L'Internet Rapide et Permanent

Sécuriser son réseau

Par Christian Caleca

Date de publication : 18 février 2012

Vous disposez d'une connexion permanente et rapide... et maintenant, vous êtes perdu dans la technique...

Cette série « L'Internet Rapide et Permanent », que Christian Caleca nous a aimablement autorisé à reproduire, est là pour répondre à quelques-unes de ces questions. Cet article parlera de la technique des VLAN avec des switches « 802.1q », de la norme WPA2 et de la mise en place d'un serveur RADIUS.

N'hésitez pas à commenter cet article !

1 - Sécuriser son réseau

Lorsque l'on doit assurer le bon fonctionnement d'un réseau qui dépasse les dimensions du réseau familial, il devient nécessaire de s'assurer aussi que le danger ne vienne pas de l'intérieur. Avec la prolifération des ordinateurs personnels portables, le risque de voir une machine inconnue venir polluer le réseau de l'intérieur doit être pris au sérieux.

Pour peu qu'un accès Wi-Fi soit également disponible, il convient en plus de s'assurer que seules les personnes autorisées puissent s'y attacher.

Ce très gros chapitre a pour but d'apporter quelques ébauches de solutions à tous ces problèmes. Nous ferons appel aux VLAN pour le réseau câblé, à la norme WPA2 pour le Wi-Fi, en mettant en œuvre une authentification RADIUS.

Dans le but de rendre plus digestes toutes ces technologies, ce chapitre est lui-même divisé en trois sous-chapitres qui traitent successivement de :

- la technique des VLAN, avec des switches « 802.1q » ;
- la norme WPA2 ;
- la mise en place d'un serveur RADIUS.

1-1 - Sécurité du réseau filaire

Les stations fixes dont nous disposons sur ce réseau sont réputées « sûres ». En effet l'administrateur en a la maîtrise, peut en contrôler l'intégrité, peut limiter les droits des utilisateurs sur ces postes de travail.

En revanche, un ordinateur portable échappe à tout contrôle. Son propriétaire en fait ce qu'il veut, la machine peut être gravement compromise et infecter par l'intérieur notre réseau local. Dans ces conditions, il faudra que ces machines incontrôlables soient isolées du réseau local :

- soit en leur interdisant purement et simplement l'accès au réseau, nous verrons que les « switches » modernes savent réaliser cette opération ;
- soit en les connectant sur un réseau différent, un réseau « d'invités », qui ne leur donnera pas accès aux ressources locales, mais à quelques ressources sans grands risques, comme un accès internet plus ou moins limité, un accès à un serveur FTP, bref, à vous de voir ce que vous voulez protéger et ce que vous voulez autoriser.

Dans le cadre de cette étude, nous utiliserons l'adresse MAC de la station, pour vérifier si cette station est connue et autorisée à se connecter au réseau local, ou si elle n'est pas connue, la diriger sur un réseau d'invités. Certes, utiliser l'adresse MAC n'est pas une garantie absolue, mais c'est une solution simple à mettre en œuvre, si nos switches savent offrir cette possibilité.

Les VLAN (Virtual Local Area Network) permettent « simplement » d'assigner dynamiquement un port (entendez par là un connecteur du switch) à un LAN ou à un autre, d'où la notion de VLAN. Nous verrons tout ceci en détail plus loin.

Pour mener à bien cette tâche, un serveur d'authentification de type RADIUS nous sera nécessaire.

1-2 - Sécurité du réseau sans fil

Ici, le problème est différent. Il faut s'assurer que seuls les invités peuvent se connecter, et non pas les intrus. De plus, il faut éviter que les invités se connectent à une borne pirate à leur insu. Pour réaliser ceci, nous ferons appel à EAP et WPA-TLS, un système d'authentification qui fait lui aussi intervenir un serveur RADIUS.

1-3 - Politique de sécurité

Nous allons certainement nous faire quelques ennemis de plus chez les utilisateurs (un administrateur a rarement des amis chez les utilisateurs), mais la politique retenue est la suivante :

nous créons un LAN « invités », sur lequel il n'y a pas de ressources sensibles, isolé du LAN des « permanents » (nous raisonnons ici en termes de machines et pas d'utilisateurs).

Les stations « connues », celles dont l'administrateur a la maîtrise, auront le droit de se connecter (par liaison filaire) sur le LAN des « permanents ».

Les stations « inconnues » (typiquement les portables), pourront :

- se connecter sans autorisation particulière au réseau filaire, mais seront intégrées au LAN « invités » (les VLAN viendront à notre secours pour cette opération) ;
- se connecter au réseau sans fil, avec l'accord de l'administrateur. La station se retrouvera alors sur le LAN « invités ».

Il est possible de faire plus fin, mais pour comprendre le principe, cette démarche sera suffisante.

2 - Les VLAN

Dans ce premier sous-chapitre, nous allons voir ce que c'est qu'un VLAN, comment il est possible de configurer un switch pour qu'il gère plusieurs VLAN :

- de façon statique (niveau 1), façon de faire ne nécessitant rien d'autre qu'un switch administrable et ne fait pas appel à un système d'authentification, mais qui ne répondra pas à notre cahier des charges initial ;
- de façon dynamique à partir des adresses MAC des clients (niveau 2), il faudra ici non seulement un switch administrable sachant dialoguer avec un serveur RADIUS pour la consultation des adresses MAC autorisées, mais aussi la mise en place de ce serveur RADIUS (que nous verrons dans le sous-chapitre suivant).

2-1 - Théorie

2-1-1 - Les bases

2-1-1-1 - Un LAN

Nous sommes ici, c'est sous-entendu tout au long de cet exposé, sur un réseau Ethernet.

Un LAN est un réseau local dans lequel toutes les trames Ethernet sont visibles depuis tous les nœuds si le LAN est construit avec un hub. Si nous avons affaire à un switch, seules les trames de diffusions (broadcast) seront visibles depuis tous les nœuds, le switch agissant comme un pont Ethernet entre chaque nœud du LAN.

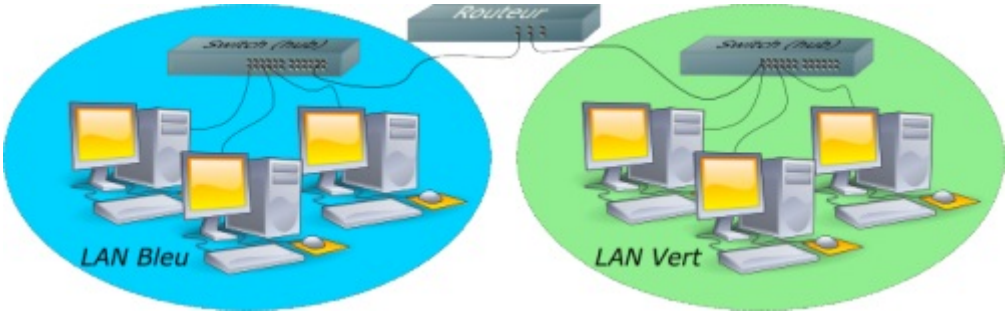
Aujourd'hui, les hubs ont quasiment disparu des catalogues des constructeurs. Compte tenu de l'usage grandissant des réseaux, il est clair que le hub ne peut être considéré comme une bonne solution que sur les tout petits réseaux.

Au-dessus de tout ceci, nous construirons un réseau IP, mais c'est à la limite assez peu notre problème ici. Que le réseau de niveau 3 soit IP, NetBEUI ou AppleTalk n'a aucune importance. Bien entendu, dans la suite, nous n'utiliserons qu'IP.

Ce qu'il est fondamental de comprendre, c'est que nous raisonnons au niveau Ethernet, que nous ne parlons que d'adresses MAC.

Un switch, c'est le composant que nous utiliserons par la suite, à l'exclusion des hubs, est capable d'apprendre et de retenir la ou les adresses MAC qui se présentent sur chacun de ses ports. Hormis les trames de diffusion qui seront systématiquement répercutées sur tous les ports, le switch ne laissera communiquer entre eux que les ports concernés par un dialogue entre deux nœuds. C'est sa fonction principale de pont Ethernet.

2-1-1-2 - Deux LAN (ou plus)

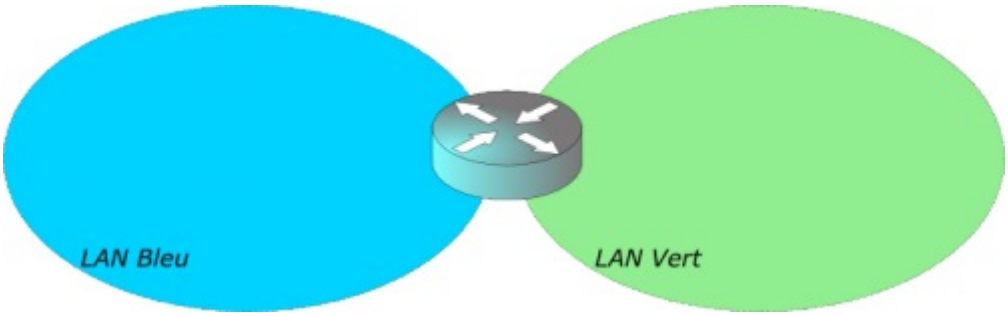


Lorsque nous avons deux LAN et que nous souhaitons les interconnecter, tout en conservant dans chaque LAN les mêmes propriétés au niveau Ethernet, nous devons faire appel à la couche 3 (IP) pour assurer l'interconnexion. Il nous faut donc un routeur.

Le routeur agit au niveau 3 (IP). Ce qu'il est absolument fondamental de comprendre, c'est qu'au niveau Ethernet, le LAN bleu ignore complètement l'existence du LAN vert, et réciproquement. Les trames Ethernet, qu'elles soient de la diffusion ou non, n'iront jamais dans l'autre LAN. Il y a isolation complète des deux LAN au niveau Ethernet et la présence du routeur n'y change rien. Les trames Ethernet qui transportent des données depuis le LAN vert dans le LAN bleu ne seront rien d'autre que des trames Ethernet issues du routeur côté LAN bleu (et réciproquement). Comme dans un roman de science-fiction de bonne facture, les mondes Ethernet bleu et vert sont des mondes parallèles, avec de temps en temps, une porte mystérieuse qui s'ouvre pour laisser passer des choses d'un monde à l'autre, mais en leur faisant perdre la mémoire de leur origine réelle (nous sommes au niveau 2, n'oublions pas).

