconise d'ignorer le paquet. A l'inverse, en cas d'une trop petite taille, il faut tenir compte des données qui se trouvent en deçà du point d'arrêt fixé par la taille.

**Authentifiant** (16 octets) : ce champ est utilisé de façon différente en fonction du type de requête à laquelle on a affaire. Le RFC distingue deux types d'authentifiants : les authentifiants de requêtes (*Request Authenticator*) et les authentifiants de réponses (*Response Authenticator*).

*Authentifiant de requête* : Dans un paquet de type `Access-Request`, il s'agit d'un nombre totalement aléatoire<sup>10</sup> prévu pour contrecarrer les attaques. RADIUS ne propose pas de méthode empêchant l'écoute ou la capture de paquet. Cette technique assure néanmoins qu'un paquet déjà transmis ne pourra plus être employé.

*Authentifiant de réponse* : Dans un paquet de type `Access-Accept`, `Access-Challenge` ou `Access-Reject`, le champ authentifiant contient le résultat d'une fonction de hachage<sup>11</sup> basée notamment sur l'authentifiant de requête et sur le secret partagé.

### 3.2.4 Secret Partagé

Afin de renforcer la sécurité et garantir l'intégrité des transactions, le protocole utilise un secret. Celui-ci est partagé entre un client et un serveur. Il est recommandé que chaque client possède son propre secret et la taille de celui-ci soit d'au minimum 16 octets.

Le secret (partagé) est utilisé dans le calcul de l'authentifiant et ce quand il s'agit d'un authentifiant de réponse (voir ci-dessus). Le secret est également utilisé pour chiffrer l'attribut `User-Password` (voir le point 3.4.2 à ce sujet).

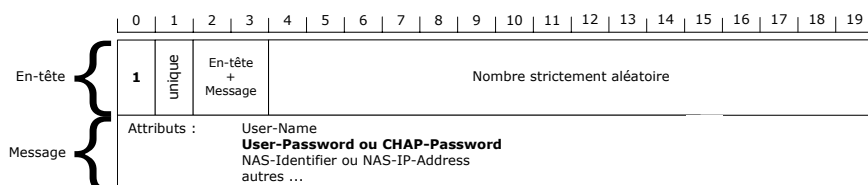
---

<sup>10</sup> un opposant ne doit pas pouvoir « deviner » le nombre suivant.

<sup>11</sup> Authentifiant = MD5(Code+ID+Length+RequestAuth+Attributes+Secret)

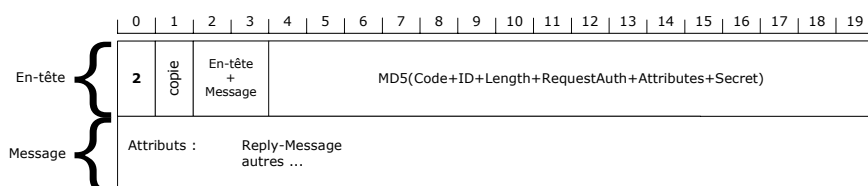
## 3.2.5 Types de paquets

### 3.2.5.1 Paquet Access-Request



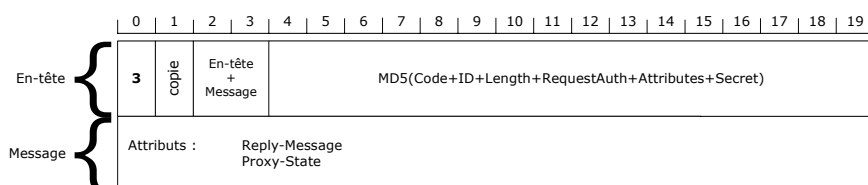
Les paquets `Access-Request` sont utilisés pour effectuer une demande d'authentification auprès d'un serveur RADIUS. Les attributs `User-Password` ou `CHAP-Password` doivent y figurer.

### 3.2.5.2 Paquet Access-Accept



Les paquets `Access-Accept` sont renvoyés par le serveur pour signifier que la demande d'authentification a été acceptée. L'identifiant utilisé dans ce paquet est le même que celui transmis au serveur lors de l'`Access-Request`. L'authentifiant est le résultat d'une fonction de hachage MD5. Aucun attribut n'est réellement requis, mais il est clair que le serveur doit en retourner s'il gère les autorisations. Des attributs permettant la configuration du lien réseau sont donc souvent joints à ce paquet de réponse.

### 3.2.5.3 Paquet Access-Reject



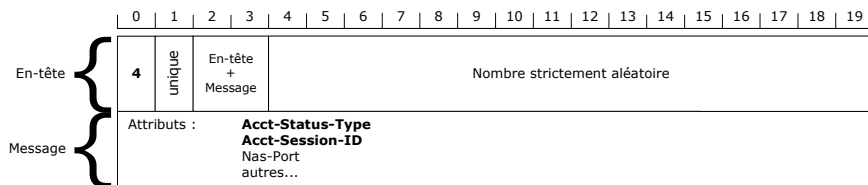
Les paquets `Access-Reject` sont renvoyés par le serveur pour signifier que la demande d'authentification a échoué. L'identifiant utilisé dans ce paquet est le même que celui transmis au serveur lors de l'

Access-Request. L'authentifiant est le résultat d'une fonction de hachage MD5. Les seuls attributs tolérés sont le Reply-Message (qui peut apparaître plusieurs fois) et le Proxy-State qui est employé par les serveurs lors des opérations de proxying.

Un paquet de ce type peut être théoriquement envoyé à n'importe quel moment afin, par exemple, de forcer l'arrêt d'une session.

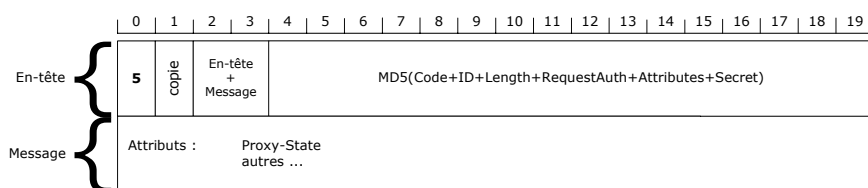
### 3.2.5.4 Paquet Accounting-Request

Les requêtes d'accounting sont définies dans le RFC 2139. Elles permettent au serveur de connaître l'état d'avancement d'une session. Leur format est en tout point similaire au requête de type « access ».



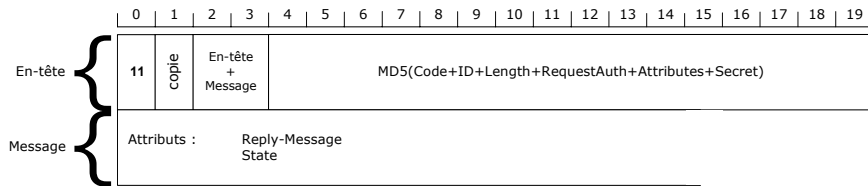
Les paquets Accounting-Request sont utilisés pour effectuer une demande d'accounting auprès d'un serveur RADIUS. Les attributs Acct-Session-ID et Acct-Status-Type doivent y figurer. Les demandes d'accounting sont caractérisées par ce dernier attribut. Elle sont généralement de type *start*, *stop* ou *update*. Les deux derniers types de requêtes sont, en principe, accompagnés d'informations sur la session en cours (durée de la session, nombre d'octets transférés, ...).

### 3.2.5.5 Paquet Accounting-Response



Les paquets Accounting-Response sont renvoyés par le serveur pour signifier que la demande d'accounting a été acceptée. L'identifiant utilisé dans paquet est le même que celui transmis au serveur lors de l'Accounting-Request. L'authentifiant est le résultat d'une fonction de hachage MD5. Aucun attribut n'est réellement requis étant donné qu'il s'agit ici d'un pure accusé de réception.

### 3.2.5.6 Paquet Access-Challenge



Les paquets Access-Challenge sont envoyés au NAS lorsque qu'une tentative d'authentification paraît suspecte au serveur. Certains NAS ne supportent pas ce type de requête et l'interprètent comme s'il s'agissait d'un Access-Reject.

## 3.2.6 Méthodes d'authentification

Le RFC prévoit deux méthodes d'authentification des utilisateurs nommées PAP (*Password Authentication Protocol*) et CHAP (*Challenge-Handshake Authentication Protocol*). Ces méthodes ne sont pas les seules, d'autres peuvent être employées : c'est notamment le cas d'EAP (voir le point 3.3 à ce sujet).

### 3.2.6.1 PAP

L'utilisation de PAP est suggérée au serveur RADIUS lorsque l'attribut `User-Password` est trouvé dans un paquet `Access-Request`. Dans ce cas, le mot de passe est alors transmis en clair du client au NAS (en principe sur une liaison point à point). Le NAS transmet ce mot de passe au serveur RADIUS par l'intermédiaire de l'attribut `User-Password` après l'avoir dûment chiffré (voir les points 3.2.4 et 3.4.2 à ce sujet).

### 3.2.6.2 CHAP

L'utilisation de CHAP est suggérée au serveur RADIUS lorsque l'attribut `CHAP-Password` est trouvé dans un paquet `Access-Request`. CHAP est basé sur l'idée de transmettre une preuve de la connaissance du mot de passe mais pas ce dernier à proprement parler.

Le NAS génère un flux de bytes aléatoire de 16 octets et le transmet au client. Le client utilise ce nombre dans une fonction de hachage MD5 avec son mot de passe comme clé. Le client retourne le login, le haché (*CHAP Response*) et un identifiant de transaction (*CHAP ID*) au NAS. Ce dernier transfère alors :

- le login via l'attribut `User-Name`,
- le *CHAP ID* suivi de *CHAP Response* via l'attribut `CHAP-Password` et

- le challenge original (l'aléatoire de 16 octets) via l'attribut CHAP-Challenge.

Le serveur effectue la même opération que le client et compare le résultat obtenu à *CHAP Response*.

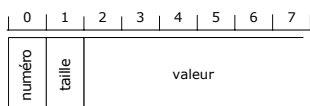
Si CHAP a l'avantage de ne pas faire transiter le mot de passe sur le réseau, il implique la lecture de celui-ci côté serveur, ce qui n'est pas toujours une situation enviable.

### 3.2.7 Attributs, valeurs et dictionnaires

Les attributs sont l'essence même du protocole. C'est eux qui véhiculent les informations nécessaires aux NAS pour qu'ils puissent assurer les connexions. Le RFC 2138 définit pas moins de 41 attributs utilisés pour les opérations d'authentifications et d'autorisations. Les 15 attributs nécessaires à la comptabilisation sont eux définis dans le RFC 2139.

Les attributs ne peuvent être employés de manière anarchique. Le standard interdit l'emploi de certains attributs dans des types de paquets bien définis. Un exemple trivial est celui de l'attribut *User-Password* qui ne peut-être employé que dans un paquet de type *Access-Request*.

Un attribut est caractérisé par 3 champs représentés graphiquement sur ce schéma :



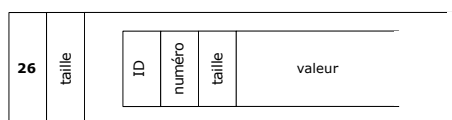
- **Le numéro** (1 octet) : compris entre 1 et 255, le numéro identifie l'information qui caractérise l'attribut. Par exemple : 1 = *User-Name*.
- **La taille** (1 octet) : la taille est calculée en fonction du contenu de l'attribut et doit être supérieure à 3.
- **La valeur** (taille variable) : c'est dans ce champ qu'est contenu l'information véhiculée par l'attribut.

La littérature définit le terme de AVP (*Attribute Value Pair*). En français cette abréviation est parfois traduite en paire A/V (pour paire d'attribut et de valeur). En effet, un attribut est similaire à une variable informatique. Une variable étant caractérisée par son nom, son type et sa valeur, il en va de même pour un attribut.

Les types d'attributs sont bien entendu standardisés et peuvent être :

Symbole	Description
INT	Il s'agit d'un entier sur 32 bits.
ENUM	Type énuméré. Il s'agit également d'un entier sur 32 bits, mais qui prend cette fois une valeur déterminée ayant une signification particulière.
IPADDR	Adresse IP stockée sur 32 bits (pour IPv4). IPv6 est également supporté et l'encodage est alors laissé à la discrétion de l'implantation.
STRING	Chaîne de caractères d'un maximum de 253 octets encodée au format UTF-8.
DATE	La date stockée grâce à un entier de 32 bits non signé représentant le nombre de secondes écoulées depuis le 1 <sup>er</sup> janvier 1970.
BINARY	Une valeur binaire.

L'attribut numéro 26 est très particulier. Baptisé *Vendor-Specific*, il permet aux équipementiers de définir leurs propres paires A/V. Chaque vendeur s'est vu attribuer un identifiant spécifique (sur 1 octets). Les paires A/V spécifiques au vendeur sont encapsulées dans l'attribut 26 comme représenté dans le schéma ci-dessous :



La plupart des implémentations RADIUS utilisent un dictionnaire pour stocker leurs informations. Un dictionnaire peut par exemple être concrétisé dans un simple fichier. L'annexe 3 montre un exemple de ce à quoi peut ressembler un dictionnaire. L'emploi d'un dictionnaire est surtout très avantageux pour définir ses propres attributs (dans le cas de l'utilisation d'attribut *Vendor-Specific* évidemment).

Une liste exhaustive des paires A/V et de leur utilisation n'a pas sa place dans ce travail, les attributs étant clairement définis dans les documents de standardisation.

### 3.3 EAP

Défini par le RFC 2284, EAP est l'abréviation de *Extensible Authentication Protocol*. Ce protocole n'est pas un composant requis dans un serveur RADIUS mais est inclus dans de nombreuses implémentations. EAP est une extension du protocole PPP<sup>12</sup>. L'abréviation EAP seule est communément admise, bien que les RFC parlent logiquement de PPP-EAP.

PPP est utilisé afin d'établir une communication point à point. Il prévoit trois phases consécutives. Tout d'abord, l'établissement d'un lien entre les deux intervenants à l'aide du protocole LCP (*Link Control Protocol*). Ensuite, ce lien étant établi, une phase optionnelle d'identification peut avoir lieu. Pour finir, ce sont les protocoles NCP (*Network Control Protocols*) qui sont utilisés pour configurer le protocole réseau employé durant toute la durée de la communication.

C'est au niveau de la deuxième phase de PPP que EAP entre en jeu. Au lieu d'imposer une phase d'identification, ce protocole entame d'abord une négociation sur la méthode d'authentification qui va ensuite être employée. Après que la négociation a abouti, la méthode retenue est appliquée. L'intérêt de cette façon de travailler réside dans le fait de supporter virtuellement n'importe quelles méthodes d'authentifications (d'où l'emploi du terme *extensible*). Ces dernières peuvent alors être vues comme de simples plugins (ou librairies). Ces librairies seront installées aussi bien sur le client PPP que sur le serveur PPP.

Les NAS<sup>13</sup> et les serveurs RADIUS modernes supportent cette extension<sup>14</sup>. Pour ce qui est du NAS, lors d'une communication EAP, il agit comme une passerelle entre le client et le serveur RADIUS (voir le schéma ci-après). Ces deux dernières entités doivent donc avoir au moins une méthode d'authentification en commun.

Les méthodes pouvant intervenir lors d'une authentification utilisant EAP sont diverses. Citons en quelques-unes :

- EAP-MD5 est similaire à CHAP.
- EAP-GTC (GTC : *Generic Token Card*) : cette méthode utilise une carte sur laquelle est noté un simple texte représentant un jeton (*token*).
- EAP-OTP (OTP : *One Time Password*) : cette méthode est basée sur l'emploi de fonctions de hachage de manière à ce que un hash (sur base d'un mot de passe, par exemple) ne soit utilisé que lors d'une et d'une seule transaction.

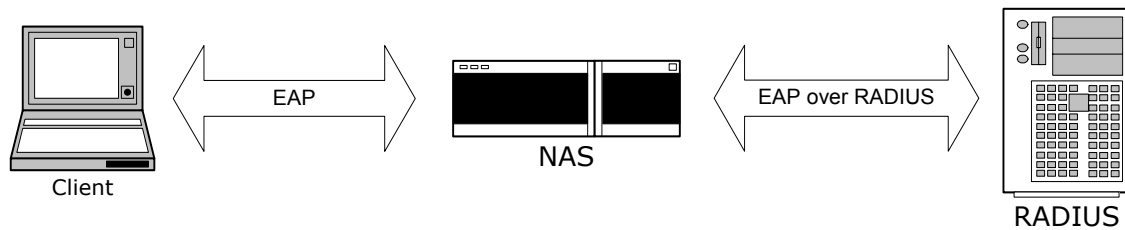
---

<sup>12</sup> PPP : Point-to-Point Protocol (RFC 1548).

<sup>13</sup> Les NAS désignés ici sont par exemple un switch ou un point d'accès WiFi.

<sup>14</sup> EAP over RADIUS est défini dans le RFC 2869.

- EAP-TLS (*Transport Layer Security*) : cette méthode emploie une infrastructure à clés publiques pour authentifier les différentes parties.



## EAP & RADIUS

Afin de supporter EAP, RADIUS définit deux nouveaux attributs `EAP-Message` et `Message-Authenticator`. La séquence classique d'authentification du client se passe comme suit :

- Le client contacte le NAS et demande une identification de type EAP.
- Le NAS envoie alors un paquet de type `EAP-Request/Identity` au client.
- Le client répond par paquet de type `EAP-Response/Identity`.
- Le NAS (qui pour rappel est un client RADIUS), contacte maintenant le serveur RADIUS et lui envoie un paquet de type `Access-Request` en encapsulant la demande du client dans l'attribut `EAP-Message`.
- Le serveur RADIUS qui est compatible EAP, renvoie un paquet de type `Access-Challenge` contenant aussi un attribut `EAP-Message`.
- Le NAS décapsule alors l'attribut et le transfère au client.

Le protocole d'authentification continue alors avec le nombre d'itérations nécessaires aboutissant en fin de parcours soit à un `Access-Accept` ou à un `Access-Reject` de la part du serveur RADIUS.

## 3.4 Problèmes de sécurité, attaques et contre-mesure

### 3.4.1 Les paquets de type Access-Request

Le problème de sécurité réside dans le fait que le standard ne prévoit aucune vérification pour ce type de message. Bien entendu, le serveur RADIUS vérifie qu'un paquet `Access-Request` est bien transmis par un de ses clients, mais les techniques de subtilisation d'adresses IP (*IP spoofing*) sont aujourd'hui une réalité. Si un opposant est capable de se faire passer pour un NAS, il peut alors effectuer une série d'attaques afin de déterminer, par exemple, le mot de passe d'un utilisateur.

La contre mesure proposée est de rendre l'utilisation de l'attribut `Message-Authenticator` obligatoire dans tous les paquets de type `Access-Request`. Si cet attribut n'est pas présent, le serveur ignore le message. L'attribut `Message-Authenticator`<sup>15</sup> est le résultat d'une fonction de hachage utilisant le secret partagé comme clé.

### 3.4.2 Attribut User-Password

L'attribut `User-Password` est protégé par une méthode qui n'est pas considérée comme cryptographiquement sûre. Elle est basée sur un chiffrement en chaîne (similaire au chiffrement de Vernam).

La clé employée pour ce chiffrement est le résultat d'un hachage MD5 du secret partagé concaténé au `Request-Authenticator`. La taille du message à chiffrer est fixée à 16 octets. Si le mot de passe est composé d'au moins de 16 caractères<sup>16</sup>, il est complété par des caractères de valeur *null*. Les deux valeurs sont ensuite XORées et placées dans le paquet envoyé sur le réseau.

L'attaque est basée sur l'idée que, sur un système chargé, le champ `Request-Authenticator` fini tôt ou tard par utiliser une ancienne valeur. Si l'opposant est capable de sniffer le trafic entre un NAS et un serveur RADIUS et de se constituer une base de données de `Request-Authenticator`, il finira par posséder deux messages chiffrés par la même clé. Le résultat d'une opération XOR sur ces deux messages sera deux

---

<sup>15</sup> `Message-Authenticator` = HMAC-MD5 (Type, Identifiant, Length, Request Authenticator, Attributes)

<sup>16</sup> Le RFC prévoit l'éventualité d'un mot de passe faisant plus de 16 caractères. Il est alors découpé en blocs de 16 octets. Une technique itérative est appliquée en recalculant le hachage MD5 sur base du résultat du chiffrement précédent.

password (ou partie de password<sup>17</sup>) XORé, ce qui permettra plus que probablement de déduire le mot de passe.

### 3.4.3 Secret partagé

Une grande critique faite à RADIUS est qu'il autorise l'emploi du même secret partagé pour plusieurs clients (ce qui facilite grandement les attaques). De plus, ce secret n'a pas de longueur minimum et certaine implémentation du serveur (ou certain NAS) le limite parfois à 16 caractères. Enfin, le secret est souvent entré à l'aide d'un clavier n'autorisant pas l'emploi des 256 valeurs possibles pour chaque caractère.