Il faut monter au niveau de conscience supérieur (niveau IP), pour commencer à démythifier le fonctionnement de ces portes.

Mais à cette hauteur, les détails lointains s'estompent. Ce qu'il y a exactement dans chaque LAN au niveau Ethernet importe finalement assez peu. À première vue, le panorama serait plutôt le suivant :

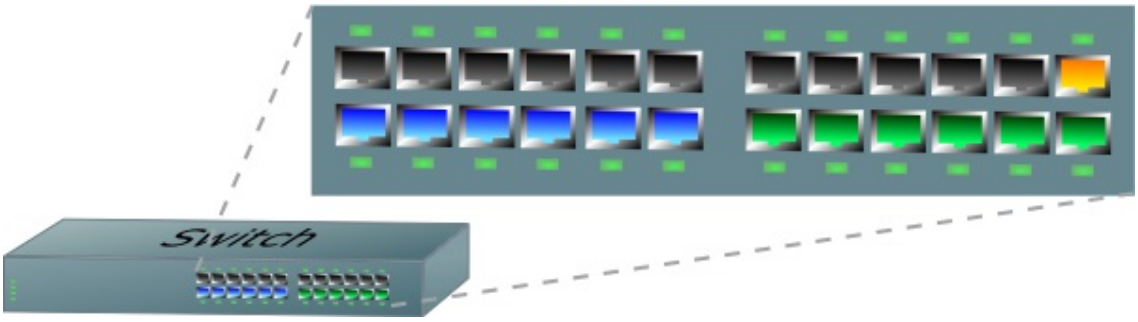


Parce que, finalement, lors de l'interconnexion de réseaux, peu importe ce qu'il y a dans chaque réseau, ce sont les routes qui importent le plus.

Entendons par là que les équipements utilisés pour construire chaque LAN, qu'il s'agisse de hubs ou de switches ou d'un mélange des deux n'a aucune importance.

2-1-1-3 - Où intervient le virtuel

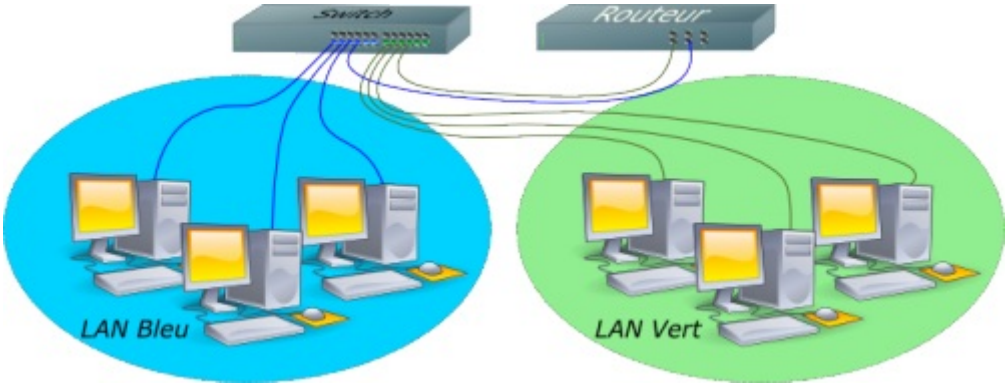
Jusqu'ici, un switch appartenait à un et un seul LAN. L'idée de base est de pouvoir assigner certains ports du switch à un LAN, certains autres ports à un autre LAN, etc. :



Sur un même switch physique, nous allons pouvoir créer plusieurs LAN et assigner certains de ses ports aux divers LAN créés. Ici, nous avons un LAN bleu et un LAN vert. Le port orangé est un peu spécial, il appartient à la fois aux deux LAN, mais il ne va pas nous servir tout de suite.

Tout va (presque) se passer comme si l'on avait découpé notre switch en deux morceaux (sans pour autant le détruire).

Dans une première approche, notre maquette deviendrait ceci :



Le switch a été virtuellement coupé en deux. Les deux VLAN sont complètement étanches au niveau Ethernet (un switch est en principe un outil qui ne va pas au-delà du niveau 2). Pour interconnecter ces deux LAN, un routeur est toujours nécessaire.

2-1-2 - Pour quoi faire ?

Il y a bien entendu quelques avantages à pratiquer de la sorte. Nous pouvons au moins en citer deux :

- optimisation du matériel. En effet, c'est évident sur l'illustration, nous n'avons plus besoin que d'un seul switch, là où il nous en fallait deux au départ ;
- passer d'un poste de travail d'un LAN à l'autre devrait pouvoir se faire de façon « soft ». Plutôt que de débrancher puis de rebrancher ailleurs le lien du poste, nous pourrons le faire par l'outil de configuration du switch.

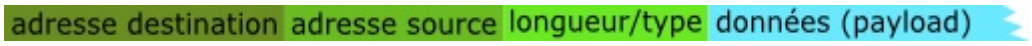
Voilà pour le principe de base. Vous devinez que si je prends la peine de rédiger un chapitre sur les VLAN, c'est qu'il y a d'autres choses encore derrière ce concept. Jusqu'ici, c'est assez simple. Les choses vont maintenant se compliquer progressivement pour arriver à des solutions qui peuvent vite devenir un casse-tête. Il faudra alors résister à la tentation de réaliser des « usines à gaz » là où ce n'est pas nécessaire. Les solutions les plus simples, pourvu qu'elles répondent au cahier des charges, sont toujours les meilleures.

2-1-3 - 802.1q (ou l'art du tag)

Ici, l'idée serait d'arriver à ce que certains ports du switch puissent être assignés à plusieurs VLAN, ça fera économiser du câble (et aussi des ports sur le switch).

Le principe consiste à ajouter dans l'en-tête de la trame Ethernet un marqueur qui va identifier le VLAN. Il existe quelques solutions propriétaires pour réaliser ceci, mais le système s'est avéré tellement intéressant qu'une norme a été définie, il s'agit de la norme 802.1q.

Alors qu'une trame Ethernet « normale » est constituée comme ceci :



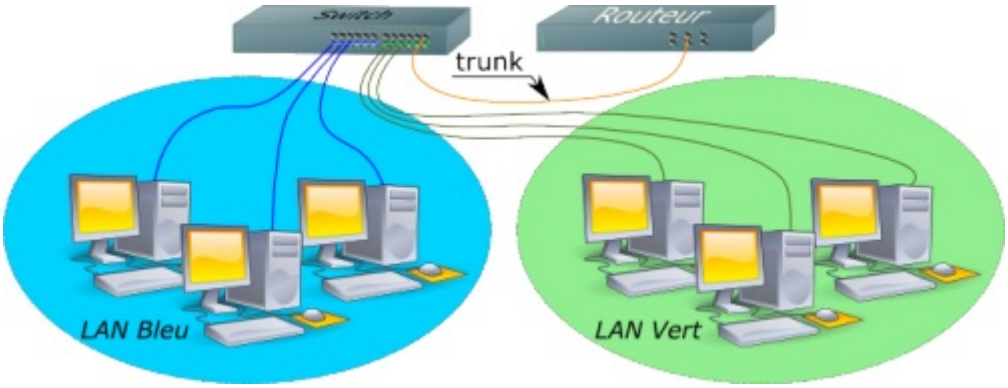
Une trame modifiée par la norme 802.1q se trouve allongée de 4 octets :

Il n'est peut-être pas nécessaire de détailler le contenu de ces deux nouveaux champs. Pour l'instant, retenons que le VID (Identifiant du VLAN) est codé sur 12 bits, ce qui laisse une latitude confortable.

Il est aussi nécessaire de rappeler qu'une trame Ethernet ne doit pas dépasser 1518 octets et que donc, quatre octets de plus dans l'en-tête risquent d'aboutir à une fragmentation des trames, ce qui n'est jamais bien bon. Si l'on doit avoir recours à des VLAN « tagués », il sera sans doute nécessaire de prévoir ce détail.

Finalement, notre switch a donc la possibilité d'ajouter ces marqueurs aux trames Ethernet. Si c'est le cas, il sera alors possible théoriquement d'assigner un même port à 212 VLAN différents. Grâce au VID de chaque VLAN, les données seront acheminées correctement.

Si nous appliquons cette technique à notre maquette, nous obtenons ceci :



Que les esprits sensibles gardent leur sérénité. Il n'y a effectivement qu'un seul câble qui relie l'unique switch au routeur, et pourtant, nous allons effectivement router les données entre les deux LAN. Il y a tout de même une condition à respecter : le routeur doit être « 802.1q compliant », c'est-à-dire qu'il doit savoir lire les tags que le switch a posés sur au moins l'un des deux VLAN.

2-1-3-1 - Explications au niveau 2

Le port orangé, marqué « trunk » appartient à la fois aux deux VLAN bleu et vert. Sur ce port, il faut bien sûr qu'au moins l'un des deux VLAN soit « tagué ».

Sur le câble relié à ce port, il circulera donc à la fois les trames du VLAN bleu et celles du VLAN vert. Il n'y aura pas de problèmes tant qu'à chaque bout du câble, l'interface Ethernet sera capable de trier les trames en fonction du tag. Ceci impose donc naturellement que le routeur soit compatible avec la norme 802.1q, c'est-à-dire que son interface soit capable d'exploiter ces tags.

Sous Linux, c'est tout à fait possible, il existe sur les distributions modernes un module spécialisé : le module 8021q (testé sur Debian Sarge et Etch).

Attention tout de même, ce lien va supporter le trafic des deux VLAN, il faut veiller à ce qu'il ne soit pas engorgé.

2-1-3-2 - Explications au niveau 3

Le switch n'a (en principe) rien à faire du niveau 3. Chacun des VLAN se trouvera avec un plan d'adressage IP qui lui est propre, mais le switch n'est pas concerné, si ce n'est par le fait que pour l'administrer, il faudra bien y accéder par IP. Pour ce faire, le switch disposera d'une adresse IP sur au moins l'un des VLAN, et la machine d'administration devra pouvoir accéder à ce VLAN. Il y aura quelques problèmes de sécurité à envisager à ce niveau, mais nous n'y sommes pas encore.