Une attaque peut être tentée pour déterminer le secret partagé. Celle-ci suppose que l'opposant peut soumettre une requête à un NAS et sniffer en même temps le trafic entre ce NAS et le serveur RADIUS. L'opposant soumet un mot de passe de son choix (et de moins de 16 caractères) au NAS et intercepte le paquet *Access-Request*. Vu qu'il connaît le mot de passe, un simple XOR lui renvoie le MD5 (voir ci-avant). Ce hash contient le secret partagé et un *Request Authenticator* connu. L'opposant peut maintenant déterminer (hors connexion), le secret partagé en utilisant une attaque de type *brute force*.

## 3.5 Quelques mots sur Diameter

RADIUS a été la première véritable implémentation d'un protocole AAA. Il est connu et utilisé par la majorité des équipementiers en télécommunication ce qui lui assure encore une belle carrière. Néanmoins, les manquements inhérents à sa conception ont encouragé de nouvelles initiatives. Diameter est l'une d'entre elles et commence déjà à faire parler d'elle. Sa standardisation est toujours en cours<sup>18</sup>, les documents déjà disponibles à l'IETF en sont encore au stade de brouillon (*Internet-Draft*).

Diameter peut se définir comme une amélioration de RADIUS (ce qui n'est pas étonnant étant donné que les notions de bases sont édictées par l'architecture AAA). En quelque sorte, Diameter corrige les défauts de son prédécesseur. Les avantages de Diameter sur RADIUS sont les suivants :

---

<sup>17</sup> Si le mot de passe fait plus de 16 caractères.

<sup>18</sup> Plus d'info à ce sujet sur le site : <http://www.diameter.org>

### **Amélioration du transport :**

- Augmentation conséquente de la taille des paquets (16 Mo au lieu de 4 Ko).
- Utilisation de TCP ou de SCTP<sup>19</sup> en lieu et place de UDP. Ces deux protocoles sont adaptés aux problématiques de congestion réseau.
- Les paquets perdus sont retransmis par le plus proche voisin. Avec RADIUS, lorsqu'une requête est perdue, c'est le NAS à l'origine de cette requête qui, puisqu'il ne recevra pas de réponse, décidera de réémettre une nouvelle demande.
- Une connexion persistante est utilisée entre chaque partie de façon à détecter une panne éventuelle.

### **Amélioration du proxying :**

- Le maillage entre les intervenants combiné à la détection des pertes provoquent la retransmission du paquet au bon endroit. Un intervenant ne pouvant joindre son voisin par défaut transfèrera le paquet en utilisant un autre chemin (si celui-ci permet d'aboutir à destination).
- Un intervenant a la possibilité de mettre des requêtes en attente. Grâce à cela, si un lien est interrompu momentanément (et qu'il est la seule voie d'accès), les paquets seront envoyés dès le rétablissement de celui-ci.

### **Amélioration du contrôle des sessions :**

- La gestion des sessions est indépendante de l'accounting. Un type de paquet spécifique est destiné à la gestion des sessions.
- Le serveur peut demander la clôture d'une session ou demander une ré-authentification.

### **Amélioration de la sécurité :**

- La sécurité de proche en proche est assurée via IPSEC ou TLS.
- Diameter propose une sécurité de bout en bout en utilisant le chiffrement et les procédures de signature digitale.
- Plus besoin de secret partagé.

---

<sup>19</sup> SCTP : Stream Control Transmission Protocol

## 4 Autres méthodes d'identification

---

### 4.1 Kerberos

#### 4.1.1 Description

Ce protocole est souvent considéré comme LE protocole permettant le *single sign on*. C'est-à-dire, qu'une fois l'utilisateur identifié, il ne lui sera plus nécessaire de prouver son identité une nouvelle fois pour accéder à un service compatible Kerberos.

Issue du projet « Athena » du MIT, la version 5 du protocole a été normalisée par l'IETF dans le RFC 1510. Son principe de fonctionnement repose sur la cryptographie à chiffrement symétrique et la notion de ticket.

#### 4.1.2 Principes de fonctionnement

Un serveur d'authentification identifie les clients. Un serveur de tickets produit des tickets d'accès à des services réseaux. Ces deux notions sont en général intégrées dans un seul et même serveur appelé centre de distribution des clés (*Key Distribution Center - KDC*). Les clients sont soit des services, soit des utilisateurs. Une authentification mutuelle est assurée par le partage de clés secrètes entre le KDC et les clients.

Un client souhaitant s'authentifier contacte le KDC qui, après validation, renvoie un message initial chiffré avec la clé du client. Dans ce message, on trouve une clé de session qui sera utilisée pour chiffrer les communications suivantes, et un ticket (*Ticket Granting Ticket - TGT*) permettant l'accès au serveur de tickets. Après déchiffrement de ce message initial, le client peut contacter le serveur de tickets (à l'aide de son TGT) afin d'obtenir un nouveau ticket qui lui permet cette fois d'accéder au service voulu. Le client n'a plus qu'à présenter ce dernier ticket au service pour y avoir accès.

La protection anti-rejeu de Kerberos est basée sur une différence de temps (classiquement 5 minutes), ce qui implique que les horloges des différentes machines intervenant dans les authentifications soient suffisamment synchronisées.

### 4.1.3 Cas d'utilisation

Kerberos fut à l'origine employé sous UNIX. Néanmoins l'utilisation la plus large aujourd'hui est probablement réalisée par les systèmes d'exploitations de Microsoft qui depuis la version 2000 emploient ce protocole lorsqu'ils interagissent au sein d'un domaine Windows.

## 4.2 Infrastructure à clé publique

### 4.2.1 Description

Une infrastructure à clé publique est souvent résumée par l'abréviation PKI (*Public Key Infrastructure*). Elle réside dans la mise en œuvre de solutions basées sur le chiffrement asymétrique (ou chiffrement à clé publique), l'emploi de certificats et la notion de tiers de confiance.

### 4.2.2 Principes de fonctionnement

Un certificat est un fichier dans lequel on retrouve une clé publique, plusieurs informations propres à son propriétaire, et des informations temporelles dont la plus importante est sa date de validité. Un certificat doit être signé (le plus souvent par un tiers de confiance).

Le tiers de confiance est une entité communément appelée autorité de certification (Certification Authority - CA). Cette entité est non seulement chargée de signer les certificats, mais elle garantit également par sa signature la véracité des informations contenues dans le certificat.

La CRL (Certificate Revocation List) contient la liste des certificats révoqués (notamment suite à une compromission). Cette liste est également signée par la CA.

L'authentification se passe de la manière suivante : un client et un serveur possèdent tous deux une clé privée et le certificat associé signé par une même autorité de certification. Le client contacte le serveur et les deux entités échangent leur certificat respectif. Le serveur détecte que le certificat est signé par une CA avec laquelle il a une relation de confiance. Il génère alors un message aléatoire et le chiffre avec sa clé secrète. Le client déchiffre ce message à l'aide de la clé publique du serveur (contenue dans le certificat de celui-ci). Ce même message est ensuite chiffré par le client et renvoyé au serveur. Le serveur reçoit le message chiffré et le déchiffre à l'aide de la clé publique du client. Si le message

transmis et le message reçu sont identiques, l'authentification (mutuelle) est considérée comme un succès !

#### 4.2.3 Cas d'utilisation

Ils sont très nombreux. Le cas qui semble le plus d'actualité est la décision de notre gouvernement d'attribuer au citoyen une carte d'identité électronique (baptisé eID). Cette carte contient deux certificats : le premier étant réservé aux procédures d'authentification et le second à la signature digitale de documents.

### 4.3 Biométrie

#### 4.3.1 Description

La biométrie est un sujet qui n'est pas récent mais qui a été remis à la mode il y a peu, suite à une baisse des prix significative de divers équipements. Le plus célèbre exemple est probablement le lecteur d'empreinte digitale, aujourd'hui présent sur certains ordinateurs portables ou disponible sous la forme de petit périphérique externe. D'autres méthodes, comme les empreintes vocales ou rétiniennes, sont parfois employées.

#### 4.3.2 Principes de fonctionnement

Une donnée biométrique est convertie dans un format binaire et constitue l'identifiant unique d'une personne.

L'identification biométrique peut être abordée de deux manières : soit comme une méthode d'authentification à proprement parler, soit comme preuve au sein d'une méthode d'authentification (tout comme un mot de passe). Cette deuxième solution est souvent considérée comme plus sûre.

Le problème actuel avec les identifiants biométriques est qu'ils peuvent trop facilement être reproduits. C'est le cas par exemple, pour les empreintes digitales qui peuvent être récupérées sur n'importe quel objet touché par une personne.

#### 4.3.3 Cas d'utilisation

En informatique, ils sont encore peu nombreux. Certains fabricants d'OS proposent l'utilisation d'empreintes digitales comme alternative à l'utilisation d'un mot de passe permettant l'ouverture d'une session.

# PARTIE PRATIQUE

## 5 Description et choix d'un serveur

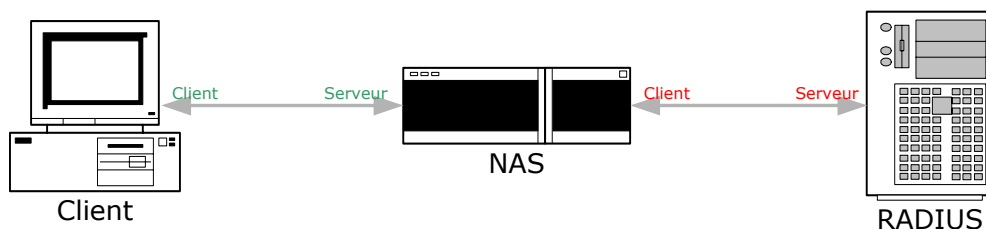
---

### 5.1 Introduction

L'architecture AAA ne se restreint pas uniquement aux serveurs RADIUS. En effet, parmi les produits implémentant cette architecture, Diameter semble être promu à un bel avenir. Malheureusement, à l'heure d'écrire ces lignes, il est encore en cours de standardisation. Les implémentations logiciels de Diameter sont excessivement rares et aucune d'entre elles n'est complètement achevée (et donc utilisable dans un environnement de production). De plus, les micro-logiciels (firmwares) embarqués dans les équipements réseaux proposent souvent un support AAA qui se limite au seul RADIUS. C'est la raison pour laquelle cette partie pratique est dédiée à l'étude des solutions implémentant ce protocole.

### 5.2 Généralités

La grande majorité des équipements qui utilisent le protocole RADIUS le font de la même manière. On a donc deux couples [clients – serveurs] (voir schéma ci-dessous) : le premier formé par le client et le serveur d'accès (NAS<sup>20</sup>) et le deuxième par le NAS et le serveur RADIUS. Le NAS jouant donc tour à tour le rôle de client et de serveur.



#### **Topologie classique (*Pull sequence*)**

Nous allons tout d'abord nous concentrer sur la partie droite du schéma concernant le serveur RADIUS et son client (souvent appelé NAS ou Terminal Server dans la littérature). Un serveur RADIUS est donc un processus à l'écoute sur 3 ports UDP<sup>21</sup> défini à l'IANA<sup>22</sup> :

- 1812 : Pour les opérations d'indentification et d'autorisation (les 4 paquets standards).
- 1813 : Pour les opérations de comptabilisation (*accouting*).
- 1814 : Pour les opération de redirection (*proxying*).

---

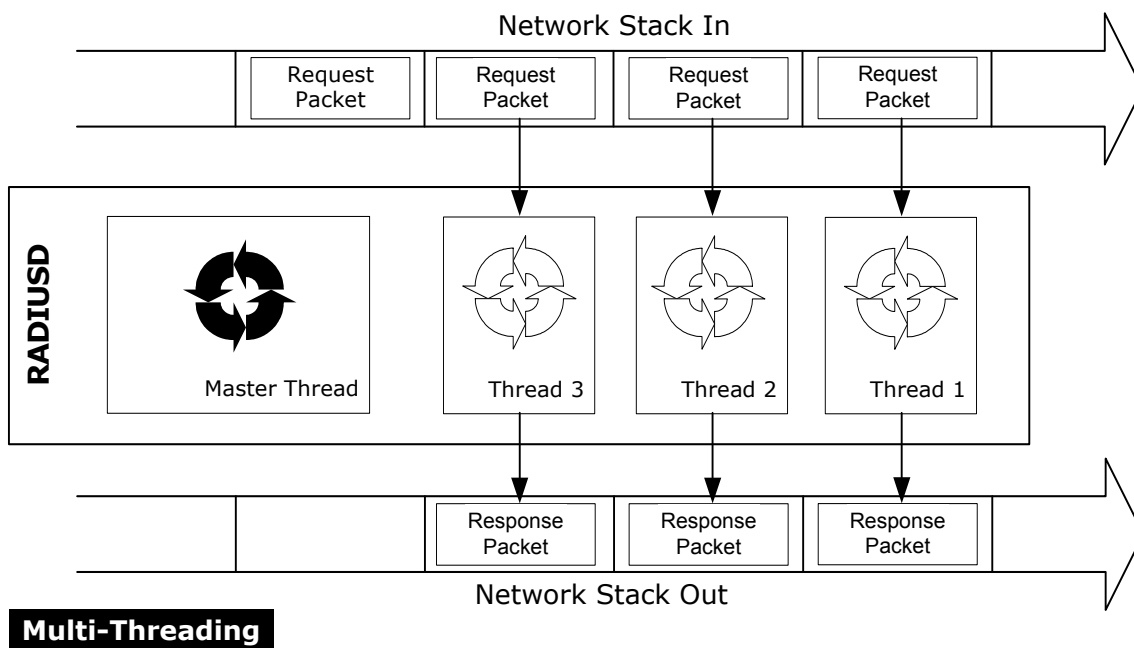
<sup>20</sup> NAS : Network Access Server

<sup>21</sup> Certains équipements utilisent encore le vieux range de port 1645-1647

<sup>22</sup> IANA : Internet Assigned Numbers Authority (<http://www.iana.org/>)

Il est par conséquent très aisé de configurer un pare-feu afin d'autoriser le trafic vers ce serveur.

Le serveur reçoit et transmet un paquet par requête, ce qui représente très peu d'informations (en général 2 kilooctets). Néanmoins, dans un environnement de production tel qu'on peut le concevoir chez un fournisseur d'accès, il s'agira de traiter des centaines de requêtes à la seconde. C'est pourquoi les serveurs implémentés de manière professionnelle utilisent une architecture *multi-thread* permettant ainsi une grande disponibilité du service. Au démarrage, le serveur crée un certain nombre de threads (dans les bons serveurs, ce nombre est configurable), chacun d'entre eux répondant à une requête avant de se terminer et d'être remplacé par un de ses congénères. Le nombre de threads tournant en parallèle permet de dimensionner le serveur en fonction des besoins (cette notion étant directement liée à la charge que le hardware du serveur peut supporter).



Si l'architecture à mettre en place prévoit l'utilisation d'une base de données ou d'un annuaire électronique pour stocker les informations relatives aux utilisateurs, alors chaque requête déclenchera une communication vers le serveur qui héberge ces données. Les serveurs de BD fonctionnent également avec plusieurs threads. Il convient donc d'adapter la configuration du serveur RADIUS (client du serveur de BD) en ce qui concerne par exemple le nombre de sessions parallèles qu'il peut établir avec le serveur de base de données (chaque thread pouvant avoir besoin d'une connexion). De même, il faut veiller à ce que le serveur de BD supporte la charge imposée par le serveur RADIUS.

Dans cette même politique de disponibilité et donc de fiabilité, diverses mesures peuvent (doivent) être mises en place. En effet, il serait du plus mauvais effet qu'un provider perde des données relatives aux connexions de ses clients, générant ainsi des problèmes de facturations, voire une indisponibilité de service.

Tout d'abord, le monitoring des serveurs ne doit pas être pris à la légère. La mise en place et l'utilisation de protocoles tels que SNMP<sup>23</sup> permet de récolter des informations sur divers processus ainsi que l'état de la plupart des NAS. De plus, certains systèmes d'exploitation (ou utilitaires<sup>24</sup> de ces systèmes) permettent de détecter l'arrêt d'un service et ainsi d'immédiatement en provoquer le redémarrage tout en inscrivant l'incident dans un journal.

Ensuite, la communication entre les équipements d'accès et le serveur doit être la plus fiable possible et sécurisée au mieux :

- **Fiabilité** : le protocole utilisé pour faire transiter les messages est UDP, ce qui ne garantit en rien la bonne transmission des informations. Il convient donc que le segment entre les serveurs terminaux et le(s) serveur(s) radius ne soit pas trop sujet aux congestions, de sorte qu'un minimum de paquets soient perdus.
- **Sécurité** : c'est le secret partagé entre un serveur radius et un NAS qui garanti la confidentialité de l'échange des mots de passe. Cette technique étant peu sûre (voir point 3.4.2), il convient de prendre les bonnes dispositions afin de limiter les possibilités d'attaques. La 1<sup>ère</sup> règle de bonne pratique est de ne jamais utiliser 2 fois le même secret de sorte que si un secret est compromis, il suffit de le modifier sur un seul lien client - serveur. Si les messages doivent transiter sur un réseau ouvert, on recommande en plus l'emploi d'une encapsulation IPSEC de type ESP<sup>25</sup> avec un algorithme de chiffrement symétrique tel que 3DES<sup>26</sup>.

Enfin, se prémunir d'un crash matériel éventuel en appliquant une politique de redondance, aussi bien au niveau des serveurs radius qu'au niveau du stockage des données dont ils dépendent et éviter ainsi les problèmes de *single point of failure*. Le plupart des NAS acceptent de configurer plusieurs serveurs radius en cascade, ce qui permet la redondance et la répartition de charges. Les SGBD<sup>27</sup> modernes ainsi que les annuaires peuvent fonctionner en mode « miroir » ou même mieux en cluster, permettant ainsi une haute disponibilité et garantissant également

---

<sup>23</sup> SNMP : Simple Network Management Protocol

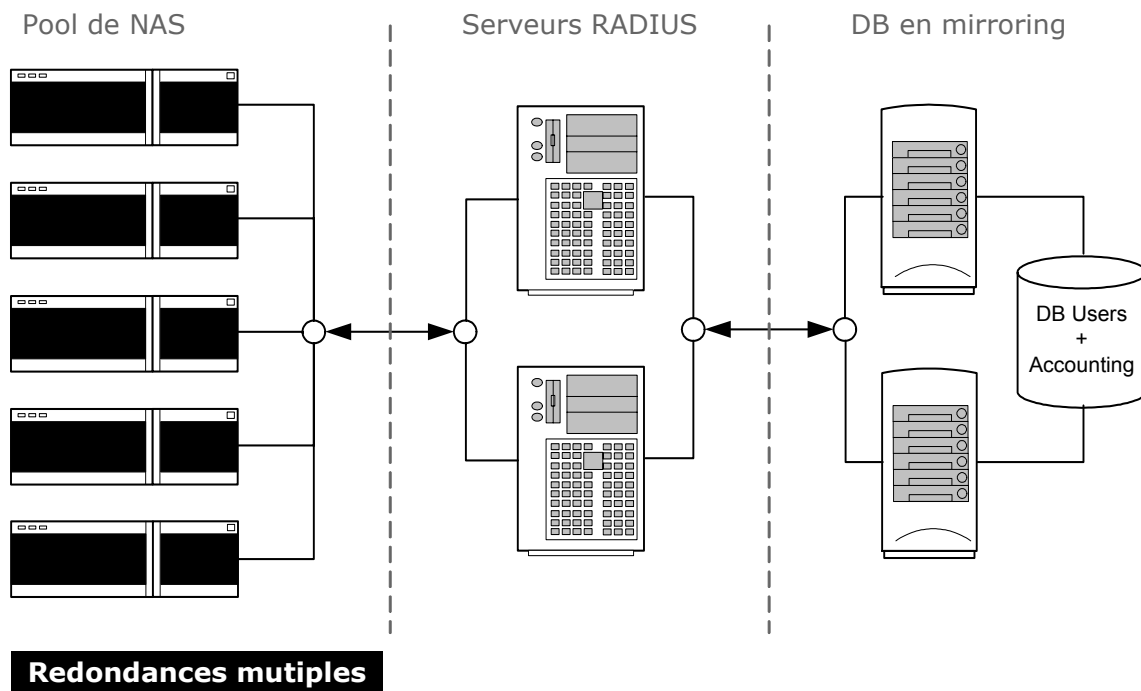
<sup>24</sup> En UNIX, par exemple : *DaemonTools* (<http://cr.yip.to/daemontools.html>)

<sup>25</sup> ESP : Encapsulated Security Payload

<sup>26</sup> 3DES - Triple DES : Triple Data Encryption Standard

<sup>27</sup> SGBD : Système de Gestion de Bases de Données

la sécurité des données. Le schéma ci-dessous représente la combinaison de ces deux techniques.