Au niveau du routeur, en revanche, il faudra que l'interface Ethernet physique puisse présenter autant d'interfaces virtuelles qu'il y a de VLAN sur le « trunk », chacune avec une adresse IP dans le VLAN concerné.

2-1-4 - Attribution d'un port à un VLAN

Il y a plusieurs façons de s'y prendre. Vous trouverez sans doute de nombreuses pages qui traitent ce sujet, en vous parlant des VLAN de niveau 1, 2, voire 3. Nous allons essayer de voir ceci de façon plus pragmatique.

2-1-4-1 - Attribution statique (niveau 1)

C'est la méthode la plus simple et aussi la moins souple, qui consiste, comme nous l'avons sous-entendu jusqu'ici, à attribuer un port du switch à un VLAN donné, en configurant statiquement le switch. Nous n'avons besoin de rien d'autre que d'un switch administrable.

2-1-4-2 - Attribution dynamique (niveaux > 1)

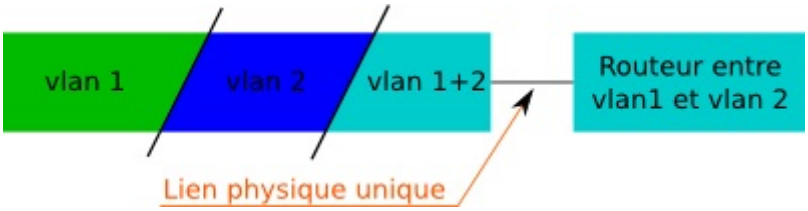
Ici, nous ferons appel à 802.1x et à un procédé d'authentification. Dans ce qui suit, nous disposerons d'un switch capable d'envoyer à un serveur d'authentification (typiquement RADIUS) l'adresse MAC de la station connectée à un port, en guise de « login/password ». Si l'adresse MAC est connue (authentification réussie), le serveur pourra envoyer au switch le numéro de VLAN attaché à la station. Attention, tous les switches 802.1q ne savent pas forcément réaliser cette opération.

Cette méthode est plus souple, puisqu'une station donnée pourra se connecter sur n'importe quel port, elle se retrouvera toujours sur le VLAN qui lui convient.

Il est possible d'utiliser cette méthode avec autre chose que l'adresse MAC (login/password, certificat x509, smartcard...), il faudra alors mettre en œuvre un « supplicant » sur la station. Nous ne verrons pas cette possibilité dans l'étude du réseau filaire, nous nous contenterons des adresses MAC.

2-2 - Manip VLAN

2-2-1 - Oui, mais au niveau 3 ?



Dans notre exemple, le switch est configuré pour supporter deux VLAN, respectivement d'ID 1 et 2. Les ports verts appartiennent au VLAN d'ID 1 et les ports bleus au VLAN d'ID 2. Aucun de ces ports n'a besoin d'être « tagué » puisqu'ils n'appartiennent qu'à un seul VLAN.

En revanche, sur les ports qui vont véhiculer les trames des deux VLAN, au moins l'un des deux devra être « tagué » au passage de ces ports. Encore une fois, c'est l'interface d'administration du switch qui permettra de réaliser cette configuration. Disons pour fixer les idées que le VLAN vert, d'ID 1 ne sera pas marqué et que le VLAN bleu, d'ID 2 le sera, sur les ports du « trunk » VLAN 1 + VLAN 2.

Pratiquement, admettons que le VLAN 1 supporte un réseau IP 192.168.10.0/24 et que le VLAN 2 soit adressé en 192.168.11.0/24.

Sur le routeur (une machine Debian Etch), nous devons configurer l'unique interface Ethernet physique de manière à ce qu'elle présente deux interfaces IP, chacune pour un VLAN. Il nous faut d'abord installer le paquetage « vlan ».

```
~# apt-get install vlan
Lecture des listes de paquets... Fait
Construction de l'arbre des d&#233;pendances... Fait
....
```

Nous devons également monter le module noyau qui permet de gérer les tags :

```
~# modprobe 8021q
```

Nous venons d'installer les outils nécessaires à la gestion des VLAN (le paquetage VLAN et le montage du module 8021q).

Nous devons maintenant créer une interface réseau virtuelle, qui sera chargée de traiter le VLAN bleu d'ID 2 (celui qui est « tagué »). La commande « vconfig » va le permettre :

```
~# vconfig add eth0 2
Added VLAN with VID == 2 to IF -:eth0:-
```

Vérifions avec la commande ifconfig :

```
~# ifconfig -a
eth0      Lien encap:Ethernet  HWaddr 00:20:18:54:99:F9
          inet adr:192.168.10.1  Bcast:192.168.10.255  Masque:255.255.255.0
          adr inet6: fe80::220:18ff:fe54:99f9/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1631 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1233 errors:0 dropped:0 overruns:0 carrier:0
          collisions:2 lg file transmission:1000
          RX bytes:993172 (969.8 KiB)  TX bytes:118423 (115.6 KiB)
          Interruption:11 Adresse de base:0xa800

 eth0.2    Lien encap:Ethernet  HWaddr 00:20:18:54:99:F9
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:332 (332.0 b)  TX bytes:0 (0.0 b)
...
```

L'argument « -a » est nécessaire pour visualiser les interfaces « down ». eth0.2 existe maintenant, mais n'est pas encore montée.

Nous disposons maintenant sur notre Debian d'un unique adaptateur Ethernet (eth0) qui pourra recevoir nativement le VLAN bleu, puisqu'il n'a pas de tag, et un adaptateur virtuel (eth0.2) qui traitera les trames Ethernet du VLAN vert, d'ID 2. Reste à fixer une adresse IP à eth0.2 et à la monter :

```
~# ip addr add 192.168.11.1/24 broadcast 192.168.11.255 dev eth0.2
~# ifconfig eth0.2 up

~# ifconfig
eth0      Lien encap:Ethernet  HWaddr 00:20:18:54:99:F9
          inet adr:192.168.10.1  Bcast:192.168.10.255  Masque:255.255.255.0
          adr inet6: fe80::220:18ff:fe54:99f9/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1631 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1233 errors:0 dropped:0 overruns:0 carrier:0
          collisions:2 lg file transmission:1000
          RX bytes:993172 (969.8 KiB)  TX bytes:118423 (115.6 KiB)
          Interruption:11 Adresse de base:0xa800

eth0.2    Lien encap:Ethernet  HWaddr 00:20:18:54:99:F9
          inet adr:192.168.11.1  Bcast:192.168.11.255  Masque:255.255.255.0
          adr inet6: fe80::220:18ff:fe54:99f9/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:0 (0.0 b)  TX bytes:344 (344.0 b)
```

La dernière étape consistera à :

- vérifier que le kernel autorise le routage ;
- écrire éventuellement des règles iptables pour le filtrage de paquets, en fonction des besoins.

Cette méthode offre l'avantage de pouvoir réaliser facilement des manipulations pour la mise en œuvre de la solution, mais offre en revanche l'inconvénient d'être totalement manuelle. Il est bien sûr possible d'arranger ça avec un script

bien placé, mais il existe sur Debian (et probablement aussi sur d'autres distributions) une autre solution, qui passe par le fichier de configuration des interfaces (/etc/network/interfaces), dont voici un exemple :

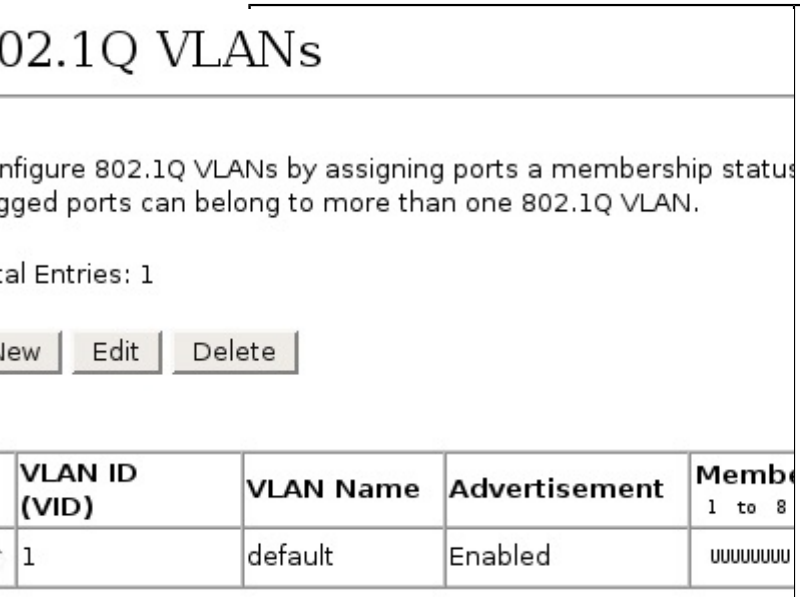
```
...
auto eth0
iface eth0 inet static
    address 192.168.10.1
    netmask 255.255.255.0
    network 192.168.10.0
    broadcast 192.168.10.255

auto vlan2
iface vlan2 inet static
    address 192.168.11.1
    netmask 255.255.255.0
    broadcast 192.168.11.255
    vlan_raw_device eth0
...
```

Nous aurons alors une interface vlan2 en lieu et place de eth0.2, mais qui remplira exactement le même rôle. Pour éviter d'éventuels problèmes de montage du module 8021q, autant l'ajouter dans le fichier /etc/modules.

2-2-2 - Mise en œuvre

La manipulation qui suit est faite avec un switch de type D-Link DES-3326s. Lorsque nous configurons notre switch, nous créons plusieurs VLAN. Chacun de ces VLAN dispose d'un VID.