Bien entendu, ces mesures de sécurité et de disponibilité sont à mettre à l'échelle des besoins et des capacités des entreprises qui les déploient. Une PME qui utilise RADIUS pour ses télétravailleurs et sa téléphonie ne prendra probablement pas les mêmes dispositions qu'un fournisseur d'accès à internet comptant des dizaines de milliers de clients !

Le choix de l'infrastructure à mettre en place et la sélection du serveur dépendront donc directement du budget et du besoin des entreprises d'accéder à cette technologie.

### 5.3 Divers Serveurs

Il est évidemment difficile de dresser une liste exhaustive de tous les serveurs radius existants et de leurs capacités. De plus, cela n'aurait probablement aucune utilité. Malgré tout, nous allons tâcher d'en présenter quelques-uns, souvent cités dans la littérature ou bien liés à l'histoire du protocole.

Comme fréquemment en informatique, l'on est devant un choix difficile, quasi religieux ; d'un côté il existe des logiciels gratuits et libres de droits (le plus souvent à code source ouvert) et de l'autre le monde du logiciel propriétaire. Dans la plupart des cas, c'est la stratégie de

l'entreprise en matière de gestion d'avares logiciels qui favorisera le choix de l'un par rapport à l'autre.

Le logiciel libre est souvent très présent lorsqu'il s'agit d'utiliser des standards. C'est probablement pour cette raison qu'il existe énormément d'initiatives autour de radius dans cette communauté.

Historiquement, la première implémentation de RADIUS fut celle de Livingston. Son successeur open source est Cistron<sup>28</sup>, très employé dans les implémentations professionnelles. Il n'est plus développé aujourd'hui, mais est encore maintenu (notamment en cas de découverte de failles éventuelles).

Le projet qui lui a succédé porte le nom de FreeRADIUS. Ce dernier a été créé à l'initiative d'un des développeurs principaux de Cistron. Ce serveur est totalement compatible avec tous les standards RADIUS publiés par l'IETF<sup>29</sup> dont les plus récents (EAP). Son architecture modulaire lui confère une très grande souplesse, devenant ainsi une bonne solution dans la majorité des cas.

Beaucoup d'autres serveurs open source sont disponibles. Les plus célèbres sont probablement GNU Radius, issu des pionniers du logiciel libre dont la *free software foundation*, et JRadius qui est l'alter ego de FreeRADIUS pour le monde Java.

En ce qui concerne les serveurs propriétaires, ils sont également très nombreux. Néanmoins, comme pour les libres, certains d'entre eux sortent du lot pour des raisons diverses. Navis Access est l'un de ceux-là, c'est un produit de Lucent technologies (qui racheta Livingston Enterprises il a quelques années). Ce rachat du créateur du premier serveur RADIUS (et donc des contrats d'entretien fait par Livingston) vaut à Lucent d'être une référence.

D'autres produits intéressants existent sur le marché. S'il fallait choisir parmi les serveurs payants, nous conseillerions probablement Radiator<sup>30</sup>. Ce serveur a la particularité d'être écrit en Perl<sup>31</sup>, ce qui fait de lui un programme à code source ouvert, mais dont la licence est payante. Les interpréteurs Perl étant disponibles sur la quasi totalité des plateformes permettent d'exécuter Radiator sur la grande majorité des systèmes. Sa grande souplesse dans la gestion des utilisateurs et de l'accounting, facilite l'implémentation dans des contextes très variés. Et enfin, la société propose une modification du protocole RADIUS offrant la possibilité à deux de leurs serveurs de faire du proxying au travers de TCP

---

<sup>28</sup> Cistron RADIUS Server : <http://www.radius.cistron.nl/>

<sup>29</sup> IETF : Internet Engineering Task Force (<http://www.ietf.org/>)

<sup>30</sup> Radiator : <http://www.open.com.au/radiator/>

<sup>31</sup> Perl est un langage interprété très répandu : <http://www.perl.org/>

(plutôt que UDP), permettant ainsi d'utiliser TLS pour établir un lien sécurisé. Malheureusement, cette modification du protocole baptisée RadSec ne fait pas l'objet d'une standardisation, et donc seul Radiator peut bénéficier de cette amélioration.

Au long de nos investigations sur les serveurs propriétaires, nous avons eu l'occasion d'essayer le produit que Microsoft propose à ses clients : Internet Authentication Service (IAS). Si vous possédez une licence Windows Server et si (et seulement si) votre base de données d'utilisateurs est Active Directory, ce service (compris dans la licence du Serveur) est très rapidement installé et sera vite configuré à condition que vous connaissiez quelque peu RADIUS. Lorsque que l'infrastructure RADIUS est simple et que la stratégie de l'entreprise est « tout Microsoft », ce choix peut s'avérer judicieux.

Nous nous arrêterons là en ce qui concerne cette micro étude comparative des serveurs disponibles. Nous concluons en rappelant que le choix d'une solution doit s'établir en fonction de plusieurs aspects et qu'un bon questionnement permet souvent d'orienter son choix :

- Quels sont les NAS utilisés ? En effet, même si radius est un standard, le serveur doit connaître les attributs spécifiques des vendeurs utilisé par vos NAS.
- Les protocoles utilisés pour l'identification (CHAP, EAP, ...) sont-ils implémentés dans le serveur ?
- Comment sont stockées les données relatives à l'identification des utilisateurs et à leurs droits ? (exemples : Fichiers plats, DB système, DB SQL)
- Avez-vous un besoin critique des données de comptabilisation (accounting) des utilisateurs ? (stockage dans fichiers de log standard ou utilisation d'une DB)
- Quel est le degré tolérance aux pannes de l'infrastructure à mettre en place ?
- Quelle est la politique en matière d'acquisition logicielle ?
- Si vous configurez vous-même le serveur : à quel type de système êtes-vous familier ?

D'autres intentions peuvent bien sûr être prises en considération et le but de ce mémoire n'est pas de privilégier un serveur au détriment d'un autre.

Pour ce travail, le choix s'est porté sur FreeRADIUS. Les motivations ont été diverses ; les principales sont développées ci-après. Cela étant, quel que soit le serveur choisi, les fonctionnalités se ressemblent et les configurations sont similaires. Le choix d'étudier ce serveur plutôt qu'un autre n'empêchera pas le lecteur de constater de grandes similitudes avec la plupart des autres produits (qui après tout

sont tous aussi des serveurs RADIUS). Voici tout de même quelques éléments qui ont retenu notre attention pour FreeRADIUS :

- Le respect de tous les standards<sup>32</sup> y compris les plus récents.
- La possibilité d'avoir diverses bases de données utilisateur : Fichiers plats, DB systèmes, DB SQL, LDAP<sup>33</sup>, exécution de programmes externes, ...
- Une communauté de développeurs très active, corrigeant les bugs et trous de sécurité rapidement, le tout autour d'un serveur dont les fonctionnalités vont croissant.
- « Petit-fils » du serveur historique de Livingston (voir ci-dessus).
- Une documentation relativement bonne autour du projet, notamment assurée par le livre de Jonathan Hassel<sup>34</sup>.
- La possibilité de pouvoir stocker les journaux (logs) d'authentification et d'accounting dans une base de données, facilitant ainsi le traitement de ceux-ci.
- La portabilité du serveur : nous l'avons testé de façon concluante aussi bien sous Windows<sup>35</sup> que sous Linux (il est également connu pour s'exécuter sous la plupart des environnements UNIX et BSD).
- Une interface graphique en mode web permettant de gérer les utilisateurs stockés dans la base de données et d'interpréter les informations journalisées dans les tables.
- La gratuité de la licence.
- Plus de détails sur : <http://www.freeradius.org>

---

<sup>32</sup> Voir les RFC publiés par l'IETF dans les références

<sup>33</sup> LDAP : Lightweight Directory Access Protocol

<sup>34</sup> Voir documents de références

<sup>35</sup> Malgré tout, déconseillé dans un environnement de production

## 5.4 Plus en avant dans FreeRADIUS

### 5.4.1 Installation

Sur certains systèmes, la compilation s'impose. Pour ce faire, il suffit de se procurer les sources sur le site de FreeRADIUS et d'avoir les utilitaires suivants : `tar`, `gzip`, `glibc`, `binutils` et `gmake`. Décompresser les sources dans le répertoire de votre choix : `tar -zxvf freeradius-1.0.X.tar.gz` et rendez-vous dans le répertoire racine de l'arborescence ainsi créée. Lancer simplement `./configure [--options]` (les options par défaut suffisent dans la majorité des cas : `./configure --localstatedir=/var --sysconfdir=/etc`). Compiler ensuite les exécutables avec la commande `make` et créer les fichiers de configuration par défaut avec `make install`. Votre système est prêt à être configuré !

Pour les adeptes des packages<sup>36</sup> (notre cas), les divers éditeurs de distribution permettent une installation aisée et automatisée. Par exemple, sous Debian, entrez la commande `apt-get install freeradius` et le tour est joué ! Veillez toutefois à installer les modules additionnels dont vous pourriez avoir besoin (ex : extensions `ldap`, `mysql`,...).

Une installation automatisée de binaires est également disponible pour Windows à l'adresse <http://www.freeradius.net>. Elle est proposée avec la plupart des modules (attention : le seul SGBD supporté par cette distribution est PostgreSQL).

### 5.4.2 Architecture et configuration de base

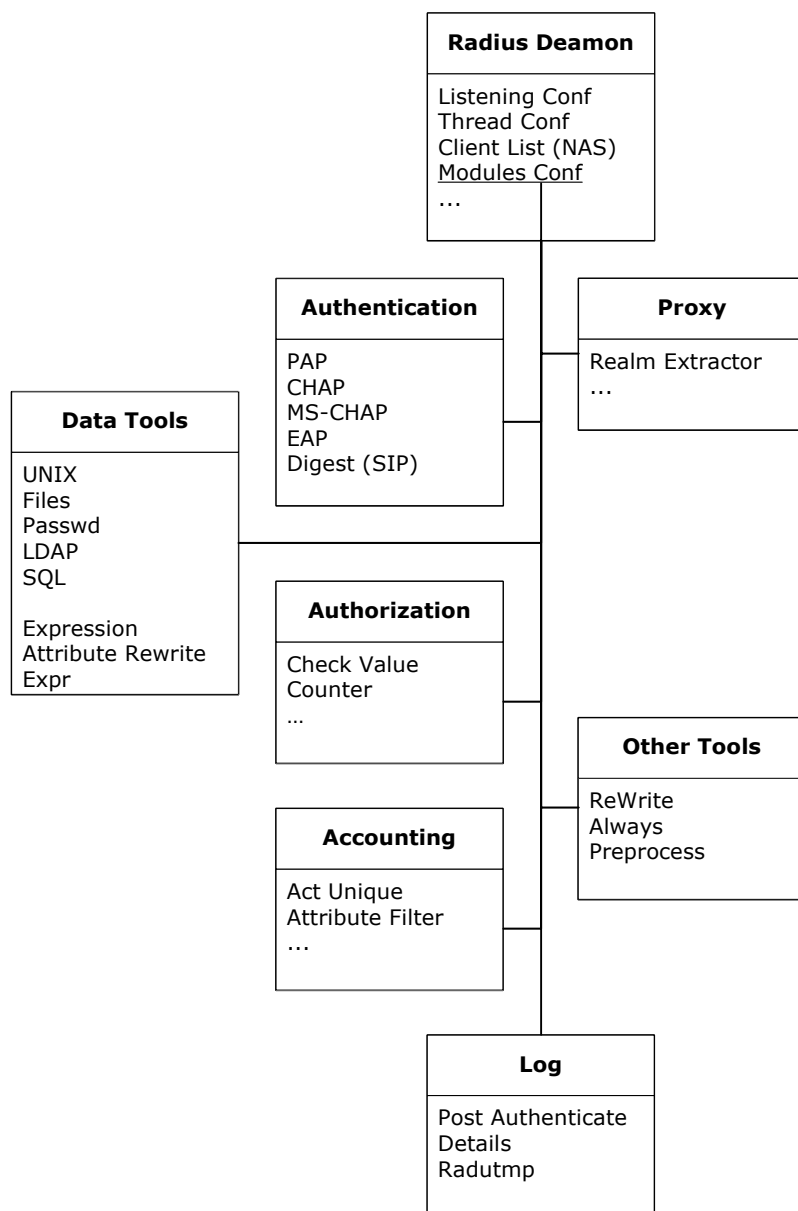
Le daemon<sup>37</sup> FreeRadius prend simplement le répertoire contenant les fichiers de configuration comme argument. Il s'attend à y trouver deux fichiers `radiusd.conf` et `dictionary`. Le reste des fichiers est inclus au sein des deux précités par une simple instruction `$INCLUDE`.

L'architecture de FreeRADIUS est évidemment multi-threadée (voir ci-dessus), mais est surtout très modulaire et peut paraître très complexe de prime abord si on la présente dans sa totalité. Toutefois, en regroupant les modules par fonctionnalités, le schéma suivant devient plus abordable (ceci n'est qu'une représentation, tous les modules peuvent virtuellement interagir entre eux).

---

<sup>36</sup> Archives autoinstallables pré-compilées.

<sup>37</sup> Processus serveur dans le monde UNIX



Tout au-dessus, le processus de base qui invoque les autres modules en fonction de son fichier de configuration.

D'abord, les modules propres aux proxying qui décident de transférer ou non la requête à un autre serveur.

Ensuite les modules d'identification et d'autorisation qui, à l'aide des modules d'accès et de manipulation des données, vont permettre de déterminer si la requête est acceptée ou rejetée en fonction de diverses conditions paramétrables.

Puis vient le module qui gère la comptabilisation qui peut d'une part fonctionner en collaboration avec les modules d'autorisation pour rejeter certains utilisateurs (ex : pas de sessions concurrentes, temps journalier limité), et d'autre part stocker les informations nécessaires à la facturation (en utilisant, par exemple, une base de données SQL).

Finalement, les modules relatifs à la journalisation des événements (ne faisant pas partie de l'accounting) tels que les requêtes d'identification et leur réponse.

### 5.4.3 Gestion des types d'authentification

La plupart des serveurs permettent d'activer/désactiver les différentes méthodes d'authentification qu'ils supportent. Un petit rappel des protocoles disponibles et leurs propriétés :

- **PAP** : *Password Authentication Protocol*. C'est la méthode la plus simple. Le mot de passe voyage en clair entre la client et le NAS (cet échange étant encapsulé par PPP<sup>38</sup> par exemple). Il est ensuite chiffré par le secret partagé entre le NAS et le serveur

<sup>38</sup> PPP : Point to Point Protocol.

radius. Une fois le mot de passe reçu, il est déchiffré et comparé à celui stocké dans la base de données.

- **CHAP** : *Challenge-Handshake Authentication Protocol*. Ici, le mot de passe n'est pas envoyé sur le réseau mais on envoie une preuve de la connaissance de celui-ci au travers du succès d'un challenge (un défi), ce qui peut sembler plus intéressant qu'un mot de passe circulant en clair. Malheureusement, l'inconvénient de cette technique est qu'elle nécessite d'avoir accès au véritable mot de passe, et non à un hachage à sens unique de celui-ci. En effet, les règles de bonne pratique en ce qui concerne le stockage des mots de passe recommandent vivement de ne pas enregistrer ce dernier en clair (afin de réduire notamment la possibilité d'usurpation d'identité). C'est ce que font d'ailleurs par défaut la majorité des systèmes d'exploitations.
- **MS-CHAP** : *Microsoft Challenge-Handshake Authentication Protocol*. Cette technique mise au point par la firme de Redmond est une modification de CHAP afin de pallier le problème d'accès au mot de passe. Elle permet en effet d'effectuer une authentification par challenge sans que le serveur n'ait à connaître réellement le mot de passe (le serveur ne connaissant que le hash de celui-ci). Deux versions co-existent, la 2<sup>ème</sup> étant plus sûre car créée pour pallier le manque de la 1<sup>ère</sup>.
- **EAP** : *Extensible Authentication Protocol* qui n'est pas un protocole d'authentification à proprement parler. Il s'agit en fait d'un protocole d'accès, tout comme PPP, qui permet (à l'instar de ce dernier) une plus grande souplesse dans la négociation de la méthode d'authentification. C'est donc au niveau de la 2<sup>ème</sup> couche du modèle OSI (802.1x) qu'il intervient. Pour l'utiliser, il faut bien entendu disposer de NAS (points d'accès sans fil ou switches) compatibles avec ce protocole. Ces derniers étant configurés pour transférer les requêtes AAA vers un serveur radius par exemple. FreeRADIUS supporte diverses méthodes d'identifications EAP que sont EAP-MD5, LEAP, EAP-GTC, EAP-SIM, EAP-TLS, EAP-TTLS, EAP-PEAP.

Remarque : le protocole *digest* (issu de l'authentification http) est également disponible mais assez peu employé. L'idée est la même que pour MS-CHAP lors de l'entrée en session, les preuves d'identités qui continuent à circuler durant la session étant dérivée de divers paramètres aléatoires passés lors de l'initiation du protocole et modifiés au cours des échanges. Ce dernier est utilisé notamment par certains serveurs SIP<sup>39</sup>.

---

<sup>39</sup> SIP : Session Initiation Protocol

Afin d'activer PAP, CHAP et MS-CHAP dans FreeRADIUS, il suffit que ces lignes soient présentes dans le fichier `radiusd.conf` :

```
pap {encryption_scheme = md5}
chap {authtype = CHAP}
mschap {authtype = MS-CHAP}
```

Le paramètre utilisé pour PAP est la méthode de hashing utilisée pour stocker le mot de passe de l'utilisateur (clear, crypt, md5 ou sha1). La méthode proposée par Microsoft peut prendre d'autres paramètres particuliers à certaines implémentations. Mais pour une utilisation générique, ils peuvent être ignorés.

Pour EAP : se rendre dans le fichier `eap.conf` (inclus par défaut dans le fichier de configuration principal). La variable `default_eap_type` permet de choisir un type d'authentification par défaut. Le reste de la configuration dépendra des méthodes EAP que l'on souhaite utiliser (voir l'exemple au point 7.1.5).

#### 5.4.4 Gestion des Utilisateurs et de leurs droits

##### 5.4.4.1 Avec des fichiers plats

Le support de l'identification au travers de simples fichiers est activé par défaut dans `radius.conf` via le module *files* qui assure également la rétro-compatibilité avec les fichiers du serveur Livingston et également avec Cistron. Ces fichiers sont lus de manière séquentielle.

Le serveur utilise des opérateurs pour gérer la logique de traitement (parsing) des fichiers utilisateurs. Le schéma classique attribut – opérateur – valeur (appelé paire A/V) est utilisé. Les principaux opérateurs sont définis dans le tableau ci-dessous :

Opérateur	Signification
<i>Affectation</i>	
=	Affectation d'une valeur
+=	Affectation d'une valeur en plus
:=	Affectation d'une valeur avec écrasement des valeurs précédentes
<i>Condition</i>	
==	Egalité
=*	Toujours égal (accepte toutes les valeurs)
!*	Jamais égal (rejette toutes les valeurs)
!=	Différent
>=	Plus grand ou égal que
<=	Plus petit ou égal que
>	Plus grand que
<	Plus petit que

Trois fichiers sont utiles pour organiser les données des utilisateurs :

- `users` : qui, comme son nom l'indique, contient les données des utilisateurs à proprement parler.
- `huntgroups` : permet d'affecter des groupes de NAS à des ensembles d'utilisateurs.
- `hints` : permet de modifier une requête en se basant par exemple sur un préfixe ou un suffixe appliqué à certains noms d'utilisateurs.

La définition BNF suivante définit la syntaxe de ces fichiers :

```
<utilisateur> ::= <1ère ligne> [<ligne suivante>]
<1ère ligne> ::= <username> <Paire A/V> [',' <Paire A/V>] <EOL> (*)
<ligne suivante> ::= <Paire A/V> {',' <EOL> <Paire A/V>}
<username> ::= <STRING> | 'DEFAULT'

<Paire A/V> ::= <attribut> <op> <valeur>
<attribut> ::= 'Auth-Type' | 'User-Password' | 'Service-Type' | ...
<op> ::= '=' | ':' | '=' | '*' | '!*' | '!=' | '>=' | '<=' | '>' | '<'
<valeur> ::= <INT> | <ENUM> | <STRING> | <IPADDR> | <DATE> | <BINARY>
```

(\*) Les paires A/V utilisables à la 1<sup>ère</sup> ligne sont :

- soit une affectation de l'attribut `Auth-Type` à un module d'authentification externe (ex : `Auth-Type := System`) ou un refus systématique en utilisant `Auth-Type := Reject`,
- soit de cette même affectation de l'attribut `Auth-Type` suivie d'une paire A/V conditionnée (ex : `Auth-Type := Local, User-Password == "X"`).