Ensuite, nous attribuons chaque port à un VLAN particulier. Encore une fois, un port (ou plus) peut appartenir à plusieurs VLAN. Voyons ceci sur l'interface d'administration du D-Link DES-3326s :


02.1Q VLANs

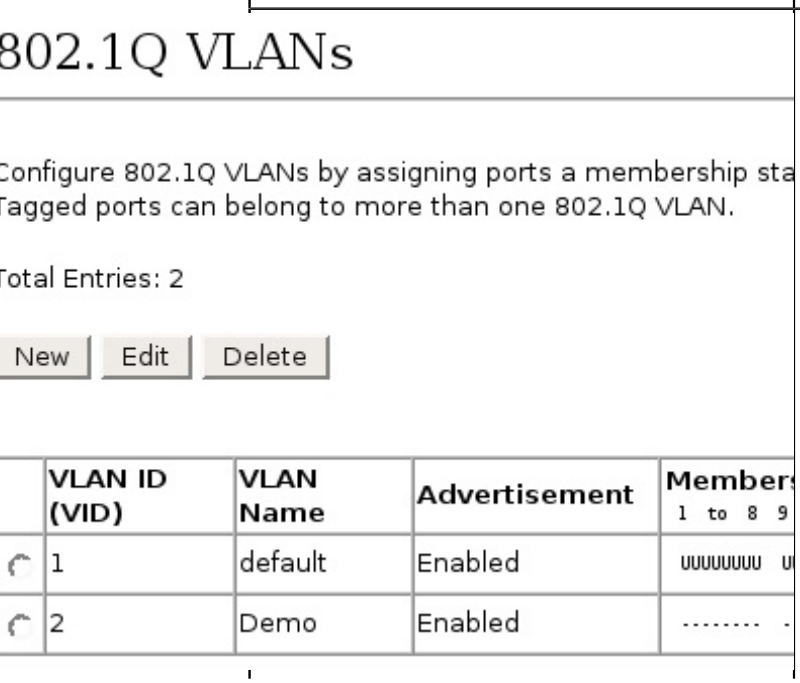
nfigure 802.1Q VLANs by assigning ports a membership status
gged ports can belong to more than one 802.1Q VLAN.

tal Entries: 1

New | Edit | Delete

	VLAN ID (VID)	VLAN Name	Advertisement	Membe 1 to 8
	1	default	Enabled	UUUUUUUU

Nous constatons qu'ici, il n'existe pour l'instant qu'un seul VLAN, de VID 1. Tous les ports de 1 à 24 lui sont assignés de façon « untagged ».


802.1Q VLANs

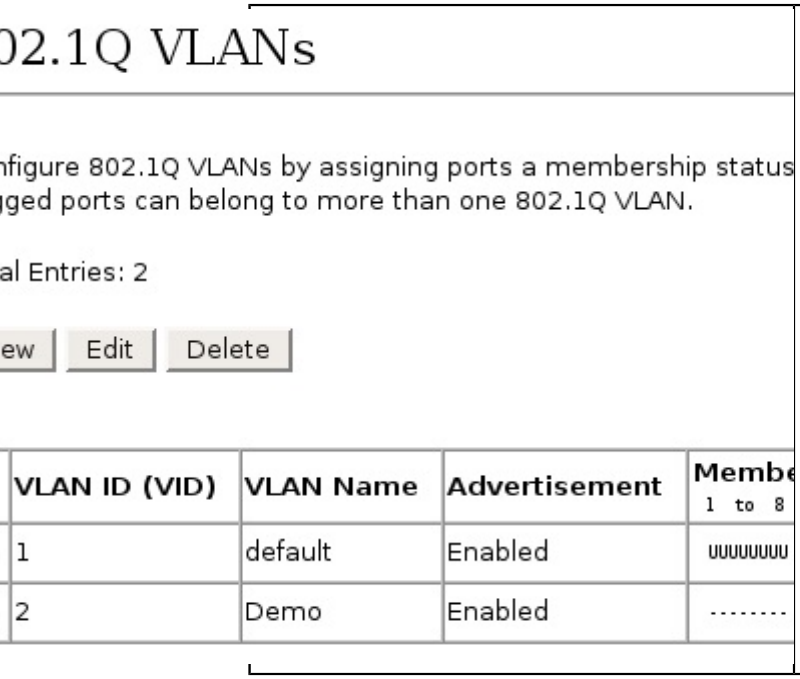
Configure 802.1Q VLANs by assigning ports a membership sta
Tagged ports can belong to more than one 802.1Q VLAN.

Total Entries: 2

New | Edit | Delete

	VLAN ID (VID)	VLAN Name	Advertisement	Members 1 to 8 9
○	1	default	Enabled	UUUUUUUU U
○	2	Demo	Enabled	-------- -

Nous allons commencer par créer un second VLAN de démonstration, puis nous nous occuperons des ports laissés libres (25 et 26). Comme nous le voyons, le second VLAN est créé, mais pour l'instant, aucun port ne lui est assigné.



Enfin, nous assignons le port 25 au VLAN 2 de façon « untagged », puis le port 26 aux deux VLAN. Ici, il faudra utiliser les tags : * « untagged » dur le VLAN 1, * « tagged » sur le VLAN 2

Finalement, si nous relions le port 26 à notre Debian Sarge sur son interface physique eth0, nous aurons la possibilité de router les paquets entre les VLAN 1 et 2.

2-3 - VLAN niveau 2

2-3-1 - Position du problème

Le principe des VLAN étant compris, la dernière étape va consister à mettre en œuvre une commutation automatique des ports du switch sur l'un ou l'autre VLAN, suivant que la machine qui s'y connecte sera authentifiée ou non.

Conformément au cahier des charges, nous utilisons simplement l'adresse MAC de la machine, ce qui évitera d'avoir à installer sur chaque client un système d'authentification plus sophistiqué (un certificat, par exemple, comme nous le verrons avec WPA2-TLS). Cette méthode n'est pas parfaite, loin de là, dans la mesure où une adresse MAC peut être falsifiée, mais elle a le mérite d'être simple à mettre en œuvre.

Il nous faudra tout de même disposer de switches capables d'interroger un serveur RADIUS, en lui envoyant l'adresse MAC du client en guise de nom d'utilisateur et de mot de passe. Nous utilisons ici un switch HP Procurve 2650.

2-3-2 - Remarques à propos du ProCurve 2650

Lorsqu'il sort de sa boîte, ce switch est configuré avec un seul VLAN, nommé « DEFAULT_VLAN » (et qui est aussi le « PRIMARY_VLAN »). Tous les ports du switch sont affectés à ce VLAN, si bien que sans aucune configuration particulière, ce switch fonctionnera comme un switch de base.

Pour le configurer, plusieurs solutions sont proposées, à commencer par une liaison série RS232 (gardez au moins un vieux PC), qui est initialement le seul moyen possible pour accéder à la configuration (**In the factory default configuration, the switch has no IP (Internet Protocol) address and subnet mask, and no passwords. In this state, it can be managed only through a direct console connection**).

Par la suite, nous pourrons accéder au switch par le réseau, via Telnet, un miniserveur web embarqué (mais vraiment minimaliste), ou même ssh. En effet les switches administrables peuvent recevoir une adresse IP pour accéder à l'administration par le réseau. Sur ce modèle de switch, nous pourrons même assigner une adresse IP par VLAN,

ce qui n'est absolument pas nécessaire, voire dangereux. Comme notre propos est plutôt de parler des VLAN, nous passerons ces détails sordides.

Nous supposons donc que la configuration de base du switch est faite, et principalement la configuration IP du DEFAULT_VLAN :

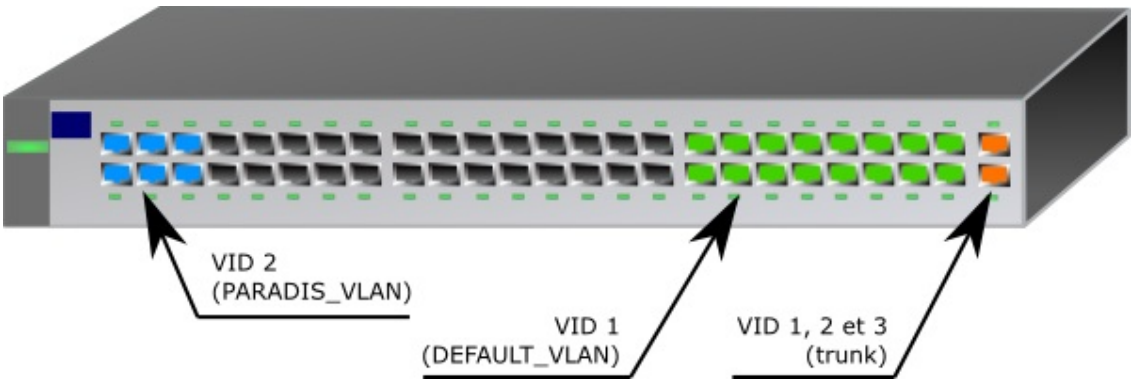
```
Internet (IP) Service

  IP Routing : Disabled

  Default Gateway : 192.168.10.1
  Default TTL     : 64
  Arp Age         : 20

  VLAN         | IP Config  IP Address      Subnet Mask
  ------------ + ---------- --------------- ---------------
  DEFAULT_VLAN | Manual     192.168.10.11   255.255.255.0
```

Donc, nous avons un « DEFAULT_VLAN » qui est aussi le « PRIMARY_VLAN », et qui contient tous les ports du switch. Il y a quelques contraintes à connaître à propos de ces deux VLAN :

- DEFAULT_VLAN ne peut pas être supprimé et a le VID 1, en revanche, rien n'interdit de ne lui assigner aucun port ;
- PRIMARY_VLAN est nécessaire à certaines fonctions d'administration que nous n'utiliserons pas forcément ici, comme le pseudoempilage de switches. Cette fonction de PRIMARY_VLAN peut être assignée à n'importe quel VLAN existant, pas forcément au DEFAULT_VLAN, mais il doit exister. Nous laissons la configuration par défaut.

Nous n'allons conserver que quelques ports, dont les deux ports 1GB/s sur DEFAULT_VLAN, laisser PRIMARY_VLAN dessus, tous les autres ports étant réservés à deux autres VLAN qu'il nous reste à créer :

- PARADIS_VLAN, de VID 2, qui accueillera le LAN des hôtes connus ;
- ENFER_VLAN de VID 3, qui accueillera toutes les machines que l'on ne sait pas identifier.