Les deux exemples qui suivent permettent une meilleure visualisation du fichier `users`. La 1<sup>ère</sup> définition « `badguy` » provoquera une réponse de type *Access-Reject*. La 2<sup>ème</sup> « `steve` », après avoir été dûment validée par une authentification système, renverra un paquet *Access-Accept* contenant toutes les informations dont le NAS a besoin pour établir la connexion.

Exemple 1 :

```
badguy  Auth-Type := Reject
        Reply-Message = "Account disabled."
```

Exemple 2 :

```
steve   Auth-Type := System
        Service-Type = Framed-User,
        Filter-ID = "intranet",
        Framed-Protocol = PPP,
        Framed-IP-Address = 172.16.3.33,
        Framed-IP-Netmask = 255.255.255.0,
        Framed-MTU = 1500,
        Framed-Compression = Van-Jacobson-TCP-IP
```

L'attribut spécial `Fall-Through` donne l'instruction au parseur de ne pas arrêter sa lecture du fichier, afin de prendre en compte, par exemple, les entrées `DEFAULT`. Exemple :

```
test    Auth-Type := Local, User-Password == "test"
        Service-Type = Login-User,
        Fall-Through = Yes

DEFAULT Service-Type == Login-User
        Login-Service = Rlogin,
        Login-IP-Host = localhost,
        Reply-Message = "Hello, %u. Have a nice day !"
```

Les entrées particulières `DEFAULT` se trouvent généralement en fin de fichier et ne sont atteintes que si le parseur n'a pas trouvé le nom d'utilisateur ou s'il a rencontré l'instruction `Fall-Through = Yes`. De plus, celles-ci peuvent se contenter d'une simple paire A/V de conditions puisqu'elles viennent en quelque sorte s'ajouter à l'entrée d'un utilisateur.

#### **5.4.4.2 Avec une base de données SQL**

FreeRADIUS permet de se passer de fichiers d'utilisateurs et de stocker un contenu tout à fait équivalent dans les tables d'une base de données. Le serveur supporte à l'origine un grand nombre de SGBD et peut être étendu à virtuellement n'importe lequel de ces derniers au travers de ses extensions `unixODBC` et `iODBC`<sup>40</sup>. L'architecture pour interagir avec les bases de données se compose d'un module SQL générique auquel est attaché le driver spécifique du SGBD utilisé. Les exemples qui suivent sont basés sur MySQL mais les adaptations à effectuer pour passer à un autre SGBD sont mineures.

Avant de connecter le serveur radius à une base de données, il faut bien entendu créer les tables de celle-ci. Pour ce faire, il faut d'abord extraire des sources le script SQL d'instanciation des tables (fourni avec les sources) et l'exécuter. Le schéma par défaut des tables et le dictionnaire des données sont présentés dans l'annexe 1.

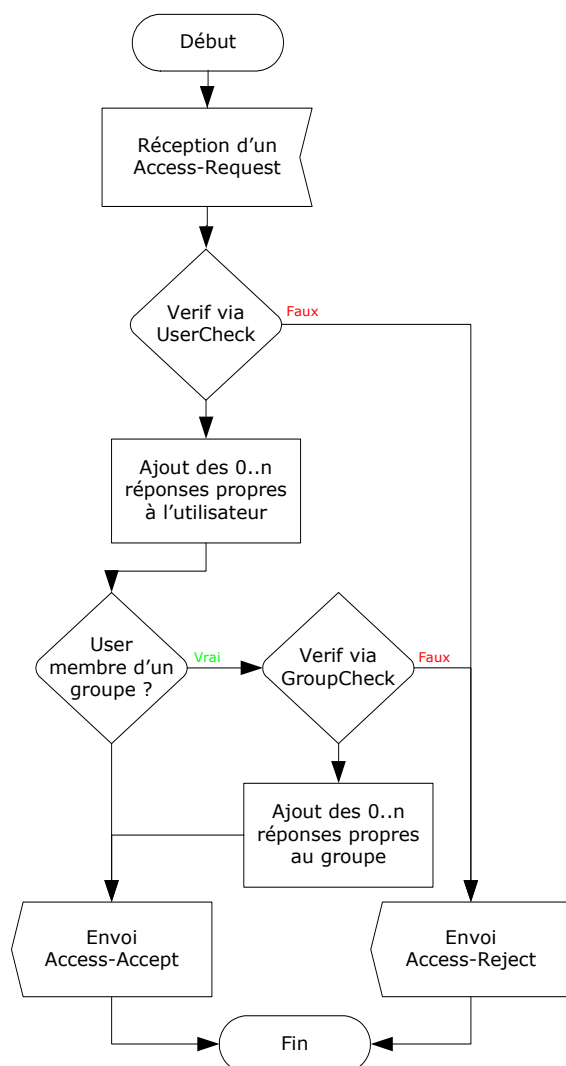
La configuration de l'accès au serveur de base de données se fait par l'intermédiaire du fichier `sql.conf`. Les quelques variables utilisées sont très explicites, elles permettent d'ouvrir la(les) connexion(s) vers le serveur et de se brancher à la base de données précédemment créée. Afin de laisser à l'administrateur de la base de données le maximum de

---

<sup>40</sup> ODBC : Open DataBase Connectivity (disponible sur les systèmes Microsoft). `unixODBC` (<http://www.unixodbc.org/>) et `iODBC` (<http://www.iodbc.org/>) sont deux projets concurrents qui proposent tous deux une interface standardisée d'accès à la base de données dans les environnements LINUX/UNIX.

flexibilité, le nom des tables et les structures proposées dans le script de création ne doivent pas forcément être respectés. Les requêtes SQL envoyées par le serveur au SGBD sont toutes configurables. Seuls le type et l'ordre des champs devant être retournés par les clauses SELECT doivent être respectés. Cette souplesse d'interaction avec les bases de données permet par exemple de configurer très finement les opérations de journalisation.

L'organisation en tables est assez simple. Deux tables *radcheck* et *radreply* représentent ce que l'on retrouve dans le fichier *users*. Les structures de celles-ci sont identiques : un tuple contient un nom d'utilisateur et une paire A/V. La 1<sup>ère</sup> table est l'équivalent de ce que l'on trouve à la première ligne dans le fichier *user* (typiquement : un couple login / password), la 2<sup>ème</sup> est la traduction dans la base de données des réponses à envoyer par rapport à une requête. Les tables *radgroupcheck* et *radgroupreply* sont les équivalentes des deux tables précédentes, mais elles représentent des groupes d'utilisateurs. Leur mode de fonctionnement est le même. La table *usergroup* permet d'affecter un utilisateur à un groupe.



Le schéma ci-contre décrit mieux la logique de traitement effectuée par FreeRADIUS au travers des informations stockées dans les tables.

La fonction `UserCheck` va déterminer si les conditions reprises dans la table *radcheck* sont bien remplies. Dans l'affirmative, le traitement se poursuivra par l'ajout des attributs contenus dans la table *radreply* pour cet utilisateur.

Si l'utilisateur est membre d'un groupe, on bascule alors dans une séquence similaire à la précédente. On vérifie les conditions propres à ce groupe et, si elles sont remplies, l'on ajoute les attributs.

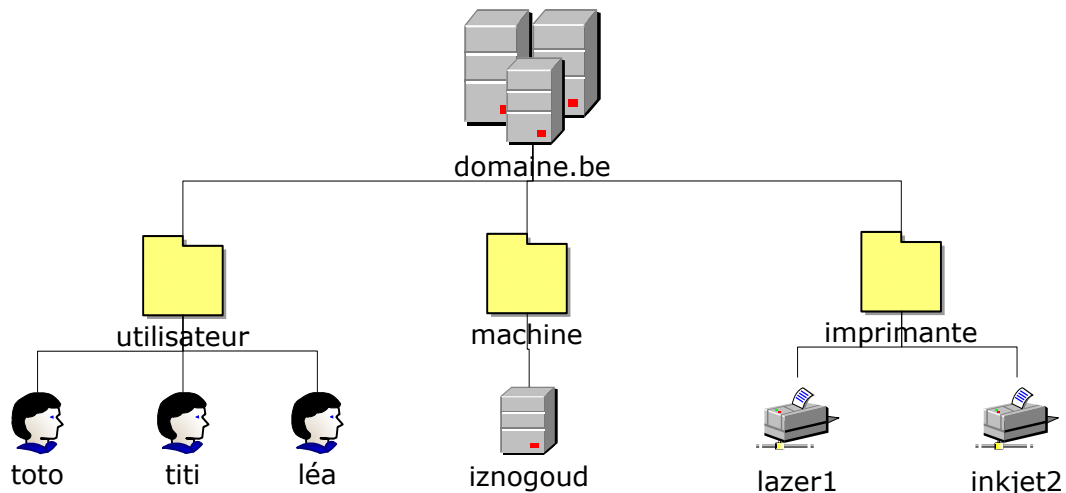
Le traitement se clôture par l'envoi d'un paquet de type `Access-Accept` ou de type `Access-Reject` contenant les attributs ajoutés tout au long du parcours.

### 5.4.4.3 Avec un annuaire LDAP

A l'origine, LDAP est un protocole d'accès à un annuaire électronique via un réseau IP (il s'agit en fait d'une adaptation du protocole DAP<sup>41</sup> permettant l'accès aux services d'annuaire X500 de l'OSI<sup>42</sup>). Par extension, LDAP est devenue une technologie à part entière assurant l'accès et le stockage des informations. Cette nouvelle génération d'annuaires "légers" permet d'entreposer des données très diverses allant des simples coordonnées d'un utilisateur aux droits d'accès de celui-ci pour une application déterminée.

Ces annuaires sont similaires aux bases de données orientées objet à ceci près qu'ils sont optimisés pour la recherche d'informations (accès rapide) et non pour le stockage de celles-ci (accès lent). En effet, ceci s'explique par le fait que l'information y est stockée sous forme d'arbre. Un nœud de cet arbre correspond à un objet du monde réel (une imprimante, un ordinateur, un utilisateur, ...) ; chaque nœud peut avoir entre 0 et n fils. Un serveur LDAP est également caractérisé par schéma définissant les classes d'objets, leurs attributs et leurs syntaxes.

Chaque entrée est référencée de manière unique dans l'arbre par son distinguished name (DN). Celui-ci représente le chemin absolu vers un nœud de l'arbre. Par exemple la chaîne "uid=toto, ou=utilisateur, dc=domaine, dc=be" représente le chemin d'accès vers l'objet toto inclus dans le conteneur utilisateur de l'arbre dont la racine est domaine.be.



**Exemple d'arborescence LDAP**

<sup>41</sup> DAP : Directory Access Protocol

<sup>42</sup> OSI : Open System Interconnection, modèle standardisé par l'ISO : Organisation Internationale de Normalisation

Les requêtes LDAP nécessitent divers arguments. Le DN en est le principal ; il permet de définir le nœud qui deviendra la racine du sous-arbre qui sera parcouru lors de la recherche. Un autre de ces arguments est le filtre. Il s'agit d'une chaîne de caractères à la syntaxe bien définie. Le filtre "`(&(uid=t*) (ou= utilisateur))`" renverra toutes les entrées appartenant au conteneur `utilisateur` dont l'attribut `uid` commence par la lettre `t`.

Les explications précédentes devraient être suffisantes pour utiliser un serveur LDAP à des fins d'authentification. Cela étant, il peut être également utile de placer les informations d'autorisation dans LDAP.



toto



radiusprofile

Pour ce faire, il est nécessaire de créer des objets contenant des attributs propres à RADIUS dans l'annuaire. FreeRADIUS propose un fichier définissant une classe (`radiusprofile`) et ses attributs à ajouter au schéma d'un annuaire existant. Ainsi, chaque nœud représentant un utilisateur peut se voir ajouter un fils de type `radiusprofile` contenant tous les attributs utiles pour l'autorisation.

Si le schéma LDAP possède déjà les attributs nécessaires à l'affectation des droits, il suffit alors de modifier le fichier `ldap.attrmap` qui effectue le lien entre les attributs du dictionnaire RADIUS standard et les attributs du serveur LDAP.

En ce qui concerne l'authentification, deux méthodes sont disponibles pour valider un utilisateur avec un annuaire. La première consiste à se connecter au serveur LDAP, de lancer une recherche sur l'attribut définissant le login de l'utilisateur. Si la recherche est fructueuse (c-à-d renvoie bien, un et un seul objet), lire au sein de cet objet la valeur de l'attribut contenant le mot de passe et effectuer la comparaison avec le mot de passe soumis au serveur RADIUS (ou valider un challenge). Malheureusement, cette 1<sup>ère</sup> méthode prévoit qu'un attribut contenant le mot de passe existe (ce qui n'est pas toujours le cas, par exemple avec l'annuaire d'Active Directory) !

L'autre méthode se fait en deux phases et suppose que le serveur LDAP requiert un login et un mot de passe pour la connexion. Dans un premier temps, on s'y connecte en utilisant un couple login + password qui permet l'exécution de requêtes, on vérifie ensuite la présence de l'utilisateur (comme dans la méthode précédente) et si la recherche est concluante : on ferme la connexion. La deuxième phase démarre alors et on se reconnecte au serveur LDAP en utilisant cette fois le login et le mot de passe soumis au serveur RADIUS. Si la connexion est acceptée avec cet identifiant, l'authentification est alors considérée comme un succès !

#### 5.4.4.4 Autres méthodes

FreeRADIUS possède également d'autres modules contribuant à l'identification des utilisateurs :

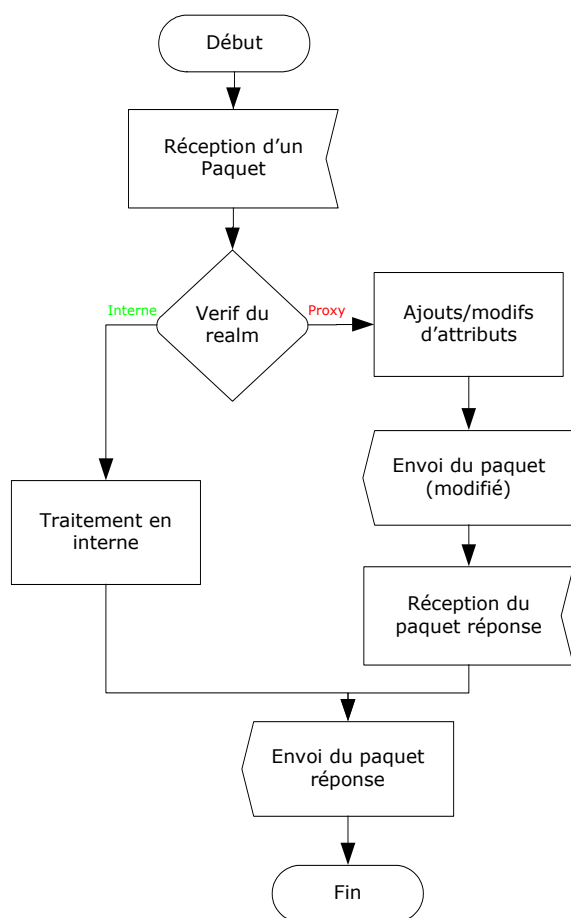
- **UNIX** : déjà abordé dans l'utilisation des fichiers plats. C'est le module qui est invoqué lorsque l'attribut `Auth-Type` est égal à `System`. Son paramétrage est simple, il suffit de lui indiquer l'emplacement du fichier `passwd` ou du fichier `shadow` et de configurer le module PAP en fonction de l'encryption<sup>43</sup> activée sur le système (par exemple `crypt`). Des paramètres permettant de bufferiser les fichiers sont disponibles afin d'améliorer les performances.
- **samba** : si votre système UNIX utilise cet add-on qui donne accès aux services Windows de partages de fichiers et d'imprimantes, le fichier `smbpasswd` contient dès lors des couples login / password pouvant être utilisés comme données d'identification. Cette prise en charge est assurée par le module `passwd`.
- **ntlm**<sup>44</sup> : cette méthode n'est pas un module en tant que tel. Il s'agit d'une option du module `mschap` qui permet de transférer une Access-Request de type `ms-chap` vers un contrôleur de domaine windows.
- **smb** : Même principe que NTLM avec PAP. Cette fois, il s'agit d'un module.
- **exec, perl, python** : sont tous trois des modules qui permettent d'exécuter des programmes externes. Ils ne sont pas utilisés directement pour identifier les utilisateurs mais il peuvent être appelés à partir du fichier `users` (par le biais d'une expression formatée) pour, par exemple, valider un mot de passe.

---

<sup>43</sup> Attention : si elle est activée, l'utilisation de CHAP n'est plus possible !

<sup>44</sup> NTLM : Méthode propriétaire de Microsoft semblable à CHAP.

### 5.4.4.5 Proxying



Cette dernière méthode est proposée par le standard RADIUS. Elle consiste à transférer la requête reçue à un autre serveur RADIUS. La difficulté de cette opération relais est, d'une part d'aiguillier correctement la requête et d'autre part, de configurer le serveur afin qu'il transforme celle-ci en une requête qui sera acceptée par leur serveur destinataire.

Le schéma ci-contre présente la manière dont le serveur décide de transférer ou non la requête.

Par exemple : une requête dont le nom d'utilisateur est "toto@domaine.be", devra être transférée au serveur "aaa.domaine.be" et le nom d'utilisateur sera modifié en "toto".

Cette technique a pour effet d'inverser les rôles. En effet, le serveur ayant reçu la demande originale devient client du serveur auquel il transmet cette demande. C'est pourquoi la configuration de cette partie du serveur ressemble à celle des NAS.

Dans FreeRADIUS, la configuration de l'aiguillage des paquets est écrite dans le fichier `proxy.conf`. Exemple d'une instruction dans le 1<sup>er</sup> fichier qui implémente l'exemple ci-dessus :

```
realm45 domaine.be {
  type = radius
  authhost = aaa.domaine.be:1645
  secret = lesecretpartagéavecAAA
}
```

La suppression du realm (ici, domaine.be) est automatique dans FreeRADIUS et elle ne nécessite pas d'ajout d'instruction dans un autre fichier. Néanmoins, les instructions de modifications des paquets peuvent s'avérer utiles, et elles sont inscrites dans le fichier `preproxy_users`.

<sup>45</sup> realm : préfixe ou suffixe assignant un utilisateur à une organisation.

```
DEFAULT Suffix == "@domaine.be"  
Service-Type = Framed-User,  
Framed-Protocol = PPP,  
Framed-IP-Address = 255.255.255.254,  
Framed-IP-Netmask = 255.255.255.0
```

### 5.4.5 Gestion de l'Accounting

Cette méthode a pour but de stocker diverses informations renvoyées par les NAS lors de l'établissement et à l'arrêt d'une connexion. Les données renvoyées sont entre autres : la durée de la session, le nombre d'octets envoyés et reçus, le nombre de canaux utilisés, la raison qui a mis fin à la session, etc. Les diverses opérations de comptabilisation seront stockées par le serveur RADIUS. Elles seront très utiles aux fournisseurs d'accès qui établiront grâce à cela les factures de leurs clients. Mais elles sont également utiles aux administrateurs réseaux qui peuvent ainsi déterminer le taux d'occupation de leur NAS ou le nombre de déconnexions (et leurs raisons). En ce sens, on peut dire que l'accounting est à RADIUS ce que les journaux d'événements sont aux systèmes d'exploitation.

Les options de configuration sont relativement minces, il faut décider où stocker les informations : dans un fichier ou dans une BD. Si la 2<sup>ème</sup> option est retenue, FreeRADIUS permet de scinder le stockage en 2 tables. La 1<sup>ère</sup> étant réservée aux paquets `Accounting-Start` et l'autre aux paquets de type `Accounting-Stop`. Cette option a pour but d'accélérer les requêtes de consultation des tables, notamment lors de l'opération de détection des sessions multiples. En effet, un autre intérêt de cette journalisation est de pouvoir répondre à la question : « qui est en ligne ? ». Par extension, on peut également déterminer combien de sessions parallèles a ouvertes un utilisateur (en utilisant par exemple deux terminaux différents). Le refus d'un nouvel accès à l'utilisateur parce qu'il a atteint le nombre maximum de sessions concurrentes peut être décidé lors des requêtes d'identification.

Enfin, le standard RADIUS définit que les requêtes de type `accounting` peuvent être transférées tout comme les requêtes d'identifications. En effet, l'utilisateur validé par un serveur AAA externe sera plus que probablement facturé par l'organisation qui détient le dit serveur. Il est donc logique que les paquets d'accounting suivent le même chemin que les paquets de demandes d'identifications et d'autorisations.

### 5.4.6 Configuration des serveur d'accès

Le fichier `client.conf` stocke les NAS de manière simple. La commande `client` suivie de l'adresse de celui-ci (ou de son FQDN<sup>46</sup>) et les attributs de cette commande sont triviaux. Il s'agit du secret, d'un nom distinctif propre à FreeRADIUS et du type de NAS.

```
client 10.93.11.2 {
    secret = testing123
    shortname = ap1200_3
    nastype= cisco
}
```

Les dernières versions de FreeRADIUS intègrent la gestion des NAS également via une base de données. Les champs de la table `NAS` sont en tous points similaires aux attributs du fichier de configuration. L'avantage de cette méthode est la gestion des serveurs d'accès par le biais de l'interface graphique. Néanmoins, pour l'instant, l'ajout ou la modification d'une ligne de cette table ne sera prise en compte qu'après le redémarrage du service (tout comme pour une modification dans le fichier `clients.conf`).

### 5.4.7 Interface graphique d'administration

Une interface graphique permettant d'administrer les fonctions de bases du serveur est fournie avec les sources. A condition que vos utilisateurs soient stockés dans une base de données ou un annuaire LDAP, elle permet de :

- Gérer les utilisateurs et leurs droits.
- Gérer les groupes d'utilisateurs et leurs droits.
- Gérer les serveurs d'accès.
- Visualiser les données de comptabilisations.
- Générer des rapports d'utilisation des services.
- Tester le serveur (via un test d'identification).

Cette interface est écrite en PHP<sup>47</sup> et est donc destinée à une utilisation en mode WEB. Malheureusement, les développeurs de cette dernière l'ont pensée de telle manière qu'elle doit préférablement être installée sur le même système qui héberge le serveur radius (qui n'est pas forcément aussi un serveur WEB). En effet, le programme doit avoir accès à certains fichiers du serveur radius pour pouvoir être exploité

---

<sup>46</sup> FQDN : Fully Qualified Domain Name (hostname + domain name)

<sup>47</sup> PHP : Hypertext Preprocessor (<http://www.php.net/>)

totalément ; la configuration de partage NFS<sup>48</sup> ou SAMBA est donc obligatoire si l'on utilise des systèmes différents.

### 5.4.8 Tester sa configuration

Avant de déclarer la configuration du serveur comme étant achevée, il convient de l'évaluer au mieux. Cette tâche, plus que nécessaire, permettra de corriger les divers problèmes qui ne manqueront pas de se présenter.

Un principe des plus simples pour comprendre les dysfonctionnements d'un système dont la fonction principale est de communiquer, est d'intercepter ses communications. Des analyseurs de trafic (tel qu'Etherreal<sup>49</sup>) permettent de réaliser cette tâche. On peut, grâce à ces logiciels, capturer les paquets entrant et sortant liés à un processus (par exemple le daemon radius). Dans la pratique, même si cette méthode porte ses fruits, elle s'avère très peu pratique et souvent très fastidieuse. Fort heureusement, d'autres méthodes sont disponibles : il existe des programmes permettant d'émuler des NAS et les bons serveurs radius peuvent être démarrés en mode « bavard » (verbose, en anglais).

FreeRADIUS peut être démarré en mode debug, il renvoie alors un nombre conséquent (mais très utile) de messages sur la sortie standard. Pour ce faire, il suffit d'ajouter l'argument "-X" sur la ligne de commande. Au démarrage, des informations sur la configuration et l'instanciation des modules sont affichées (cf. annexe 2). Ensuite, le contenu des paquets et les actions entreprises pour les traiter sont retournés à chaque requête. Voici l'exemple de 2 requêtes accounting, concernant l'utilisateur toto :

Le NAS signale le démarrage d'une session pour toto :

```
rad_recv: Accounting-Request packet from host 127.0.0.1:1701, id=6, length=43
  User-Name = "toto"
  Acct-Status-Type = Start
  Acct-Session-Id = "812"
  NAS-Port = 0
  Processing the preacct section of radiusd.conf
modcall: entering group preacct for request 0
  modcall[preacct]: module "preprocess" returns noop for request 0
  rlm_acct_unique: Hashing 'NAS-Port = 0,Client-IP-Address = 127.0.0.1,NAS-IP-Address = 127.0.0.1,Acct-Session-Id = "812",User-Name = "toto"'
  rlm_acct_unique: Acct-Unique-Session-ID = "ac5efbe5b6adeb23".
  modcall[preacct]: module "acct_unique" returns ok for request 0
    rlm_realm: No '@' in User-Name = "toto", looking up realm NULL
    rlm_realm: No such realm "NULL"
  modcall[preacct]: module "suffix" returns noop for request 0
  modcall[preacct]: module "files" returns noop for request 0
modcall: group preacct returns ok for request 0
  Processing the accounting section of radiusd.conf
modcall: group accounting returns ok for request 0
```

---

<sup>48</sup> NFS : Network File System

<sup>49</sup> ethereal : sous licence GNU (<http://www.ethereal.com/>)

```
Sending Accounting-Response of id 6 to 127.0.0.1:1701
Finished request 0
```

Le NAS signale l'arrêt de la session. On peut également y lire les informations d'accouting renvoyées.

```
rad_recv: Accounting-Request packet from host 127.0.0.1:1706, id=11, length=67
  User-Name = "toto"
  Acct-Status-Type = Stop
  Acct-Session-Id = "812"
  NAS-Port = 0
  Acct-Input-Octets = 14265
  Acct-Output-Octets = 1984357
  Acct-Session-Time = 1247
  Acct-Terminate-Cause = User-Request
  Processing the preacct section of radiusd.conf
  modcall: entering group preacct for request 5
    modcall[preacct]: module "preprocess" returns noop for request 5
    rlm_acct_unique: Hashing 'NAS-Port = 0,Client-IP-Address = 127.0.0.1,NAS-IP-Address =
127.0.0.1,Acct-Session-Id = "812",User-Name = "toto"'
    rlm_acct_unique: Acct-Unique-Session-ID = "ac5efbe5b6adeb23".
    modcall[preacct]: module "acct_unique" returns ok for request 5
      rlm_realm: No '@' in User-Name = "toto", looking up realm NULL
      rlm_realm: No such realm "NULL"
    modcall[preacct]: module "suffix" returns noop for request 5
    modcall[preacct]: module "files" returns noop for request 5
  modcall: group preacct returns ok for request 5
  Processing the accounting section of radiusd.conf
  modcall: entering group accounting for request 5
  modcall: group accounting returns ok for request 5
  Sending Accounting-Response of id 11 to 127.0.0.1:1706
  Finished request 5
```

Pour ce qui est des opérations de décision, on peut par exemple déceler que, l'utilisateur n'ayant pas de realm, le serveur a décidé de ne pas proxyier la requête.

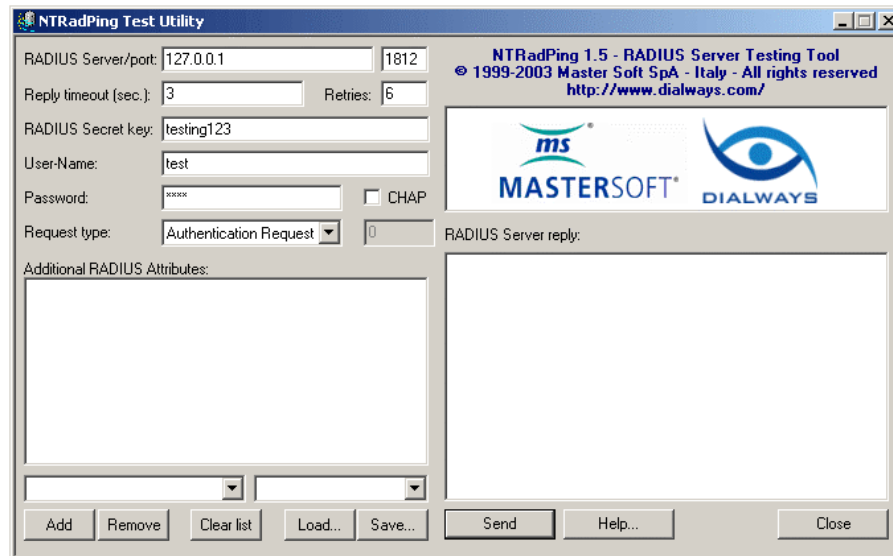
Afin de pouvoir vérifier le bon comportement du serveur, nous allons nous placer maintenant « côté client ». Le binaire `radclient` fourni en standard avec FreeRADIUS simule un NAS. Il a besoin des arguments suivants sur la ligne de commande pour fonctionner :

- le répertoire stockant le dictionnaire des attributs RADIUS,
- l'adresse IP du serveur RADIUS,
- le type de paquet envoyé aux serveurs,
- le secret partagé avec ce serveur.

Ensuite, l'outil lit sur l'entrée standard une série de paires A/V et envoie la requête après avoir reçu le caractère de contrôle EOF<sup>50</sup>. La réponse à cette requête est finalement renvoyée sur la sortie standard.

---

<sup>50</sup> EOF : End Of File



Un autre outil, graphique cette fois, aux fonctionnalités totalement équivalentes est également disponible (gratuitement). Il se nomme NTRadPing<sup>51</sup> et n'existe que pour Windows. Très simple d'utilisation, il se compose d'une simple fenêtre (voir ci-dessus) de laquelle on accède à tous les paramètres. Au côté de l'exécutable, on trouve un fichier texte reprenant le dictionnaire, ce dernier pouvant être modifié.

---

<sup>51</sup> NTRadPing : <http://www.mastersoft-group.com/download/>

## 6 Plan de travail

---

### 6.1 Environnement

Je suis employé au sein d'une institution parastatale de la région wallonne. Cette dernière a pour nom "Agence Wallonne des Télécommunication" (AWT). Sa mission principale est de promouvoir l'utilisation des TIC (Technologies de l'Information et de la Communication) en région wallonne. Il s'agit d'une petite structure d'une trentaine d'employés, relativement comparable à celle d'une PME.

Mon rôle est d'ordinaire désigné sous l'appellation « d'administrateur système ». Je suis chargé de l'achat, de la configuration, de l'installation et de l'entretien du parc informatique. Cela va de l'installation d'une station de travail à la configuration d'un serveur.

La caractéristique principale de l'agence étant de promouvoir les TIC, elle se doit de ne privilégier aucun constructeur, aussi bien dans le choix du matériel que dans le choix des logiciels. De plus, étant en relation avec des entreprises aussi nombreuses que diverses, l'agence tente d'adopter des solutions qui reflètent celles choisies par ce type de structure. Les choix qui ont été faits dans certaines implémentations peuvent dès lors paraître démesurés, mais ils garantissent notre crédibilité auprès de ces entreprises.

A contrario, là où une petite PME choisira le plus souvent d'implémenter une solution totalement basée autour d'un même constructeur logiciel afin de minimiser ses coûts de formations et de configurations, nous essayons (dans la mesure du possible) de garder un regard neutre ne privilégiant pas une technologie au détriment d'une autre par facilité ou gain de temps (voire d'argent). Ainsi à l'agence, les technologies employées sont très diverses : Microsoft côtoie UNIX/Linux et ceux-ci sont interconnectés par réseaux faits à l'essentiel de matériel Cisco.

Dans la même veine d'idées, les choix en matière de logiciels peuvent paraître quelque peu atypiques. En effet, une de nos prérogatives, qui est d'avoir l'idée la plus concrète possible de l'utilisation des technologies informatiques et d'être également une vitrine de cette technologie, nous amène souvent à faire la démonstration que les technologies libres et propriétaires peuvent cohabiter sans difficulté. Cette idée nous a poussés dans la voie de l'open source pour la concrétisation de notre infrastructure RADIUS.

## 6.2 Rationaliser la gestion des utilisateurs

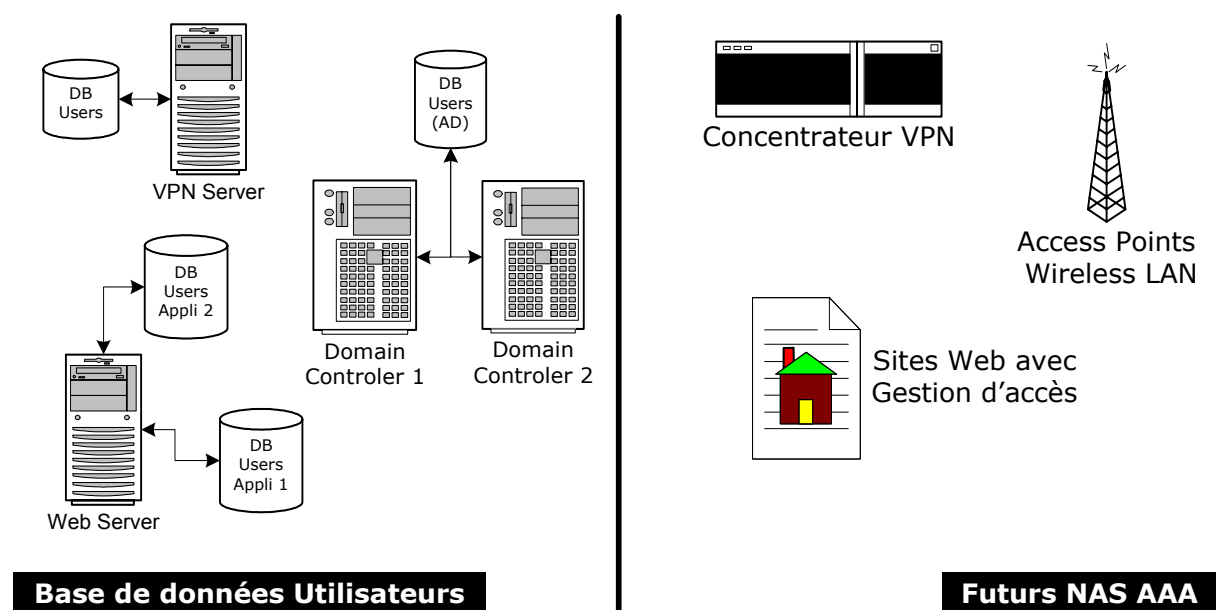
Les utilisateurs de nos systèmes sont d'origines diverses. Il s'agit de membres du personnel, de membres du conseil d'administration, de partenaires ainsi que de diverses autres cellules externes (dont, par exemple, un comité scientifique). De plus, ces deux dernières années, notre institution a vu le nombre de ses effectifs presque doubler suite à diverses restructurations. En regard de tout cela, on comprend aisément qu'une gestion structurée des utilisateurs soit devenue une des priorités.

Jusqu'alors, chaque application organisait la gestion de ses utilisateurs, de leur mot de passe et de leurs droits d'accès de manière indépendante. L'objectif est donc d'essayer de centraliser cette gestion au mieux, c'est-à-dire en tenant compte des réalités du terrain. Un premier pas vers cette rationalisation est la mise en service et la configuration d'un serveur RADIUS, objet de ce travail.

Le choix du protocole RADIUS et d'une architecture de type AAA est un besoin à l'origine. En effet, sans vouloir faire l'historique complète de la société qui m'emploie, un déménagement nous a amenés à effectuer un nouveau design pour notre réseau informatique. Dans ce design apparaissaient deux éléments intimement liés à RADIUS : un concentrateur VPN et quatre bornes d'accès sans fil. La nécessité de s'intéresser à RADIUS devenait incontournable.

## 6.3 Description de l'existant et plan d'évolution

### 6.3.1 Présentations des éléments



L'illustration ci-dessus présente à gauche les différents services que nous avons et à droite les nouveaux que nous souhaitons intégrer :

- Le concentrateur vient remplacer le serveur qui s'occupait de l'accès VPN,
- les services WEB ne vont plus gérer leurs utilisateurs de manière indépendante,
- seules les bornes Wifi sont un ajout.

Toute cette infrastructure va communiquer avec le même serveur RADIUS qui devra être capable d'identifier tous les utilisateurs et d'éventuellement loguer les accès effectués par ceux-ci.

En ce qui concerne le VPN, qui est réservé aux employés, le client utilisé pour s'y connecter est celui intégré à Windows. Le protocole choisi pour créer le tunnel est PPTP<sup>52</sup> et l'ouverture de la session est assurée par MS-CHAP V2. Les identifiants des utilisateurs du VPN<sup>53</sup> ne sont pas ceux stockés dans Active Directory car les règles de sécurité relatives aux mots de passes ont été jugées insuffisantes. Les employés conserveront donc un login / mot de passe différent pour leur accès VPN.

Pour ce qui est des sites WEB, l'idée est de proposer à court terme un extranet basé cette fois sur les identifiants utilisateurs stockés dans Active Directory (pour les employés) et également sur ceux stockés dans la BD (pour les membres des divers conseils).

Quant à la mise en place du réseau sans fil, elle sera sécurisée par une authentification EAP-TLS basée sur une infrastructure PKI<sup>54</sup> et un chiffrement basé sur WPA<sup>55</sup>. La raison de ce choix pour la sécurisation du réseau WiFi a été motivée par le fait que les postes clients (Windows XP) supportent d'origine la méthode EAP-TLS.

### 6.3.2 Description du réseau

Le réseau interne se compose de 2 segments en aval du pare-feu : d'une part un segment DMZ<sup>56</sup> sur lequel on trouve, entre autres, nos serveurs d'application web de production. D'autre part un segment LAN contenant principalement les serveurs de bases de données, les serveurs de développements et un domaine Windows avec ses serveurs et ses postes clients. En amont de ce pare-feu, on trouve le routeur d'accès vers

---

<sup>52</sup> PPTP : Point-to-Point Tunnelling Protocol

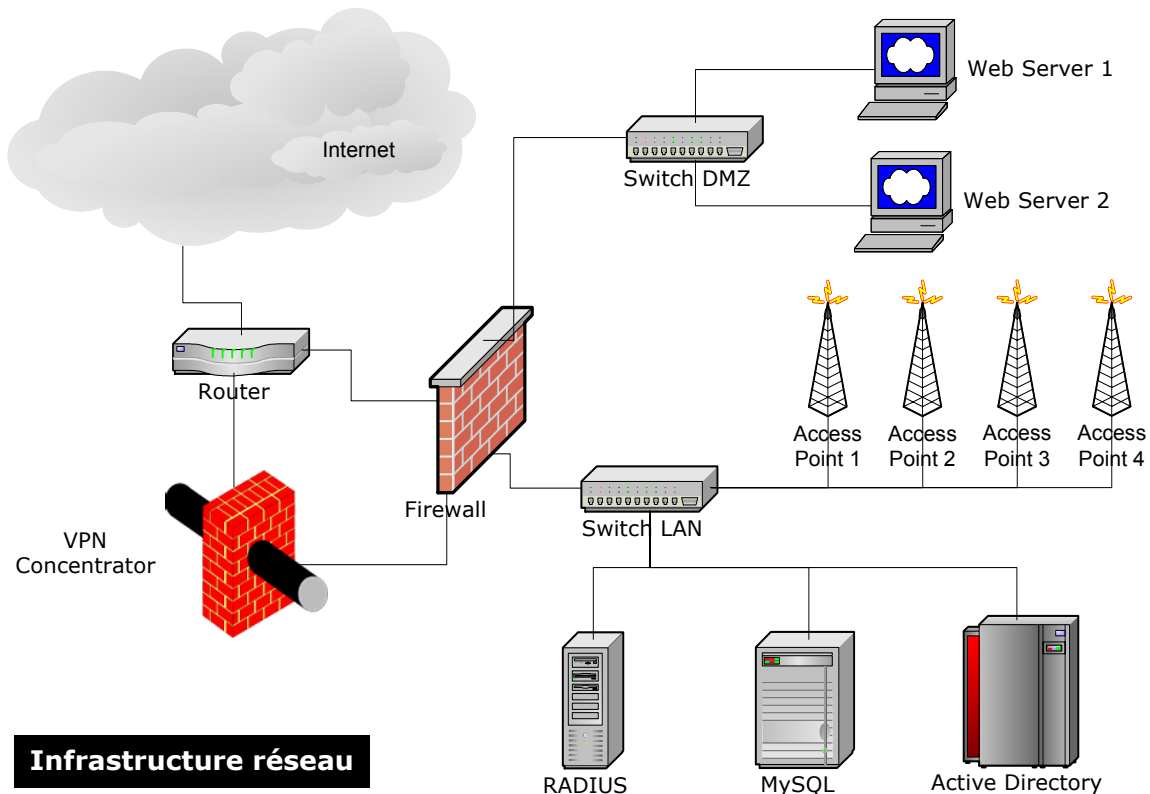
<sup>53</sup> VPN : Virtual Private Network

<sup>54</sup> PKI : Public Key Infrastructure

<sup>55</sup> WPA : Wi-Fi Protected Access

<sup>56</sup> DMZ : De-Militarized Zone.

Internet et le concentrateur VPN. Le schéma ci-dessous est une représentation graphique ayant pour but de mieux situer les différents équipements devant communiquer au sein de ce réseau pour ce projet.