Nous allons restreindre le DEFAULT_VLAN aux ports 33-50 (sans tag). Ce VLAN nous servira à administrer le switch, à accueillir les DNS, DHCP, RADIUS et autres services « administratifs ».

Les ports 1 à 6 seront assignés au PARADIS_VLAN de façon statique (sans tag). Nous y placerons par la suite les ressources du réseau à offrir aux stations « connues ».

Comme ce switch ne supporte pas d'avoir des ports assignés à aucun VLAN, nous allons mettre les ports 7 à 32 dans le ENFER_VLAN. Nous reviendrons éventuellement sur ce choix plus tard.

Les ports 49 et 50 seront quant à eux assignés aux trois VLAN. Bien entendu, ici, il faudra utiliser les tags. L'un des deux ports servira à connecter le routeur et l'autre sera réservé aux extensions futures (un second switch, par exemple).

Finalement, la (magnifique) interface web nous montrera ceci :

VLAN ID	VLAN Name	VLAN Type	Tagged Ports	Untagged Ports	Forbid Ports	Auto	
1	DEFAULT_VLAN (Primary)	STATIC	(STATIC) None (GVRP) None	33-50	None	None	Modify
2	PARADIS_VLAN	STATIC	(STATIC) 49-50 (GVRP) None	1-6	None	None	Modify
3	ENFER_VLAN	STATIC	(STATIC) 49-50 (GVRP) None	7-32	None	None	Modify

Mais qu'importe la beauté lorsque le travail est bien fait, n'est-ce pas ?

Notez que tout le travail fait jusqu'ici a pu l'être par l'entremise de cette interface. Pour la suite, ce sera quelque peu différent.

2-3-2-1 - Choix VLAN suivant l'authentification

Il nous reste maintenant à expliquer au switch que les ports 7 à 32 doivent être attribués au VLAN 2 ou au VLAN 3, suivant que l'authentification par adresse MAC sera réussie ou non. Pour réaliser cette opération, l'interface web n'est pas exploitable, le menu en mode texte via Telnet, ssh ou rs232 non plus. Il nous faut passer par le moyen le plus rustique, la ligne de commande. C'est la documentation qui va venir à notre secours,

- Access Security Guide : SWITCH 2600 Series SWITCH 2600-PWR Series SWITCH 2800 Series SWITCH 4100 Series SWITCH 6108
  - Web and MAC Authentication for the Series 2600/2600-PWR and 2800 SWITCHes
    - Configuring MAC Authentication on the SWITCH
      - Configure the SWITCH for MAC-Based Authentication

Voici la procédure. Dans la console :

```
configure
```

(Pour entrer dans le mode configuration)

```
aaa port-access mac-based addr-format multi-colon
aaa port-access mac-based 7-32
```

Vérifiez que vous obtenez un message indiquant que LACP a été désactivé (LACP est un protocole de gestion d'agrégations de liens, qui sort du cadre de cette étude, mais qui est incompatible avec les VLAN de niveau 2).

- La première ligne indique que les adresses MAC doivent être envoyées sous la forme xx:yy:zz:aa:bb:cc. D'autres formats sont possibles, tout dépendra de la façon qui nous est la plus commode pour collecter et enregistrer ces adresses MAC dans notre futur RADIUS.
- La seconde ligne indique que les ports 7 à 32 seront assignés à un VLAN en fonction de l'adresse MAC du client connecté.

```
aaa port-access mac-based 7-32 auth-vid 2
```

```
aaa port-access mac-based 7-32 unauth-vid 3
```

- La première ligne indique que les ports 7 à 32 devront être assignés au VLAN de VID 2 si l'authentification est réussie.
- La seconde ligne indique que les ports 7 à 32 devront être assignés au VLAN de VID 3 si l'authentification est ratée.

```
show port-access 7-32 mac-based config
```

Cette ligne permet de vérifier que notre configuration est bien enregistrée :

```
Port Access MAC-Based Configuration
  MAC Address Format : multi-colon

                Client Client Logoff    Re-Auth   Unauth   Auth
  Port  Enabled Limit  Moves  Period    Period    VLAN ID  VLAN ID
  ----- ------- ------ ------ --------- --------- -------- --------
  7     Yes     1      No     300       0         3        2
  8     Yes     1      No     300       0         3        2
  9     Yes     1      No     300       0         3        2
  10    Yes     1      No     300       0         3        2
  11    Yes     1      No     300       0         3        2
  12    Yes     1      No     300       0         3        2
  13    Yes     1      No     300       0         3        2
  14    Yes     1      No     300       0         3        2
  15    Yes     1      No     300       0         3        2
  16    Yes     1      No     300       0         3        2
  17    Yes     1      No     300       0         3        2
  18    Yes     1      No     300       0         3        2
  19    Yes     1      No     300       0         3        2
  20    Yes     1      No     300       0         3        2
  21    Yes     1      No     300       0         3        2
  22    Yes     1      No     300       0         3        2
  23    Yes     1      No     300       0         3        2
  24    Yes     1      No     300       0         3        2
  25    Yes     1      No     300       0         3        2
  26    Yes     1      No     300       0         3        2
  27    Yes     1      No     300       0         3        2
  28    Yes     1      No     300       0         3        2
  29    Yes     1      No     300       0         3        2
  30    Yes     1      No     300       0         3        2
  31    Yes     1      No     300       0         3        2
  32    Yes     1      No     300       0         3        2
```

C'est correct. Nous pouvons écrire le tout en mémoire :

```
write memory
```

En ce qui concerne « l'auth-vid », ce paramètre pourra éventuellement être écrasé par une valeur renvoyée, en cas d'authentification réussie, par le serveur RADIUS, ce qui apporte plus de souplesse si l'on doit disposer de plus de deux VLAN suivant les clients.

C'est presque fini, mais pas tout à fait. Notre switch ne devinera pas tout seul qui est notre serveur RADIUS, il faut le lui indiquer.

Notre futur serveur RADIUS aura l'adresse 192.168.10.2. Comme nous le verrons plus loin, il faudra définir une clé de chiffrement partagée entre le serveur et ses « clients ». Nous allons choisir quelque chose de difficile à trouver : chut !

```
radius-server host 192.168.10.2 key epikoi
```

Vérifions :

```
show radius
```

```
Status and Counters - General RADIUS Information

  Deadtime(min) : 0
  Timeout(secs) : 5
  Retransmit Attempts : 3
  Global Encryption Key :

                 Auth  Acct
  Server IP Addr  Port  Port  Encryption Key
  --------------- ----- ----- --------------------------------
  192.168.10.2    1812  1813  epikoi
```

C'est OK

```
write memory
```

Il ne reste plus qu'à mettre en place le serveur RADIUS, que nous connecterons sur le DEFAULT_VLAN (que nous avons configuré pour un réseau IP 192.168.10.0/24).

2-3-2-2 - Revenir en arrière

Il peut se faire par la suite que l'on souhaite redonner à des ports leur fonctionnement « normal » après les avoir attribués à une authentification par RADIUS. Pour ce faire, il faudra sur ces ports successivement :

- inhiber l'authentification par adresse MAC ;
- supprimer le VID en cas d'authentification réussie ;
- supprimer le VID en cas d'authentification ratée.

Supposons que l'on veuille remettre les ports 30 à 32 dans leur état « normal » :

```
configure
no aaa port-access mac-based 30-32
no aaa port-access mac-based 30-32 auth-vid
no aaa port-access mac-based 30-32 unauth-vid
write memory
```

Vérifions :

```
show port-access 7-32 mac-based config

Port Access MAC-Based Configuration
  MAC Address Format : multi-colon

                 Client Client Logoff    Re-Auth   Unauth   Auth
  Port  Enabled  Limit  Moves  Period    Period    VLAN ID  VLAN ID
  ----- -------- ------ ------ --------- --------- -------- --------
 ...
  24    Yes      1      No     300       0         3        2
  25    Yes      1      No     300       0         3        2
  26    Yes      1      No     300       0         3        2
  27    Yes      1      No     300       0         3        2
  28    Yes      1      No     300       0         3        2
  29    Yes      1      No     300       0         3        2
  30    No       1      No     300       0         0        0
  31    No       1      No     300       0         0        0
  32    No       1      No     300       0         0        0
```

3 - Réseaux sans fil sécurisés

Le Wi-Fi (Wireless Fidelity) est à la mode. Il faut dire que la solution est souvent alléchante, qu'elle apporte une réponse décisive aux problèmes de connectivité « nomade », à l'heure où le parc des ordinateurs portables se développe.

Wi-Fi trouve aujourd'hui, plus ou moins légitimement, sa place aussi bien en entreprise que chez le particulier, souvent au mépris des contraintes technologiques des liaisons radio, et aussi au mépris des règles les plus élémentaires de sécurité.

Ce chapitre fait suite à la **présentation générale du WI-FI**, exposée plus haut dans ce site. Nous nous intéresserons ici plus particulièrement au problème de la sécurité, dans un réseau d'entreprise.

3-1 - Introduction

3-1-1 - Les différents types de risque

D'une manière générale, dans le domaine des systèmes d'information, nous sommes confrontés à divers types de risques, qui découlent tous d'une intrusion sur notre réseau, hormis les DoS (Deny of Service) qui n'ont pour but que de « simplement » rendre le système inutilisable. Ce dernier aspect ne sera pas traité ici, la façon que nous avons d'utiliser le spectre radio n'offrant que très peu de moyens de nous protéger de la méthode d'attaque la plus simple : le brouillage radio. Tout de même, il faut en parler un tout petit peu, dans la mesure où la promiscuité de réseaux sans fil, montés de façon anarchique, peut arriver de façon involontaire à cet état de fait.

3-1-1-1 - Intrusion en vue d'une compromission des postes

L'objet de ce type d'intrusion est de prendre possession d'un équipement du réseau, soit pour en extraire des données intéressant le pirate, soit pour constituer une armée de « zombies », avec pour objectif final, par exemple, de réaliser un DDoS (Distribued Deny of Service) sur une cible stratégique, ou encore de déclencher un envoi massif de spams à un instant donné, pour prendre de court les stratégies mises en place pour contenir ce fléau.