En réalité, il existe un 3<sup>ème</sup> segment commun à toute l'infrastructure. Les équipements sont compatibles avec la norme IEEE<sup>57</sup> 802.1q ou, plus simplement, permettent l'emploi de VLAN<sup>58</sup>. Cette technique qui se trouve au niveau 2 du modèle OSI, permet de faire coexister des réseaux différents sur la même couche physique en les numérotant. Ce 3<sup>ème</sup> segment est donc affecté à VLAN baptisé « VLAN de management » et possède son propre segment d'adresses IP.

Les switches garantissent l'isolation entre différents VLAN. Chacun de leurs ports pouvant être affecté à un VLAN particulier. Un mode spécial appelé *trunking* place un port dans tous les VLAN. Les trunks sont très utiles notamment pour relier les switches entre eux. De plus, les cartes réseaux modernes peuvent (avec un driver adapté) être connectées sur un port en mode trunk. Ainsi, certaines machines seront reliées à plusieurs segments (c'est le cas des serveurs WEB).

Pour être complet, il faut ajouter que le concentrateur VPN n'a pas de support pour les VLAN. Il ne peut donc pas être affecté à un port en

<sup>57</sup> IEEE : Institute of Electrical and Electronics Engineers

<sup>58</sup> VLAN : Virtual LAN

mode trunking. Pour palier ce manque, le firewall est utilisé comme routeur entre le VLAN de management et l'interface privée du concentrateur.

L'emploi de cette technique permet de restreindre la surface d'attaque. Dans le cas présent, nous allons confiner les communications RADIUS au sein du VLAN de management. Un utilisateur du réseau interne n'aura ainsi pas accès au serveur radius, le daemon étant uniquement lié au segment de management.

### 6.3.3 Périphérie du serveur RADIUS

Comme stipulé précédemment, il existe diverses bases de données utilisateurs dont la plus importante (celle des utilisateurs internes) est stockée dans Active Directory. Dans ce cas, la plupart des administrateurs systèmes décideraient de consolider cette base et y centraliseraient le maximum d'informations. La démarche a été tout autre.

Se passer d'Active Directory était un choix peu envisageable. Nous venions de terminer l'installation d'une solution de courrier électronique et de travail collaboratif basée sur Microsoft Exchange (nos utilisateurs ayant l'habitude d'utiliser Microsoft Outlook et des PDA<sup>59</sup> avec un OS Microsoft). Cette solution étant intimement liée à AD, faire marche arrière était impossible.

Un choix de raison a donc été fait : celui d'avoir, non pas une, mais deux bases de données d'utilisateurs. Les membres du personnel restant gérés par l'infrastructure Windows déjà en place et tous les autres étant regroupés dans une base de donnée MySQL.

Le dernier problème à résoudre est de déterminer le chemin par lequel FreeRADIUS va accéder à la DB utilisateur de AD. Trois choix possibles :

- Utiliser NTLM et une simple authentification basée sur MS-CHAP. (Cette technique est abordée au point 5.4.4.4)
- Utiliser le connecteur LDAP d'Active Directory. Qui permet à la fois l'authentification et l'autorisation, mais oblige l'utilisation du protocole PAP (voir le point 5.4.4.3)
- Installer IAS et utiliser FreeRADIUS comme proxy.

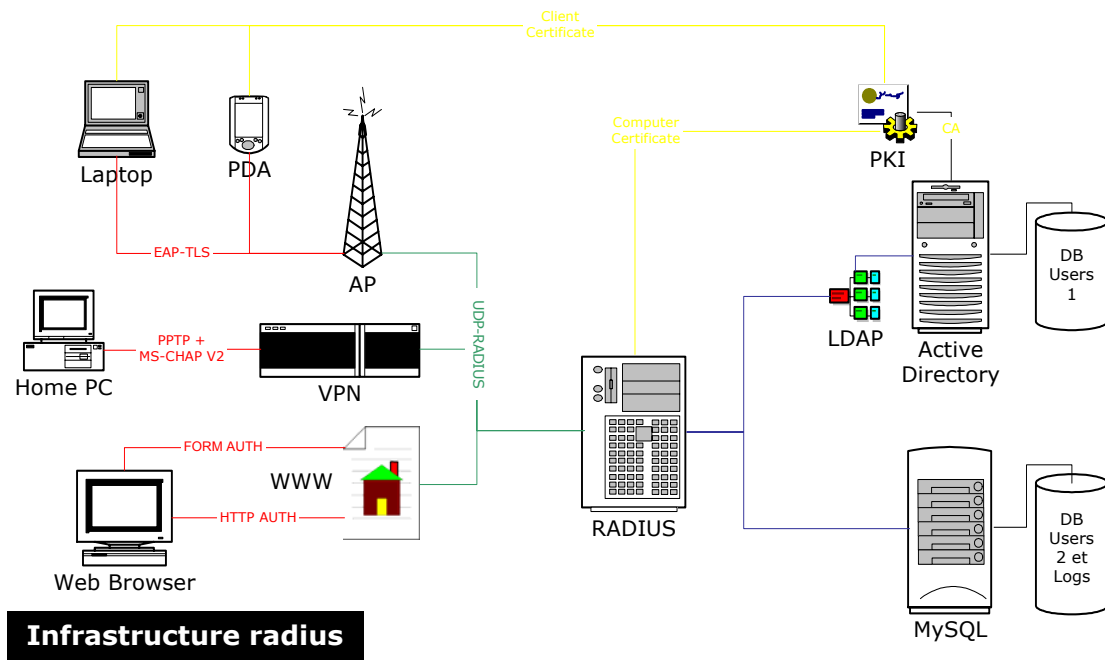
C'est la 2<sup>ème</sup> solution qui a été retenue assumant le fait que l'on ne souhaitait pas utiliser IAS et que l'utilisation de LDAP n'imposait aucune

---

<sup>59</sup> PDA : Personal Digital Assistant (Agenda électronique de poche)

modification sur les serveurs Microsoft. La contrainte de l'utilisation de PAP ne posait pas de problèmes, les paquets étant restreints au VLAN de management car les applications employées par ces utilisateurs sont de type WEB.

Le schéma ci-après clôture cette partie. Il est organisé autour du serveur radius, on y retrouve tous les intervenants et les protocoles utilisés. Les traits de couleurs caractérisent les protocoles : en rouge le type d'authentification entre le client final et le NAS, en vert les paquets radius, en jaune les échanges de certificats et en bleu l'accès vers les bases de données.



## 7 Mise en oeuvre

---

### 7.1 Configuration du serveur

#### 7.1.1 Installation

L'installation est déjà abordée au point 5.4.1. En voici un bref résumé. Le système est un Debian Woody (version réputée stable). L'intérêt de cette distribution est la possibilité de téléchargement et l'installation de package en 1 seule et simple commande : `apt-get install <nom du package>`. Lorsque les packages dépendent d'autres librairies, la procédure d'installation demande à l'utilisateur s'il souhaite également les installer.

Après avoir installé correctement le serveur et ajouté les deux modules nécessaires aux connexions vers LDAP et MySQL, nous pouvons maintenant entrer dans le vif du sujet. Tous les fichiers de configuration se trouvant dans le répertoire `/etc/freeradius`<sup>60</sup>. Ils sont abondamment commentés, ce qui est un élément très précieux lorsqu'on débute.

#### 7.1.2 Configuration générale :

La configuration générale se trouve dans le fichier `radiusd.conf`. Commencer par vérifier le contexte utilisateur sous lequel le serveur va s'exécuter. Moins le profil de l'utilisateur aura de droits, plus le serveur sera considéré comme sécurisé. En effet, si une faille venait à être découverte dans la version de FreeRADIUS employée, une personne malveillante pourrait obtenir les droits sous lesquels le daemon s'exécute. Les dégâts pourraient être dramatiques si c'est le contexte `root` qui est employé. Debian crée par défaut un utilisateur `freerad` qui est chargé d'exécuter le daemon.

De plus, les fichiers de configuration de FreeRADIUS contiennent des informations sensibles. La seule protection de ses données repose sur les droits de lecture et d'écriture du système de fichier. Il est donc nécessaire d'employer un système d'exploitation stable et sécurisé.

On paramètre ensuite l'adresse et le port d'écoute du serveur. Dans notre cas, nous avons utilisé le segment IP de management. Avant de passer aux modules, la configuration de la disponibilité du serveur

---

<sup>60</sup> Cet emplacement est propre à Debian, sur tous les autres système le répertoire par défaut est `/etc/raddb`.

s'effectue au niveau de la section thread ; le serveur n'ayant pas à répondre à beaucoup de requêtes, on laisse la valeur par défaut de 5.

Pour mener à bien la configuration, nous n'allons activer que les modules dont nous avons besoin :

- pap : pour l'authentification simple login / mot de passe,
- mschap : pour nos clients VPN,
- acct\_unique : pour identifier les sessions concurrentes,
- expr : pour l'emploi d'expression régulière dans la configuration,
- ldap : pour les utilisateurs stockés dans AD,
- sql\_rlm\_mysql : pour les utilisateurs stockés dans MySQL.

Comme cette configuration ne prévoit pas de proxying, toutes les lignes relatives à cette fonction sont ignorées.

A la fin du fichier `radiusd.conf`, on trouve une série d'instructions qui paramètrent le comportement du serveur :

```
instantiate {
    expr
}

authorize {
    mschap
    eap
    ldap
    sql
}

authenticate {
    Auth-Type PAP {
        pap
    }
    Auth-Type MS-CHAP {
        mschap
    }
    Auth-Type LDAP {
        ldap
    }
    eap
}

preacct {
    acct_unique
}

accounting {
    sql
}

session {
    sql
}

post-auth {
    sql
}
```

Instancie le module d'expression régulière.

Les deux premières commandes ont pour effet d'activer le basculement automatique du type d'authentification (par défaut PAP). Les deux suivantes lui commandent d'aller valider les utilisateurs en utilisant le module ldap et le module mysql.

Définit les types d'authentifications admises.

Le type LDAP indique que l'utilisateur sera validé si FreeRADIUS peut se connecter au serveur LDAP en utilisant le login et le mot de passe transmis dans une requête `Access-Request` en PAP (voir le dernier § du point 5.4.4.3 à ce sujet).

Lors des requêtes `Accounting Start` et `Accounting Stop`, ce module va générer un hash qui identifiera une session de manière unique.

Commande au serveur d'utiliser la base de données pour les opérations d'accounting.

FreeRADIUS utilisera la table d'accounting pour détecter les sessions multiples.

Les requêtes `Access-Request` seront loguées dans la table `radpostauth`.

### 7.1.3 Configuration du module LDAP

Toujours dans le fichier `radiusd.conf`, on trouve les données relatives à la configuration de la connexion au serveur LDAP :

```
ldap {
    server = "mondad"
    identity = "cn=ldapsearch, cn=Users, dc=local"
    password = dumbpasswd
    start_tls = no
    basedn = "cn=Users, dc=local"
    filter = "(samaccountname=%{user-name})"
    ldap_connections_number = 2
    access_attr_used_for_allow = no
}
```

Les paramètres ci-dessus suffisent pour la configuration, LDAP n'étant employé que pour l'authentification.

Les variables `identity` et `password` sont utilisées pour binder le serveur. L'option `start_tls` permet le chiffrement de la communication vers le serveur LDAP (incompatible avec AD). Le `basedn` représente le nœud sous lequel la recherche est effectuée (par défaut dans Active Directory, les utilisateurs sont situés dans le conteneur "Users"). Le filtre va canaliser la recherche sur le login de l'utilisateur (dans AD cet attribut est "samaccountname"). Le paramètre suivant définit le nombre de connexions simultanées vers le serveur d'annuaire.

La dernière option, `access_attr_used_for_allow` doit être désactivée pour notre installation. Dans le cas contraire (cas par défaut), elle utilise le paramètre `access_attr` dont la valeur doit être booléenne. `access_attr` contient donc un nom d'attribut (pour AD il s'agit de "msnppallowdialin"), si la valeur de l'attribut est VRAI : l'accès est autorisé, dans le cas contraire c'est une fin de non recevoir qui est renvoyée.

### 7.1.4 Configuration du module MySQL

Comme précisé au point 5.4.4.2, le script de création des tables a été exécuté à l'aide d'un client MySQL. Le schéma est conforme à l'annexe 1. Le nom de la base est "Radius". Par sécurité, un utilisateur interne à MySQL et dédié à cette connexion a été créé (le scope d'action de l'utilisateur est donc restreint à la DB radius). Le fichier `sql.conf` reprend ses différents paramètres.

Deux autres points à souligner : l'utilisation de la base de données pour stocker la liste des clients (en lieu et place du fichier `clients.conf`) nécessite de décommenter la ligne `readclients = yes`. D'autre part, la table `radpostauth` d'origine prévoit le stockage du mot de passe (ce qui n'est pas très intéressant et peu s'avérer dangereux) et n'a pas de champ relatif à l'identification du NAS et du client. La structure de la table a été modifiée pour pallier ces petits problèmes ainsi que la requête d'insertion.

## 7.1.5 Interaction avec les NAS

### 7.1.5.1 Concentrateur VPN

Le but premier de cet équipement est de prendre en charge le chiffrement nécessaire à la sécurisation des tunnels. Côté matériel, il s'agit d'un VPN Concentrator 3005 de chez Cisco. Sa configuration a été réalisée via son interface web de gestion. Elle se résume en deux points : ajouts dans la section AAA/Authenticate des paramètres nécessaires à la connexion au serveur RADIUS (adresse IP, ports et secrets partagés) ; même démarche dans la section AAA/Accounting.

Comme précisé auparavant, les utilisateurs de ce NAS sont enregistrés dans la DB MySQL dont voici le détail des données :

**radcheck**

<b>Id</b>	<b>UserName</b>	<b>Attribute</b>	<b>op</b>	<b>Value</b>
1	pptpuser1	User-Password	==	MotDePasseComplexeA
2	pptpuser2	User-Password	==	MotDePasseComplexeB
3	pptpuser3	User-Password	==	MotDePasseComplexeC

En se référant à l'ordinogramme du point 5.4.4.2 : on constate que c'est cette table qui va recevoir la 1<sup>ère</sup> requête. FreeRADIUS va donc vérifier si une (ou plusieurs) ligne(s) de cette table concerne(nt) le nom d'utilisateur transmis. Dans l'affirmative, il vérifie chaque paire A/V et les valide avant de poursuivre.

Dans cette configuration, aucune donnée propre à un seul utilisateur ne doit être renvoyée. Si tel devait être le cas (par exemple pour affecter une IP déterminée), cela se ferait par le biais de la table `radreply`.

**usergroup**

<b>Id</b>	<b>UserName</b>	<b>GroupName</b>
1	pptpuser1	VPN
2	pptpuser2	VPN
3	pptpuser3	VPN

FreeRADIUS vérifie ensuite l'appartenance de l'utilisateur à un groupe et si c'est le cas, poursuit son traitement.

### radcheck

<b>Id</b>	<b>GroupName</b>	<b>Attribute</b>	<b>op</b>	<b>Value</b>
1	VPN	Prefix	==	pptp
2	VPN	NAS-Port-Type	==	Virtual
3	VPN	NAS-IP-Address	==	<IP du Concentrateur>

Cette table est la réplique de la 1<sup>ère</sup> table à ceci près que le premier champ concerne les groupes et non plus les utilisateurs. Le même traitement est effectué. Dans le cas présent, les login devront commencer par "pptp", le type de port (renvoyé par le NAS) devra être virtuel et l'adresse IP du concentrateur sera également un élément de contrôle.

### radgroupreply

<b>Id</b>	<b>GroupName</b>	<b>Attribute</b>	<b>op</b>	<b>Value</b>
1	VPN	Framed-Protocol	=	PPP
2	VPN	Service-Type	=	Framed-User
3	VPN	Framed-IP-Address	=	255.255.255.224
4	VPN	Framed-Compression	=	Van-Jacobson-TCP-IP

Pour finir, le paquet retourné au concentrateur reprendra les paires A/V de la table radgroupreply. Les instructions des 2 dernières lignes intimement au concentrateur d'affecter l'IP de son choix et d'activer la compression des entêtes.

## 7.1.5.2 Point d'accès sans fil

### Installation de la PKI

Avant de pouvoir implémenter cette partie, il a fallu déployer une infrastructure PKI mettant en œuvre une CA<sup>61</sup>. Le choix d'utiliser la PKI de Microsoft (et d'ainsi créer notre propre CA) a été fait pour diverses raisons dont les principales sont :

- L'intégration et l'installation rapide de cette PKI sur le domaine existant (qui est l'argument massue, souvent avancé en faveur de Microsoft).
- La création assistée des certificats par le biais d'une interface WEB (nécessitant une identification préalable)
- Les postes clients étant intégrés au domaine, l'utilisation des GPO<sup>62</sup> permet d'installer les certificats sans qu'une intervention de l'utilisateur ne soit requise.
- Pas de frais de licence supplémentaires.

---

<sup>61</sup> CA : Certification Authority

<sup>62</sup> GPO : Group Policy Object

Un des contrôleurs de domaines Windows s'est vu ajouter le rôle de CA. Il va donc servir à la signature des divers certificats. Le 1<sup>er</sup> certificat créé est celui de la CA validé par elle-même.

Les certificats clients sont créés via l'interface WEB proposée par Microsoft. Celle-ci est restreinte à une utilisation en interne pour des raisons évidentes de sécurité. Cette interface installe un ActiveX qui se charge de générer la clé secrète, la clé publique et la demande de certificat. Cette demande est alors transférée à la CA qui appose sa signature et renvoie le certificat validé (au passage, elle prend soin de stocker ce certificat dans l'annuaire Active Directory). Ce processus est totalement transparent pour l'utilisateur. Ce dernier ne fait que cliquer sur deux liens : « Request a certificate » et « user certificate ».

Bien que cette méthode puisse faire horreur à un expert en sécurité averti, le choix de l'emploi de cette interface WEB est un gain de temps précieux pour l'administrateur système.

Il faut maintenant générer un certificat pour le serveur RADIUS. Comme notre serveur ne possède pas d'interface graphique, nous sommes contraints de passer par la génération manuelle de la clé secrète et de la demande de certificat. Pour générer ces deux éléments, l'outil OpenSSL<sup>63</sup> s'avère être une aide précieuse. Deux commandes suffisent à créer les fichiers :

- openssl genrsa 1024 : va générer une clé privée non chiffrée.
- openssl req -new -key <chemin vers la clé privée> : va produire une demande de certificat. Après avoir répondu à plusieurs questions (paramétrées par le fichier openssl.cnf), on obtient la précieuse demande.

Le service web de la CA contient une page dédiée à l'envoi de *Certificate Request*. Après avoir sélectionné un canevas (dans le cas qui nous occupe il s'agit d'un canevas serveur), on peut poster le fichier généré par OpenSSL. Et obtient en retour un certificat dûment validé par la CA. Finalement, les trois éléments nécessaires à la bonne configuration de notre serveur radius sont réunis :

- La clé privée du serveur (ci-après : `privkey.pem`),
- Le certificat du serveur (ci-après : `radius.crt`),
- Le certificat de la CA (ci-après : `ca.crt`).

---

<sup>63</sup> OpenSSL Project : <http://www.openssl.org/>

## Configuration de FreeRADIUS

La configuration suivante suffit à paramétrer FreeRADIUS pour qu'il accepte les authentifications basées sur EAP :

```
eap {
  default_eap_type = tls
  timer_expire = 60
  ignore_unknown_eap_types = no
  tls {
    #private_key_password = serversecret
    private_key_file = ${certsdir}/privkey.pem
    certificate_file = ${certsdir}/radius.crt
    CA_file = ${certsdir}/ca.crt
    dh_file = ${certsdir}/dh
    random_file = ${certsdir}/random
    check_crl = yes
  }
}
```

Le paramètre `private_key_password` est commenté car notre clé privée n'est pas encapsulée par un mot de passe. Le flag `check_crl` est placé à `yes` afin que FreeRADIUS consulte la CRL (qui peut être récupérée via LDAP ou une simple requête HTTP). L'argument `random_file` est un fichier aléatoire de 1024 octets qui peut être généré par l'instruction : `openssl rand 1024`. L'argument `dh_file` est un fichier de paramètres Diffie-Hellman qui peut être généré à l'aide de la commande `openssl gendh`.

## Configuration des Access Point

Les NAS employés sont des Cisco Aironet 1200. La configuration a été effectuée par le biais de l'interface WEB. Les opérations à effectuer sont les suivantes :

- Définir le serveur RADIUS (adresse IP, ports et secret partagé)
- Créer un SSID<sup>64</sup> qui requiert une authentification de type EAP et pour lequel la gestion des clés de chiffrement est assurée par le protocole WPA.
- Activer la gestion du chiffrement via TKIP<sup>65</sup> (et éventuellement AES<sup>66</sup> CCMP<sup>67</sup> si les périphériques supportent la norme WPA2).