Ce type d'intrusion se réalise généralement depuis l'internet, ce n'est pas un problème spécifique aux réseaux sans fil. Tout de même, dans une certaine mesure, ces derniers sont un peu plus exposés, dans la mesure où ils sont plus souvent destinés à accueillir des postes clients dont l'état sanitaire n'est pas maîtrisé par les responsables du système d'information. Si un invité se connecte avec un poste déjà compromis par un ver, ce ver pourra à son tour contaminer les autres postes du réseau, suivant le type de propagation qu'il utilise et suivant les protections mises en place sur les postes du réseau. Si l'on peut considérer que généralement les réseaux sont assez bien protégés d'attaques venant de l'extérieur par un système de « firewall », ils le sont en général beaucoup moins contre les attaques venant de l'intérieur, qui est a priori considéré comme une zone « de confiance ».

3-1-1-2 - Intrusion en vue d'exploitation d'un accès à l'internet

Un pirate peut chercher à exploiter votre connexion à l'internet, en vue d'effectuer des attaques sur d'autres systèmes. Cette méthode lui offre beaucoup de confort, puisque si ses activités sont repérées, dans les faits, ce ne sera pas lui, mais vous qui serez inquiété, voire condamné.

Ne sous-estimez pas ce risque, surtout avec les réseaux sans fil dont on ne peut maîtriser parfaitement la portée !

3-1-1-3 - Intrusion « passive » en vue d'extraire des informations

Pour récupérer des informations « intéressantes », un pirate peut se contenter d'écouter de façon passive les échanges d'information, pour en retirer (en temps réel ou non) des éléments ayant pour lui une certaine valeur.

Sur un réseau sans fil, il est très facile de mettre en place des systèmes d'écoute dans ce but. Le pirate se contente d'écouter les conversations et de les enregistrer. Il n'agit pas sur vos équipements et ne laisse aucune trace, surtout sur un réseau sans fil.

3-1-2 - Les parades

Il existe un certain nombre de règles de base, qu'il faudra mettre en œuvre pour tenter de minimiser ces risques.

3-1-2-1 - Savoir qui est présent sur votre réseau

L'authentification des clients d'un réseau est bien sûr un élément fondamental. Les adresses IP comme les adresses MAC ne constituent plus, depuis longtemps, un moyen efficace. Ces adresses peuvent très facilement être usurpées sur un LAN. C'est évidemment un point très sensible sur les réseaux sans fil, ça l'est de plus en plus sur un réseau filaire, avec l'apparition massive d'ordinateurs portables.

La méthode sans doute la plus aboutie consiste à utiliser des certificats, principalement sur les serveurs, mais aussi sur les clients. Cependant, l'usage des certificats sur les clients est une opération lourde. Il faut créer un certificat par client, maintenir une base de données des certificats autorisés et aussi des certificats révoqués.

Rappelons-le, un certificat contient schématiquement une partie publique et une partie privée. La partie publique peut être récupérée par tout tiers désirant entrer en contact avec le possesseur du certificat, la partie privée doit rester confidentielle. Dans chacune de ces parties, il y a une clé de chiffrement. Pour plus de détails, voyez le chapitre sur la cryptographie.

Il est possible sur les clients de remplacer le certificat par un autre moyen d'authentification, allant du simple couple « nom d'utilisateur/mot de passe », à l'usage d'une carte à puce ou d'une empreinte biométrique.

La dernière méthode est probablement la plus sûre et a fait déjà rêver bon nombre d'auteurs de science-fiction. La carte à puce est très proche du certificat et reste assez lourde à gérer.

Le couple « utilisateur/mot de passe » reste le plus simple, et peut souvent être considéré comme suffisamment sûr, à la condition que le mot de passe ne soit pas évident et que l'échange de ce mot de passe ne soit pas facilement « sniffable », c'est-à-dire récupérable par une écoute passive. Bien sûr, il faut aussi que les utilisateurs aient pris conscience des risques qu'il y a à partager leur identifiant avec un (ou plusieurs) tiers.

En conclusion, nous devrons mettre en place un système strict, qui n'autorise qu'une personne dûment authentifiée à rejoindre notre réseau, en minimisant les risques d'usurpation d'identité.

Ceci évitera qu'un intrus profite d'une faille du système pour :

- compromettre des machines du réseau ;
- exploiter votre accès à l'internet pour effectuer des opérations malveillantes ;
- identifier éventuellement, toute personne (authentifiée) qui se livrerait à des opérations malveillantes.

3-1-2-2 - Cacher les informations échangées

Le principe de base est de chiffrer les échanges, mais il faut être bien conscient qu'un chiffrement n'est pas (jamais ?) incassable. Tout est question de temps.

Ainsi, dans un système de chiffrement bien réalisé, il faut tenir compte de quelques éléments importants :

- combien de temps les informations doivent-elles rester confidentielles ? ;
- quel préjudice subirons-nous lorsque ces informations seront découvertes ?

Prenons quelques exemples simples...

3-1-2-2-1 - Découverte d'un « login » autorisé

Si le temps nécessaire à la découverte d'un moyen d'authentification est supérieur à la durée de vie de ce dernier, le moyen d'authentification est considéré comme sûr et il n'y a pas de préjudice subi.

3-1-2-2-2 - Découverte d'une information « stratégique » caduque

Une information communiquée à un tiers de confiance doit rester confidentielle pendant une semaine, puis elle sera divulguée au public, ou n'aura plus aucune valeur. Le temps nécessaire au déchiffrage de cette information est supérieur à une semaine. Le système de protection est sûr et il n'y a pas de préjudice.

3-1-2-2-3 - Découverte d'une information « stratégique » à long terme

Une information communiquée à un tiers de confiance doit rester confidentielle le plus longtemps possible. Le préjudice subi en cas de découverte sera d'autant moindre que le temps de découverte sera long. Ici, nous avons un problème. Si l'information chiffrée est récupérée, un jour où l'autre, elle sera déchiffrée et préjudice il y aura. Le jeu consiste donc à mettre en place un système qui « tienne » le plus longtemps possible, à mettre en œuvre divers procédés pour tenter de minimiser les risques d'écoute passive, de mettre hors d'état de nuire un pirate repéré avant qu'il n'ait eu le temps de déchiffrer les données, etc. Le tout, jusqu'à ce que le préjudice ne soit plus suffisant pour justifier le coût de toutes ces mesures.

Clairement, les besoins du particulier ne sont pas les mêmes que ceux d'une banque (le particulier qui utilise les moyens de commerce en ligne doit tout de même envisager ces trois types d'intrusion, une carte de crédit, par exemple, ayant maintenant deux ans, voire plus, de durée de vie).

3-1-2-3 - Assurer un minimum de sécurité au client

Authentifier le client, c'est bien. Mais le client ne doit-il pas être lui aussi protégé ? Le client peut, pour sa sécurité, pouvoir s'assurer qu'il se connecte bien au réseau auquel il pense se connecter. Il n'est pas très difficile, sur les réseaux sans fil, de présenter un « SSID » usurpé. Le client doit pouvoir, si besoin est, authentifier le réseau auquel il se connecte.

3-1-2-3-1 - Bon gros avertissement à l'usage des clients

Un client mal configuré peut très bien se connecter tout seul à un réseau non protégé. Pour peu que le SSID de ce réseau soit le même que le vôtre, le client peu au fait des règles du WI-FI, pourra croire qu'il est connecté à votre réseau, alors que ce n'est pas le cas. Ne riez pas, il existe des gens qui utilisent le WI-FI sans rien y comprendre, j'en ai rencontré.

3-2 - Solutions

Comme souvent en informatique, des solutions ont été déployées sans avoir vraiment prévu les risques associés. Mais nous vivons dans un monde de brutes, et tôt ou tard, les besoins de protection nous rattrapent.

3-2-1 - WEP

(Wired Equivalent Privacy). Basée sur une solution de clé de chiffrement statique partagée par tous les membres d'un même réseau Wi-Fi, avec un algorithme de chiffrement devenu aujourd'hui plutôt faible, il a été clairement démontré que cette protection n'en est plus une, et nous ne nous attarderons pas dessus.

Sachez simplement qu'avec des outils comme **airodump, aircrack et aireplay**, il est possible, de découvrir une clé WEP en quelques dizaines de minutes, voire moins.

3-2-2 - WPA

Devant la panique générée par la découverte des failles de WEP, et alors même que les différents partenaires travaillaient à l'élaboration d'une norme destinée à sécuriser les réseaux sans fil, il a fallu mettre en place un procédé de secours : le « Wifi Protected Access », issu des travaux non encore aboutis de la norme qui est depuis finalisée et connue sous le doux nom de 802.11i.

WPA apparaît comme un ensemble de rustines logicielles, destinées à boucher les plus gros trous de sécurité du WEP, tout en ayant comme contrainte de pouvoir fonctionner sur le matériel existant. Il est donc possible en général d'exploiter WPA sur du matériel conçu pour WEP.

WPA n'est finalement qu'un compromis, acceptable si l'on doit intégrer à son réseau du matériel ancien, ne supportant pas les méthodes de chiffrement préconisées par 802.11i.

3-2-3 - WPA2

Ce n'est rien de plus que l'appellation commerciale de la norme 802.11i. Cette norme a été finalisée en 2004 et donc, le matériel n'a des chances d'être compatible que s'il a été conçu aux environs de cette date. Le chiffrement préconisé nécessite la présence d'un composant matériel dédié.

3-2-4 - Personnal ou Enterprise ?

WPA comme WPA2 peuvent s'utiliser de deux manières.

3-2-4-1 - Mode « personnel »

La méthode dite « personnelle », c'est-à-dire celle qui est préconisée pour les particuliers disposant d'un petit réseau peu stratégique, fait appel à une clé de chiffrement partagée, la PSK (Pre Shared Key). Cette clé, au contraire de WEP, ne sert pas directement au chiffrement des données. Elle sert de base à la création de clés dérivées, qui sont non seulement différentes pour chaque session (deux utilisateurs d'un même réseau, même s'ils disposent de la même PSK, utiliseront des clés de session différentes), mais encore d'un usage limité dans le temps. Ces clés sont renégociées fréquemment en cours de session, ce qui rend leur découverte nettement plus difficile.

Reste que toute la sécurité repose sur la PSK qui est commune à tous les utilisateurs, et qu'un secret, plus il est partagé, moins il devient secret.

Cette solution (actuellement WPA2-PSK) reste tout à fait acceptable dans le cas d'un réseau sans fil domestique, à la condition d'utiliser une PSK non évidente. Pratiquement, la clé PSK est une suite numérique, calculée à partir d'une « phrase secrète » en ASCII et du SSID du point d'accès. Il faut une phrase secrète qui satisfasse les règles de non-évidence classiques (« Ahjk7£&£_#1-[qht » vaut mieux que « SalutLesMecs », votre nom, votre date de naissance, celle de la petite dernière...).

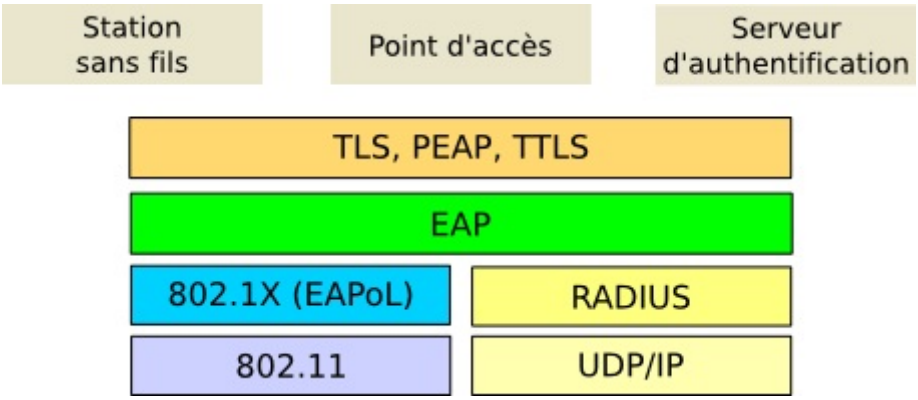
3-2-4-2 - Mode « entreprise »

Ici, nous avons plus de moyens techniques et surtout, plus de besoins de sécurité. Il n'y a pas de clé partagée. Nous ferons appel à un système d'authentification centralisé, allié à un protocole défini par la norme 802.1x.

Typiquement, nous aurons à mettre en place un serveur d'authentification de type RADIUS (Remote Authentication Dial-In User Service) et un point d'accès supportant la norme 802.1x (finalisée en 2001).

Il n'y a, encore une fois, aucun secret partagé, tout le système cryptographique sera construit pendant et après le processus d'authentification, comme nous allons le voir en détail.

3-2-5 - Techniquement...



Nous allons avoir besoin de plusieurs briques logicielles, dont certaines nous sont déjà familières, d'autres moins.

Un serveur d'authentification est connecté au point d'accès, par réseau filaire. Le type de serveur d'authentification n'est pas défini par les normes, mais clairement, c'est le protocole RADIUS qui est sous-entendu, c'est celui que nous utiliserons. RADIUS est un protocole applicatif, qui s'appuie sur UDP, IP et Ethernet.

Entre le point d'accès et la station cliente, nous sommes sur du 802.11 (WI-FI). Au-dessus, nous trouverons 802.1x.

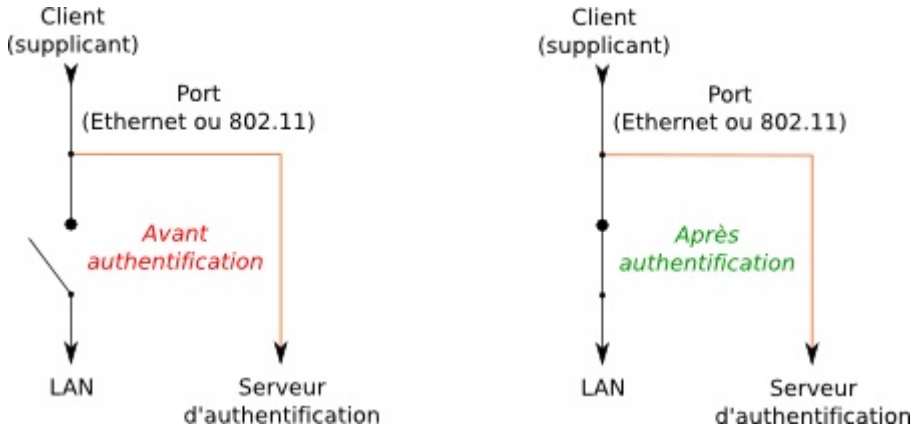
EAP (Extensible Authentication Protocol) va passer au-dessus de 802.1x dans les airs et au-dessus de RADIUS sur les fils, pour transporter le mode d'authentification choisi (TLS, PEAP, TTLS ou d'autres encore, mais considérés comme moins sûrs).

Dans la suite de cet exposé, nous nous intéresserons surtout à TLS (Transport Layer Security), qui fait appel à des certificats x509 sur le serveur RADIUS comme sur le client.

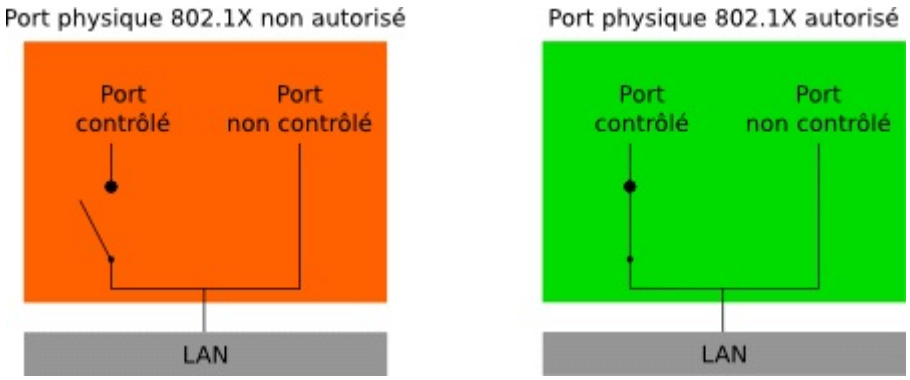
3-3 - Authenticator

3-3-1 - Définition

L'authenticator utilise le protocole 802.1x. C'est un protocole qui a été défini dans le but d'autoriser l'accès physique à un réseau local après une phase d'authentification. Ce protocole peut aussi bien s'appliquer à un « port » physique (un point d'entrée sur un switch, par exemple), que sur un « port » virtuel (l'attachement à un point d'accès Wi-Fi).



Il est peut-être nécessaire de passer un peu de temps à bien comprendre le principe d'un port contrôlé par 802.1x (authenticator). Lorsque le client (supplicant) se connecte (Ethernet) ou s'attache à un point d'accès (802.11), l'accès au LAN lui est fermé. Le seul trafic autorisé sera constitué des échanges entre le supplicant et le serveur d'authentification. Le port ne pourra s'ouvrir sur le LAN sans restrictions qu'une fois l'authentification réussie.



Bien entendu, physiquement, il n'y a qu'un seul port. Ce port physique est composé de deux ports virtuels : * l'un, non contrôlé, mais qui ne laisse passer que le trafic d'authentification (RADIUS), * l'autre, contrôlé, qui est destiné à laisser passer tout le trafic Ethernet, mais qui est fermé jusqu'à ce que l'authentification soit réussie.

802.1x ne peut se jouer qu'à trois personnages :

- le **supplicant**, qui est le client souhaitant se raccorder au réseau ;
- l'**authenticator**, ou **network access server (NAS)** qui est l'équipement sur lequel le « supplicant » désire se connecter ;
- l'**authentication server (AS)**, qui est le serveur d'authentification.

Il est primordial de comprendre qu'une station cliente qui ne sait pas jouer le rôle de « supplicant » ne pourra jamais accéder au LAN. Pour jouer ce rôle, il faut un bout de logiciel spécifique sur le client. Windows XP intègre ce logiciel depuis le SP1. Sous Linux, il faudra installer wpasupplicant, par exemple. La plupart des distributions modernes installent par défaut ce composant.

Typiquement, dans le cadre de notre étude :

- le supplicant sera un PC (GNU/Linux, Mac OS, ou même Windows) ;
- l'authenticator sera un point d'accès Wi-Fi (mais ce pourrait être aussi bien un switch sur un réseau filaire) ;
- l'authentication server sera un serveur RADIUS. (FreeRADIUS 1.1.3, sur une Debian « etch », ou encore sur une Ubuntu 6.10).

Mais 802.1x n'est qu'un support. 802.1x va servir à transporter un protocole d'authentification, comme EAP (Extensible Authentication Protocol). Comme son nom le laisse supposer, EAP est plutôt souple et permet diverses méthodes d'authentification, plus ou moins efficaces. Il faut bien comprendre qu'EAP n'apporte pas à lui seul la moindre authentification, il sert lui-même à supporter diverses méthodes d'authentification.

Nous retiendrons trois de ces méthodes :

- **TLS**, qui nécessite un certificat pour le serveur comme pour le supplicant ;
- **PEAP**, où seul un certificat côté serveur est nécessaire, le supplicant utilisant un couple « utilisateur/mot de passe » ;
- **TTLS**, assez similaire à PEAP, mais non supporté par Microsoft Windows.

Pratiquement, nous ferons du TLS si nous avons le courage de créer un certificat par supplicant, et la liste de révocation qui doit aller avec ; nous ferons du PEAP sinon, puisque c'est possible sur les clients GNU/Linux comme

sur les clients Windows et Mac OS X. Les deux solutions apportent un niveau de sécurité technique que l'on peut, au moment où ces lignes sont écrites, considérer comme suffisant.

Cependant, un couple « utilisateur/mot de passe » valide peut s'obtenir plus facilement qu'un certificat. Aussi préfèrerons-nous utiliser TLS que PEAP (ou TTLS).