## Configuration des Clients

Les *supplicants* (clients du réseau sans fil) doivent d'abord créer leur clé privée et obtenir leur certificat validé par la CA (voir ci-dessus). Ils doivent également accepter la CA en tant que tiers de confiance en

---

<sup>64</sup> SSID (Service Set Identifier) : nom d'un réseau sans-fil.

<sup>65</sup> TKIP : Temporal Key Integrity Protocol

<sup>66</sup> AES : Advanced Encryption Standard

<sup>67</sup> CCMP : Counter-Mode/CBC-Mac Protocol

important son certificat. Ces deux opérations sont rendues entièrement automatiques par l'utilisation de l'interface WEB de la CA Microsoft. Il est bien évident que pour avoir accès à ce site web, les supplicants se connectent par le biais du réseau filaire.

Une fois la création et l'import des éléments nécessaires à la PKI effectués, le client n'a plus qu'à lancer une détection des réseaux sans fil et se connecte au SSID précédemment créé.

### **7.1.5.3 Authentification en mode WEB**

#### Apache et Auth mod radius

Le projet FreeRADIUS est également à l'origine d'un module<sup>68</sup> Apache<sup>69</sup> permettant d'utiliser RADIUS comme base de données d'utilisateurs.

Pour pouvoir employer ce module, il faut tout d'abord l'intégrer à Apache. Selon le type d'installation préférée, on emploiera la compilation ou les binaires qui sont également directement disponibles auprès des fournisseurs de distributions.

Après avoir terminé l'installation du module, ajouter ces quelques lignes au fichier `httpd.conf` :

```
LoadModule radius_auth_module modules/mod_auth_radius.so
AddModule mod_auth_radius.c

<IfModule mod_auth_radius.c>
  AddRadiusAuth monserveur:1812 monsecret 5:3
  AddRadiusCookieValid 60
</IfModule>
```

L'ajout d'une authentification HTTP se fait ensuite soit directement dans `httpd.conf`, soit par le biais des fichier `.htaccess`. Dans les deux cas, les directives passées au serveur sont les mêmes :

```
AuthType Basic
AuthName "RADIUS based protection"
AuthAuthoritative off
AuthRadiusAuthoritative on
AuthRadiusActive On
require valid-user
```

---

<sup>68</sup> Disponible à l'adresse : [http://www.freeradius.org/mod\\_auth\\_radius/](http://www.freeradius.org/mod_auth_radius/)

<sup>69</sup> Apache est un serveur WEB très répandu : <http://httpd.apache.org/>

Si l'utilisation de Microsoft IIS est incontournable, un module d'authentification RADIUS payant est disponible pour ce serveur. Plus d'info à l'adresse : [http://www.tcpdata.com/radiis\\_overview.htm](http://www.tcpdata.com/radiis_overview.htm)

Ces paramètres définissent une authentification HTTP basique. Le mot de passe est envoyé en clair avec les en-têtes http (Le mode digest n'est pas supporté). Les lignes 3 à 5 donnent l'instruction au serveur d'utiliser RADIUS pour l'authentification des utilisateurs.

La particularité de ce module est qu'il a été écrit comme une *passerelle* entre le monde RADIUS et le monde HTTP. En effet, `mod_auth_radius` transforme le serveur WEB en NAS et le client de ce NAS est le navigateur de l'utilisateur. La seule authentification disponible est le mode *basic*, ce qui implique le passage en clair du mot de passe entre le navigateur et le serveur WEB ; ce dernier utilise alors la méthode PAP avec le serveur.

Néanmoins, il est amusant de constater que la documentation précise que l'on peut utiliser le mode CHAP. Il faut pour cela entrer uniquement son nom d'utilisateur et le serveur renvoie une réponse de type 401. Il faut ensuite rafraîchir la page, le login étant réaffiché avec le challenge CHAP. L'utilisateur n'a plus qu'à calculer la réponse et l'introduire dans le champ mot de passe... Bien évidemment, peu d'utilisateurs possèdent une machine à résoudre les challenges CHAP de cette manière !

La seule méthode viable étant l'emploi de mots de passe en clair, l'emploi de HTTPS est plus que vivement recommandé.

### PHP et l'extension RADIUS

Une autre possibilité pour utiliser RADIUS en mode web est possible moyennant l'emploi du langage PHP. Ce dernier propose un module (dans le cas de PHP on parle d'extension) qui se comporte comme un client RADIUS. Le comportement est différent de celui de `mod_auth_radius` car avec PHP, le client et le NAS fusionnent en une seule entité (représentée par le serveur WEB). Cette propriété lui permet d'utiliser toutes les méthodes d'authentification disponibles<sup>70</sup>.

A vrai dire, l'extension RADIUS de PHP doit être considérée comme un émulateur de NAS. Toutes les fonctions RADIUS sont disponibles (en ce compris les fonctions d'accounting). De plus PHP n'est pas lié à un serveur WEB ou à un OS particulier, ce qui rend son emploi quasi universel.

Le choix de l'identification des utilisateurs par un simple formulaire HTML est le plus souvent retenu avec PHP. L'emploi de scripts côté client (par ex. un JavaScript) en combinaison avec PHP s'exécutant côté

---

<sup>70</sup> Cela étant, EAP n'est pas supporté.

serveur, permet d'employer des schémas d'authentifications de type challenge / réponse sans difficultés<sup>71</sup>.

- Cette extension est basée sur la `libradius` de l'OS FreeBSD<sup>72</sup>.
- Cette extension est téléchargeable à l'adresse :  
<http://www.bretterklieber.com/php/>.
- La documentation complète des fonctions se trouve à l'adresse :  
<http://www.php.net/manual/fr/ref.radius.php>
- Une classe facilitant l'accès à l'extension est disponible à l'adresse : [http://pear.php.net/package/Auth\\_RADIUS/](http://pear.php.net/package/Auth_RADIUS/). Elle est fournie avec deux exemples fort intéressants (l'un pour l'accounting et l'autre pour l'authentification)...

### Modification d'une application existante

L'AWT dispose d'une application documentaire avec une gestion interne des utilisateurs. Cette application propose deux méthodes d'accès aux documents : un client propriétaire permettant la gestion des documents (dont l'authentification est basée sur Kerberos) et un client web permettant un accès en lecture aux documents avec une authentification par formulaire. Nous souhaitons sans trop de difficultés adapter le logiciel WEB pour que les mots de passe de nos utilisateurs soient synchronisés avec l'application.

Pour mener cette idée à bien, nous avons procédé comme suit :

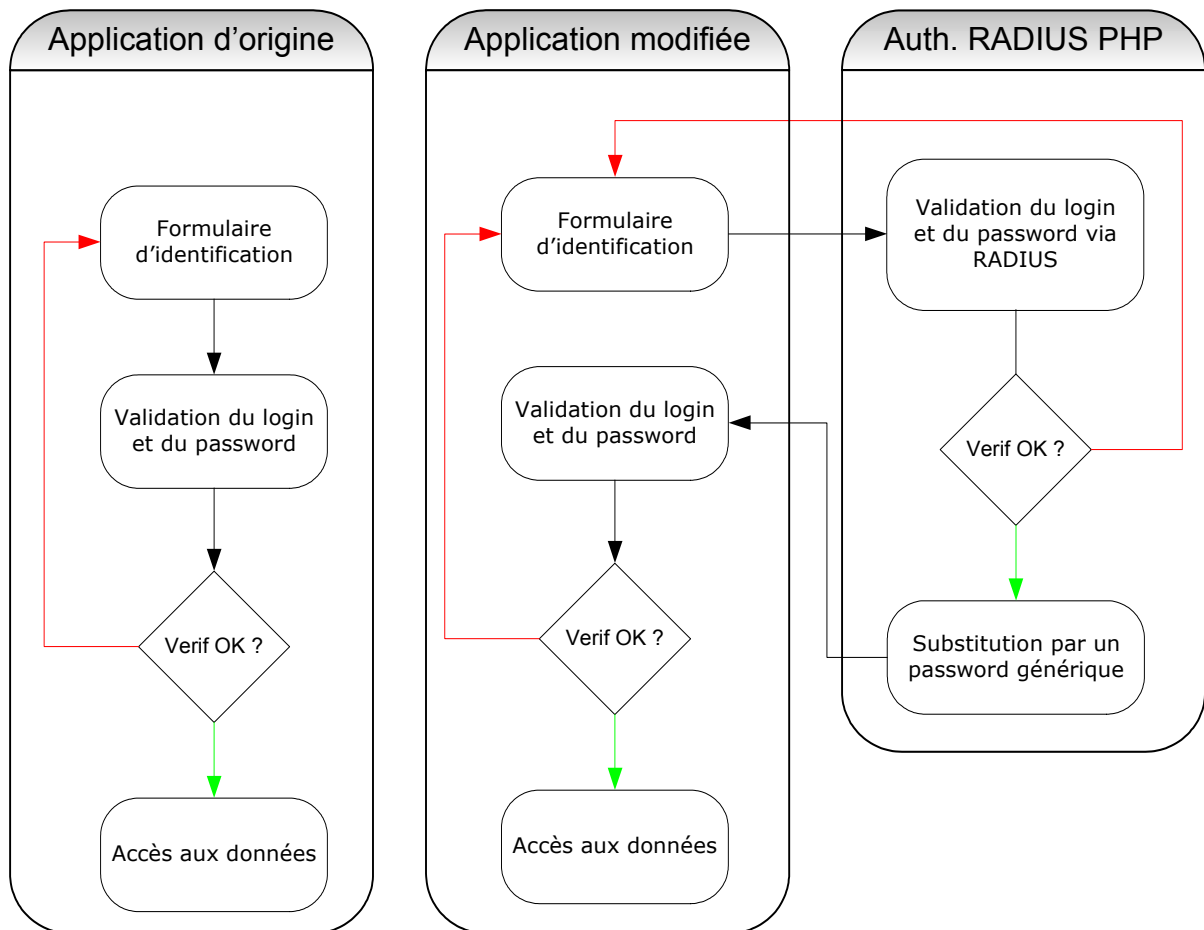
- Chaque utilisateur devant avoir accès à cette application est défini dans l'une de nos deux DB utilisateurs. Le login de l'utilisateur servant d'identifiant, nous avons veillé à ce qu'il soit exactement le même dans la DB de notre programme documentaire et dans nos DB d'utilisateurs.
- Tous les mots de passe des utilisateurs ont alors été modifiés par une chaîne de caractères aléatoire suffisamment complexe.
- Le client WEB étant programmé par le biais d'un langage interprété (ASP), la source du formulaire d'identification a été modifiée de façon à dévier la procédure d'identification. (Le cas échéant, nous aurions pu créer notre propre page d'identification).

Le schéma ci-après permet de se faire une meilleure idée de ce qui a été réalisé :

---

<sup>71</sup> Le cas échéant, on peut bien entendu se reposer plus simplement sur HTTPS.

<sup>72</sup> FreeBSD : <http://www.freebsd.org/>



Le serveur WEB employé est IIS, ce qui ne pose aucun problème à l'emploi de PHP pour mettre en place la déviation. Une fois mise en place, la séquence obtenue est la suivante :

- Le login et le mot de passe sont envoyés par un HTTP POST vers le script de déviation.
- Ce script vérifie à l'aide d'une fonction (voir annexe 4) la validité des informations reçues auprès du serveur RADIUS.
- Après validation, le script de déviation transmet le même login et un mot de passe générique à l'application d'origine.
- Cette dernière reprend son cours et l'utilisateur accède ainsi aux informations.

Il est certain que cette astuce n'est pas sans poser certains problèmes de sécurité, mais elle est l'objet d'un choix entre opérationnalité et sécurité. La modification d'application web non ouverte telle que démontrée dans cet exemple, même si elle fonctionne, n'est clairement pas recommandable.

## Critiques de l'authentification WEB sur RADIUS

L'emploi de RADIUS comme méthode d'authentification WEB est tout à fait fonctionnel si il se limite à ce seul aspect. En effet, la gestion des droits d'une application en mode WEB est totalement impossible avec RADIUS. Et ce n'est pas étonnant vu que l'on sort largement du cadre des opérations orientées réseau.

En ce qui concerne l'accounting, même s'il est exploitable via PHP, son exploitation ne paraît pas souhaitable. La nature de HTTP ne permet pas réellement de maintenir l'état d'une session (même si ce processus est simulable). L'idée est qu'il paraît difficile de savoir avec certitude quand un client ferme son navigateur, quitte le site ou interrompt sa connexion. Ce problème rend donc peu probable l'envoi des paquets `Accounting-Stop` nécessaires à la bonne interprétation des données d'accounting. De plus, les types de données que permet de stocker l'accounting sont limités et il est peu probable qu'elles rencontrent les besoins d'une application WEB.

## 8 Conclusion

---

Quelles sont les possibilités de mise en œuvre de l'identification des membres d'un réseau ? Et peut-on affecter des ressources différentes à ces membres ? Ces deux questions résument les thèmes principaux de ce travail.

Celui-ci m'a permis de mieux appréhender les notions d'authentification et de gestion des droits proposés par une structure AAA. L'application de ces notions au sein du protocole RADIUS les rendant plus tangibles. Mais c'est l'utilisation quotidienne et transparente de ces techniques qui sont la meilleure preuve de leur efficacité.

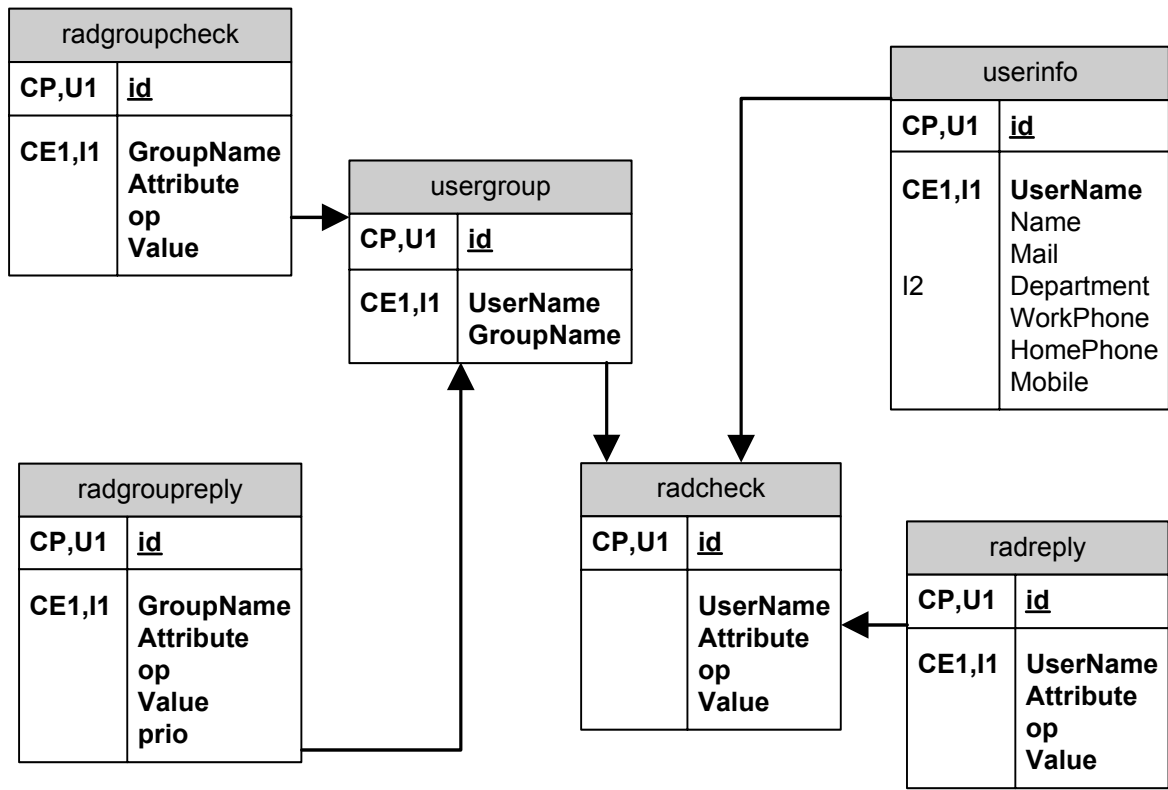
Déployer une architecture AAA de manière à l'incorporer dans un environnement déjà en place, requiert une bonne connaissance de ce dernier. L'emploi de diverses technologies telles qu'un annuaire LDAP, une infrastructure à clé publique ou une base de données SQL nécessaire pour mener à bien le projet, s'est avéré très enrichissant.

Néanmoins, AAA n'est pas la solution universelle d'autorisation et d'identification. Cette structure a ses limites et doit coexister avec d'autres méthodes complémentaires (notamment pour la gestion des droits et les procédés de journalisation) afin de proposer une solution véritablement intégrée.

Ce travail m'a amené à acquérir diverses compétences. L'informatique est un monde en constante évolution et ce travail est pour moi la meilleure preuve que l'informaticien est en quelque sorte « un éternel étudiant ».

## 9 Annexes

### Annexe 1 : Schéma des tables MySQL



radacct	
CP,U1	<u>RadAcctId</u>
I3	AcctSessionId
I4	AcctUniqueId
I1	UserName Realm
I7	NASIPAddress NASPortId NASPortType
I5	AcctStartTime
I6	AcctStopTime AcctSessionTime AcctAuthentic ConnectInfo_start ConnectInfo_stop AcctInputOctets AcctOutputOctets
I2	CalledStationId CallingStationId AcctTerminateCause ServiceType FramedProtocol FramedIPAddress AcctStartDelay AcctStopDelay

nas	
CP,U1	<u>id</u>
U1	nasname shortname type ports secret community description

radpostauth	
CP,U1	<u>id</u>
	user reply client nas date

## Annexe 2 : FreeRADIUS mode DEBUG

```
Starting - reading configuration files ...
reread_config: reading radiusd.conf
Config: including file: ../etc/raddb/proxy.conf
Config: including file: ../etc/raddb/clients.conf
Config: including file: ../etc/raddb/snmp.conf
Config: including file: ../etc/raddb/eap.conf
Config: including file: ../etc/raddb/sql.conf
main: prefix = ".."
main: localstatedir = "../var"
main: logdir = "../var/log/radius"
main: libdir = "../lib"
main: radacctdir = "../var/log/radius/radacct"
main: hostname_lookups = no
main: max_request_time = 30
main: cleanup_delay = 5
main: max_requests = 1024
main: delete_blocked_requests = 0
main: port = 0
main: allow_core_dumps = no
main: log_stripped_names = yes
main: log_file = "../var/log/radius/radius.log"
main: log_auth = yes
main: log_auth_badpass = yes
main: log_auth_goodpass = yes
main: pidfile = "../var/run/radiusd/radiusd.pid"
main: user = "(null)"
main: group = "(null)"
main: usercollide = no
main: lower_user = "no"
main: lower_pass = "no"
main: nospace_user = "no"
main: nospace_pass = "no"
main: checkrad = "../bin/checkrad"
main: proxy_requests = yes
proxy: retry_delay = 5
proxy: retry_count = 3
proxy: synchronous = no
proxy: default_fallback = yes
proxy: dead_time = 120
proxy: post_proxy_authorize = yes
proxy: wake_all_if_all_dead = no
security: max_attributes = 200
security: reject_delay = 1
security: status_server = no
main: debug_level = 0
read_config_files: reading dictionary
read_config_files: reading clients
read_config_files: reading realms
radiusd: entering modules setup
Module: Library search path is ../lib
Module: Loaded exec
exec: wait = yes
exec: program = "(null)"
exec: input_pairs = "request"
exec: output_pairs = "(null)"
exec: packet_type = "(null)"
rlm_exec: Wait=yes but no output defined. Did you mean output=none?
Module: Instantiated exec (exec)
Module: Loaded expr
Module: Instantiated expr (expr)
Module: Loaded pap
pap: encryption_scheme = "clear"
Module: Instantiated pap (pap)
Module: Loaded chap
Module: Instantiated chap (chap)
Module: Loaded mschap
mschap: use_mppe = yes
mschap: require_encryption = no
mschap: require_strong = no
mschap: with_ntdomain_hack = no
mschap: passwd = "(null)"
mschap: authtype = "MS-CHAP"
mschap: ntlm_auth = "(null)"
Module: Instantiated mschap (mschap)
Module: Loaded System
unix: cache = no
unix: passwd = "/etc/passwd"
unix: shadow = "/etc/shadow"
unix: group = "/etc/group"
unix: radwtmp = "../var/log/radius/radwtmp"
unix: usegroup = no
unix: cache_reload = 600
Module: Instantiated unix (unix)
Module: Loaded eap
eap: default_eap_type = "tls"
eap: timer_expire = 60
eap: ignore_unknown_eap_types = no
eap: cisco_accounting_username_bug = no
tls: rsa_key_exchange = no
tls: dh_key_exchange = yes
tls: rsa_key_length = 512
tls: dh_key_length = 512
tls: verify_depth = 0
tls: CA_path = "(null)"
tls: pem_file_type = yes
tls: private_key_file = "../etc/raddb/certs/FreeRADIUS.net/DemoCerts/privkey.pem"
tls: certificate_file = "../etc/raddb/certs/FreeRADIUS.net/DemoCerts/crocut.cer"
tls: CA_file = "../etc/raddb/certs/FreeRADIUS.net/DemoCerts/ca.cer"
tls: private_key_password = "brosteni2814"
tls: dh_file = "../etc/raddb/certs/FreeRADIUS.net/DemoCerts/dh"
tls: random_file = "../etc/raddb/certs/FreeRADIUS.net/DemoCerts/random"
tls: fragment_size = 1024
tls: include_length = yes
tls: check_crl = no
tls: check_cert_cn = "(null)"
rlm_eap: Loaded and initialized type tls
Module: Instantiated eap (eap)
Module: Loaded preprocess
preprocess: huntgroups = "../etc/raddb/huntgroups"
preprocess: hints = "../etc/raddb/hints"
preprocess: with_ascend_hack = no
preprocess: ascend_channels_per_line = 23
preprocess: with_ntdomain_hack = no
preprocess: with_specialix_jetstream_hack = no
preprocess: with_cisco_vsa_hack = no
Module: Instantiated preprocess (preprocess)
Module: Loaded realm
```

```

realm: format = "suffix"
realm: delimiter = "g"
realm: ignore_default = no
realm: ignore_null = no
Module: Instantiated realm (suffix)
Module: Loaded files
files: usersfile = "../etc/raddd/users"
files: acctusersfile = "../etc/raddd/acct_users"
files: preproxy_usersfile = "../etc/raddd/preproxy_users"
files: compat = "no"
Module: Instantiated files (files)
Module: Loaded Acct-Unique-Session-Id
acct_unique: key = "User-Name, Acct-Session-Id, NAS-IP-Address, Client-IP-Address, NAS-Port"
Module: Instantiated acct_unique (acct_unique)
Module: Loaded detail
detail: detailfile = "../var/log/radius/radacct/%(Client-IP-Address)/detail-%Y%m%d"
detail: detailperm = 384
detail: dirperm = 493
detail: locking = no
Module: Instantiated detail (detail)
Module: Loaded radutmp
radutmp: filename = "../var/log/radius/radutmp"
radutmp: username = "%(User-Name)"
radutmp: case_sensitive = yes
radutmp: check_with_nas = yes
radutmp: perm = 384
radutmp: callerid = yes
Module: Instantiated radutmp (radutmp)
Listening on authentication *:1812
Listening on accounting *:1813
Listening on proxy *:1814
Ready to process requests.

rad_recv: Accounting-Request packet from host 127.0.0.1:1701, id=6, length=43
User-Name = "toto"
Acct-Status-Type = Start
Acct-Session-Id = "812"
NAS-Port = 0
Processing the preacct section of radiusd.conf
modcall: entering group preacct for request 0
modcall[preacct]: module "preprocess" returns noop for request 0
rlm_acct_unique: Hashing 'NAS-Port = 0,Client-IP-Address = 127.0.0.1,NAS-IP-Address = 127.0.0.1,Acct-Session-Id = "812",User-Name = "toto"'
rlm_acct_unique: Acct-Unique-Session-ID = "ac5efbe5b6adeb23".
modcall[preacct]: module "acct_unique" returns ok for request 0
rlm_realm: No '@' in User-Name = "toto", looking up realm NULL
rlm_realm: No such realm "NULL"
modcall[preacct]: module "suffix" returns noop for request 0
modcall[preacct]: module "files" returns noop for request 0
modcall: group preacct returns ok for request 0
Processing the accounting section of radiusd.conf
modcall: entering group accounting for request 0
radius_xlat: '../var/log/radius/radacct/127.0.0.1/detail-20050817'
rlm_detail: ../var/log/radius/radacct/%(Client-IP-Address)/detail-%Y%m%d expands to ../var/log/radius/radacct/127.0.0.1/detail-20050817
modcall[accounting]: module "detail" returns ok for request 0
radius_xlat: '../var/log/radius/radutmp'
radius_xlat: 'toto'
modcall[accounting]: module "radutmp" returns ok for request 0
modcall: group accounting returns ok for request 0
Sending Accounting-Response of id 6 to 127.0.0.1:1701
Finished request 0
Going to the next request
--- Walking the entire request list ---
Cleaning up request 0 ID 6 with timestamp 4303a8ac
Nothing to do. Sleeping until we see a request.

rad_recv: Accounting-Request packet from host 127.0.0.1:1706, id=11, length=67
User-Name = "toto"
Acct-Status-Type = Stop
Acct-Session-Id = "812"
NAS-Port = 0
Acct-Input-Octets = 14265
Acct-Output-Octets = 1984357
Acct-Session-Time = 1247
Acct-Terminate-Cause = User-Request
Processing the preacct section of radiusd.conf
modcall: entering group preacct for request 5
modcall[preacct]: module "preprocess" returns noop for request 5
rlm_acct_unique: Hashing 'NAS-Port = 0,Client-IP-Address = 127.0.0.1,NAS-IP-Address = 127.0.0.1,Acct-Session-Id = "812",User-Name = "toto"'
rlm_acct_unique: Acct-Unique-Session-ID = "ac5efbe5b6adeb23".
modcall[preacct]: module "acct_unique" returns ok for request 5
rlm_realm: No '@' in User-Name = "toto", looking up realm NULL
rlm_realm: No such realm "NULL"
modcall[preacct]: module "suffix" returns noop for request 5
modcall[preacct]: module "files" returns noop for request 5
modcall: group preacct returns ok for request 5
Processing the accounting section of radiusd.conf
modcall: entering group accounting for request 5
radius_xlat: '../var/log/radius/radacct/127.0.0.1/detail-20050817'
rlm_detail: ../var/log/radius/radacct/%(Client-IP-Address)/detail-%Y%m%d expands to ../var/log/radius/radacct/127.0.0.1/detail-20050817
modcall[accounting]: module "detail" returns ok for request 5
radius_xlat: '../var/log/radius/radutmp'
radius_xlat: 'toto'
modcall[accounting]: module "radutmp" returns ok for request 5
modcall: group accounting returns ok for request 5
Sending Accounting-Response of id 11 to 127.0.0.1:1706
Finished request 5
Going to the next request
--- Walking the entire request list ---
Cleaning up request 5 ID 11 with timestamp 4303aa96
Nothing to do. Sleeping until we see a request.

```

## Annexe 3 : Exemple de dictionnaire d'attributs

ID	NOM	TYPE
=====		
1	User-Name	STRING
2	User-Password	STRING
3	CHAP-Password	STRING
4	NAS-IP-Address	STRING
5	NAS-Port	INT
6	Service-Type	ENUM
7	Framed-Protocol	ENUM
8	Framed-IP-Address	STRING
9	Framed-IP-Netmask	STRING
10	Framed-Routing	ENUM
11	Filter-Id	STRING
12	Framed-MTU	INT
13	Framed-Compression	ENUM
14	Login-IP-Host	STRING
15	Login-Service	ENUM
16	Login-TCP-Port	INT
18	Reply-Message	STRING
19	Callback-Number	STRING
20	Callback-Id	STRING
22	Framed-Route	STRING
23	Framed-IPX-Network	STRING
24	State	STRING
25	Class	STRING
26	Vendor-Specific	STRING
27	Session-Timeout	INT
28	Idle-Timeout	INT
29	Termination-Action	ENUM
30	Called-Station-Id	STRING
31	Calling-Station-Id	STRING
32	NAS-Identifiant	STRING
33	Proxy-State	STRING
34	Login-LAT-Service	STRING
35	Login-LAT-Node	STRING
36	Login-LAT-Group	STRING
37	Framed-AppleTalk-Link	INT
38	Framed-AppleTalk-Network	INT
39	Framed-AppleTalk-Zone	STRING
40	Acct-Status-Type	ENUM
41	Acct-Delay-Time	INT
42	Acct-Input-Octets	INT
43	Acct-Output-Octets	INT
44	Acct-Session-Id	STRING
45	Acct-Authentic	ENUM
46	Acct-Session-Time	INT
47	Acct-Input-Packets	INT
48	Acct-Output-Packets	INT
49	Acct-Terminate-Cause	ENUM
50	Acct-Multi-Session-Id	STRING
51	Acct-Link-Count	INT
52	Acct-Input-Gigawords	INT
53	Acct-Output-Gigawords	INT
55	Event-Timestamp	DATE
60	CHAP-Challenge	STRING
61	NAS-Port-Type	INT
62	Port-Limit	INT
63	Login-LAT-Port	INT

ENUM : Framed Compression

ID	NOM	VAL
=====		
13	None	0
13	Van-Jacobson-TCP-IP	1
13	IPX-Header-Compression	2
13	Stac-LZS	3

# Annexe 4 : PHP d'authentification RADIUS

```
function radiusAuth($username,$password,$radserver,$radport=1812,$sharedsecret='testing123',$auth_type='pap')
{
    $auth_type = trim(strtolower($auth_type));
    if (substr($auth_type,0,7) == 'mschapv')
        include_once('mschap.php');
    $return_pkt = "";

    $res = radius_auth_open();

    if (!radius_add_server($res, $radserver, $radport, $sharedsecret, 3, 3))
        return 'RadiusError : ' . radius_strerror($res);
    if (!radius_create_request($res, RADIUS_ACCESS_REQUEST))
        return 'RadiusError : ' . radius_strerror($res);
    if (!radius_put_string($res, RADIUS_USER_NAME, $username))
        return 'RadiusError : ' . radius_strerror($res);

    if ($auth_type == 'pap')
    {
        if (!radius_put_string($res, RADIUS_USER_PASSWORD, $password))
            return 'RadiusError : ' . radius_strerror($res);
    }
    else if ($auth_type == 'chap')
    {
        /* generate Challenge */
        mt_srand(time());
        $chall = mt_rand();

        // FYI: CHAP = md5(ident + plaintextpass + challenge)
        $chapval = pack('H*', md5(pack('Ca*',1, $password . $chall)));

        // Radius wants the CHAP Ident in the first byte of the CHAP-Password
        $pass = pack('C', 1) . $chapval;

        if (!radius_put_attr($res, RADIUS_CHAP_PASSWORD, $pass))
            return 'RadiusError : RADIUS_CHAP_PASSWORD: ' . radius_strerror($res);
        if (!radius_put_attr($res, RADIUS_CHAP_CHALLENGE, $chall))
            return 'RadiusError : RADIUS_CHAP_CHALLENGE: ' . radius_strerror($res);
    }
    else if ($auth_type == 'mschapv1')
    {
        $challenge = GenerateChallenge();

        if (!radius_put_vendor_attr($res, RADIUS_VENDOR_MICROSOFT, RADIUS_MICROSOFT_MS_CHAP_CHALLENGE, $challenge))
            return 'RadiusError : RADIUS_MICROSOFT_MS_CHAP_CHALLENGE: ' . radius_strerror($res);

        $ntresp = ChallengeResponse($challenge, NtPasswordHash($password));
        $lmresp = str_repeat ("\0", 24);
        // Response: chapid, flags (1 = use NT Response), LM Response, NT Response
        $resp = pack('CCa48',1, 1, $lmresp . $ntresp);

        if (!radius_put_vendor_attr($res, RADIUS_VENDOR_MICROSOFT, RADIUS_MICROSOFT_MS_CHAP_RESPONSE, $resp))
            return 'RadiusError : RADIUS_MICROSOFT_MS_CHAP_RESPONSE: ' . radius_strerror($res);
    }
    else if ($auth_type == 'mschapv2')
    {
        $authChallenge = GenerateChallenge(16);

        if (!radius_put_vendor_attr($res, RADIUS_VENDOR_MICROSOFT, RADIUS_MICROSOFT_MS_CHAP_CHALLENGE, $authChallenge))
            return 'RadiusError : RADIUS_MICROSOFT_MS_CHAP_CHALLENGE: ' . radius_strerror($res);

        // we have no client, therefore we generate the Peer-Challenge
        $peerChallenge = GeneratePeerChallenge();
        $ntresp = GenerateNTResponse($authChallenge, $peerChallenge, $username, $password);
        $reserved = str_repeat ("\0", 8);
        // Response: chapid, flags (1 = use NT Response), Peer challenge, reserved, Response
        $resp = pack('CCa16a8a24',1, 1, $peerChallenge, $reserved, $ntresp);

        if (!radius_put_vendor_attr($res, RADIUS_VENDOR_MICROSOFT, RADIUS_MICROSOFT_MS_CHAP2_RESPONSE, $resp))
            return 'RadiusError : RADIUS_MICROSOFT_MS_CHAP2_RESPONSE: ' . radius_strerror($res);
    }
}
/*
if (!radius_put_int($res, RADIUS_SERVICE_TYPE, RADIUS_FRAMED))
    return 'RadiusError : ' . radius_strerror($res);
if (!radius_put_int($res, RADIUS_FRAMED_PROTOCOL, RADIUS_PPP))
    return 'RadiusError : ' . radius_strerror($res);
*/
if (!radius_put_string($res, RADIUS_NAS_IDENTIFIER, $_SERVER["HTTP_HOST"]))
    return 'RadiusError : ' . radius_strerror($res);
if (!radius_put_int($res, RADIUS_SERVICE_TYPE, RADIUS_AUTHENTICATE_ONLY))
    return 'RadiusError : ' . radius_strerror($res);

if ($_SERVER["SERVER_ADDR"]!=null)
    $server_addr = gethostbyname($_SERVER["HTTP_HOST"]);
else
    $server_addr = $_SERVER["SERVER_ADDR"];

if (!radius_put_addr($res, RADIUS_NAS_IP_ADDRESS, $server_addr))
    return 'RadiusError : ' . radius_strerror($res);
if (!radius_put_string($res, RADIUS_CALLING_STATION_ID, $_SERVER["REMOTE_ADDR"]))
    return 'RadiusError : ' . radius_strerror($res);

$req = radius_send_request($res);
if (!$req)
    return 'RadiusError : ' . radius_strerror($res);

switch($req)
{
    case RADIUS_ACCESS_ACCEPT:
        $return_pkt.= "Access-Accept\n";
        break;

    case RADIUS_ACCESS_REJECT:
        $return_pkt.= "Access-Reject\n";
        break;

    default:
        $return_pkt.= "Unexpected return value\n";
        break;
}

while ($resa = radius_get_attr($res))
{
    if (!is_array($resa))
        return "Return_pkt Error getting attribute: ".radius_strerror($res)."\n";

    $attr = $resa['attr'];
```

```

$data = $resa['data'];

switch ($attr)
{
    case RADIUS_FRAMED_IP_ADDRESS:
        $ip = radius_cvt_addr($data);
        $return_pkt.= "IP: $ip\n";
        break;

    case RADIUS_FRAMED_IP_NETMASK:
        $mask = radius_cvt_addr($data);
        $return_pkt.= "MASK: $mask\n";
        break;

    case RADIUS_FRAMED_MTU:
        $mtu = radius_cvt_int($data);
        $return_pkt.= "MTU: $mtu\n";
        break;

    case RADIUS_FRAMED_COMPRESSION:
        $comp = radius_cvt_int($data);
        $return_pkt.= "Compression: $comp\n";
        break;

    case RADIUS_SESSION_TIMEOUT:
        $time = radius_cvt_int($data);
        $return_pkt.= "Session timeout: $time\n";
        break;

    case RADIUS_IDLE_TIMEOUT:
        $idletime = radius_cvt_int($data);
        $return_pkt.= "Idle timeout: $idletime\n";
        break;

    case RADIUS_SERVICE_TYPE:
        $stype = radius_cvt_int($data);
        $return_pkt.= "Service Type: $stype\n";
        break;

    case RADIUS_CLASS:
        $class = radius_cvt_int($data);
        $return_pkt.= "Class: $class\n";
        break;

    case RADIUS_FRAMED_PROTOCOL:
        $proto = radius_cvt_int($data);
        $return_pkt.= "Protocol: $proto\n";
        break;

    case RADIUS_FRAMED_ROUTING:
        $rout = radius_cvt_int($data);
        $return_pkt.= "Routing: $rout\n";
        break;

    case RADIUS_FILTER_ID:
        $fid = radius_cvt_string($data);
        $return_pkt.= "Filter ID: $fid\n";
        break;

    case RADIUS_VENDOR_SPECIFIC:
        $resv = radius_get_vendor_attr($data);
        if (is_array($resv))
        {
            $vendor = $resv['vendor'];
            $attrv = $resv['attr'];
            $datav = $resv['data'];
            if ($vendor == RADIUS_VENDOR_MICROSOFT)
            {
                switch ($attrv)
                {
                    case RADIUS_MICROSOFT_MS_CHAP2_SUCCESS:
                        $mschap2resp = radius_cvt_string($datav);
                        $return_pkt.= "MS CHAPv2 success: $mschap2resp\n";
                        break;

                    case RADIUS_MICROSOFT_MS_CHAP_ERROR:
                        $errmsg = radius_cvt_string(substr($datav,1));
                        $return_pkt.= "MS CHAP Error: $errmsg\n";
                        break;

                    case RADIUS_MICROSOFT_MS_CHAP_DOMAIN:
                        $domain = radius_cvt_string($datav);
                        $return_pkt.= "MS CHAP Domain: $domain\n";
                        break;

                    case RADIUS_MICROSOFT_MS_MPPE_ENCRYPTION_POLICY:
                        $policy = radius_cvt_int($datav);
                        $return_pkt.= "MS MPPE Policy: $policy\n";
                        break;

                    case RADIUS_MICROSOFT_MS_MPPE_ENCRYPTION_TYPES:
                        $stype = radius_cvt_int($datav);
                        $return_pkt.= "MS MPPE Type: $stype\n";
                        break;

                    case RADIUS_MICROSOFT_MS_CHAP_MPPE_KEYS:
                        $demangled = radius_demangle($res, $datav);
                        $lmkey = substr($demangled, 0, 8);
                        $ntkey = substr($demangled, 8, RADIUS_MPPE_KEY_LEN);
                        $return_pkt.= "MS MPPE Keys: LM-Key: ".bin2hex($lmkey)." NT-Key: ".bin2hex($ntkey)." \n";
                        break;

                    case RADIUS_MICROSOFT_MS_MPPE_SEND_KEY:
                        $demangled = radius_demangle_mppe_key($res, $datav);
                        $return_pkt.= "MS MPPE Send Key: ".bin2hex($demangled)." \n";
                        break;

                    case RADIUS_MICROSOFT_MS_MPPE_RECV_KEY:
                        $demangled = radius_demangle_mppe_key($res, $datav);
                        $return_pkt.= "MS MPPE Send Key: ".bin2hex($demangled)." \n";
                        break;

                    case RADIUS_MICROSOFT_MS_PRIMARY_DNS_SERVER:
                        $server = radius_cvt_string($datav);
                        $return_pkt.= "MS Primary DNS Server: $server\n";
                        break;

                    default:
                        $return_pkt.= "Unexpected Microsoft attribute: $attrv\n";
                }
            }
        }
    }
}
else

```

```
        $return_pkt.= "Error getting Vendor attribute ".radius_strerror($res)."\n";
        break;
    default:
        $return_pkt.= "Unexpected attribute : $attr\n";
        break;
    }
}
$secret = radius_server_secret($res);
if (!$secret)
    return 'RadiusError : ' . radius_strerror($res);
$authent = radius_request_authenticator($res);
if (!$authent)
    return 'RadiusError : ' . radius_strerror($res);
else
    $return_pkt.= "Request Authenticator : ".bin2hex($authent)." Len : ".strlen($authent)."\n";
radius_close($res);
return $return_pkt;
```

## 10 Références

---

### Livres :

- **RADIUS – Securing Public Access to Private Ressources**  
Jonathan Hassell  
O'Reilly  
ISBN : 0-596-00322-6
- **LDAP : Administration système**  
Gerald Carter  
O'Reilly  
ISBN : 2-841-77293-4

### Publications internet (White Paper) :

- **Windows 2000 Kerberos Authentication**  
<http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/kerberos.msp>
- **HP Introduction to Diameter**  
<http://docs.hp.com/en/T1428-90011/index.html>
- **IEFT - RFC 1510, 2138, 2139, 2284, 2903, 2904**  
<http://www.ietf.org/rfc/rfcXXXX.txt>

### Site web consultés :

- **Crypto FAQ – RSA Laboratories**  
<http://www.rsasecurity.com/rsalabs/>
- **Wikipédia**  
<http://fr.wikipedia.org/wiki/Accueil>
- **Comment ça marche ?**  
<http://www.commentcamarche.net/>