3-3-1-1 - Note importante

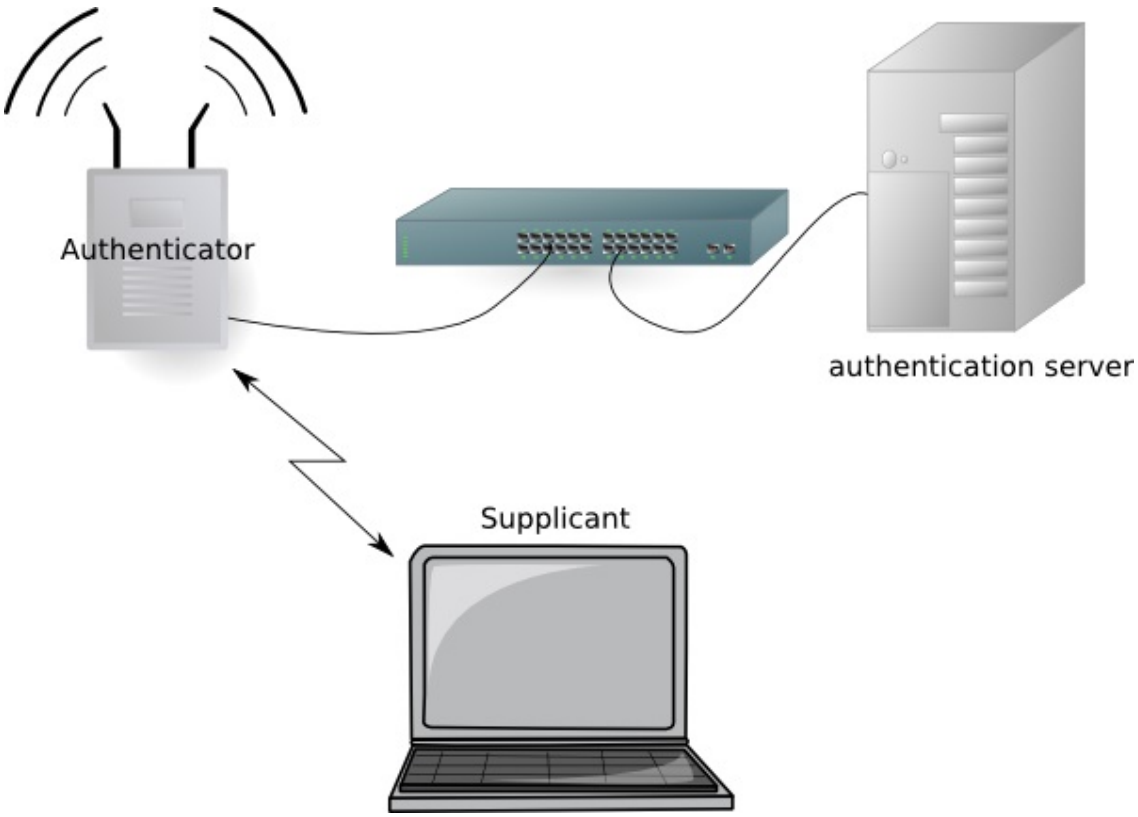
802.1.x n'est pas une solution exempte de failles. Il est indispensable d'utiliser par-dessus des systèmes d'authentification qui permettent :

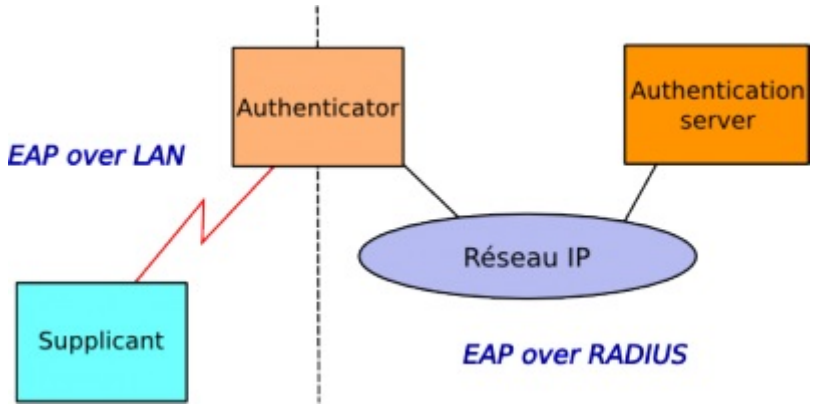
- au supplicant d'être authentifié par le serveur ;
- par la suite, au point d'accès d'authentifier le supplicant ;
- enfin, le supplicant doit également authentifier le point d'accès.

Si les deux derniers points ne sont pas vérifiés à chaque paquet transmis, nous courons le risque de voir un attaquant mettre hors jeu un client dûment authentifié, pour lui voler sa place, ou placer un point d'accès pirate pour tromper le client en cours de session.

TLS, PEAP et TTLS répondent à cette problématique.

3-3-2 - Principe de fonctionnement



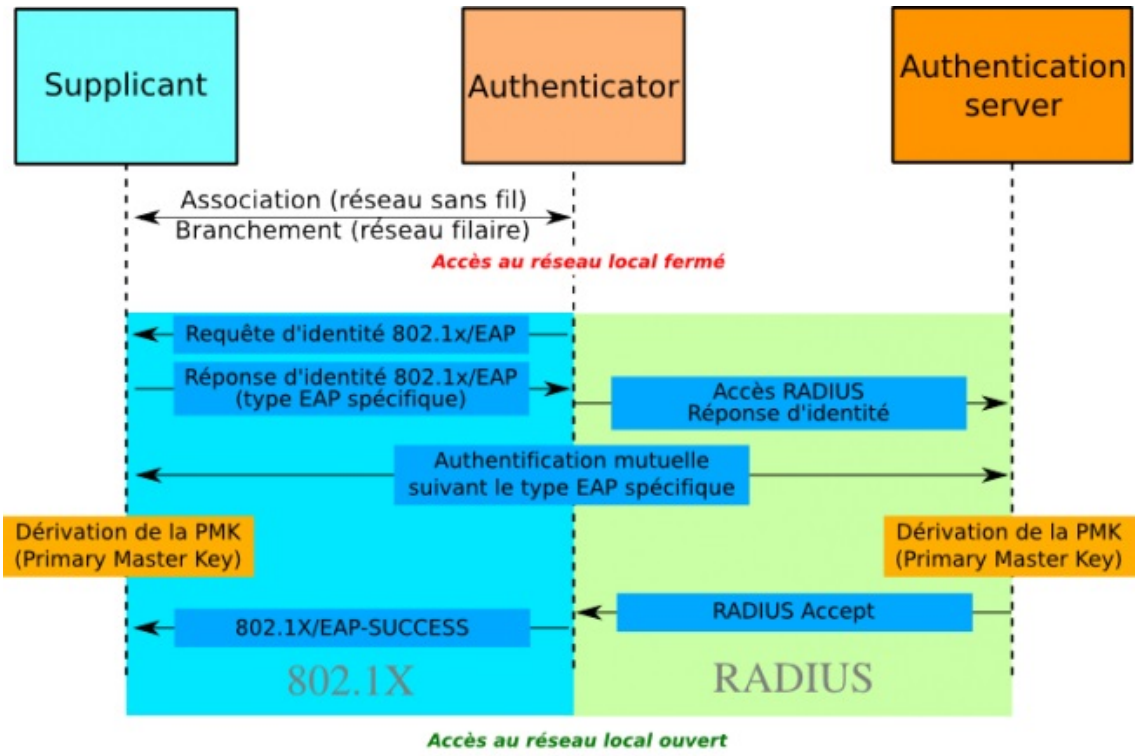


Lorsque le « supplicant » découvre le point d'accès, ce dernier ne lui ouvre pas de port, jusqu'à ce que le « supplicant » soit authentifié par le serveur. Seul le trafic nécessaire à 802.1x sera toléré avant une authentification réussie.

Entre le supplicant et l'authenticator, aussi longtemps que l'authentification n'est pas entièrement réussie (réponse positive du serveur RADIUS), le seul trafic permis est le dialogue d'authentification, le client n'a pas accès au réseau (depuis le temps que je le dis, j'espère maintenant que c'est bien compris).

Entre l'authenticator et le serveur, le trafic se fait classiquement sur UDP/IP, ce qui permet de disposer d'un serveur d'authentification très éloigné, si besoin est. Ce trafic est appelé « EAP over RADIUS ».

D'une manière générale, les échanges se font de la sorte (cas d'une authentification réussie) :



3-3-3 - Authentifications nécessaires

Compte tenu des divers risques introduits (faux serveur d'authentification, faux point d'accès, faux client), il convient de mettre en place des stratégies qui permettent de minimiser les risques de leurre.

3-3-3-1 - Authentification du serveur

Le serveur va utiliser un certificat (sorte de carte d'identité, réputée authentique, sous la responsabilité d'une « Certificate Authority », ou « Autorité de certification »). Le serveur communique au client la partie publique de ce certificat, et le client peut vérifier auprès de la CA que ce certificat est bien valide.

3-3-3-2 - Authentification du client

Le client peut utiliser divers moyens, qui vont du certificat (même procédure que pour le serveur), au bon vieux couple « nom d'utilisateur/mot de passe », en passant par des systèmes à base de carte à puce ou d'empreintes biométriques.

3-3-3-3 - Authentification du point d'accès

Ici, le problème est un peu plus compliqué. Que devons-nous vérifier en réalité ? Nous devons être « sûr » que le point d'accès auquel le client s'est attaché à la suite de la procédure d'authentification ne va pas être remplacé par un point d'accès « voleur » en cours de session (man in the middle).

802.11i (WPA2) offre la technique, actuellement la plus sûre, pour s'affranchir de ce risque. Nous verrons ceci plus loin.

3-4 - WPA2

3-4-1 - Définition

802.11i (WPA2) est une norme tendant à sécuriser un accès sans fil (Wi-Fi). Dans la version « WPA2-Enterprise », celle qui nous intéresse ici, il nous faut 802.1x pour assurer l'attachement du « supplicant » au point d'accès et RADIUS pour la partie authentification. RADIUS n'est pas obligé par la norme, mais c'est un standard de fait.

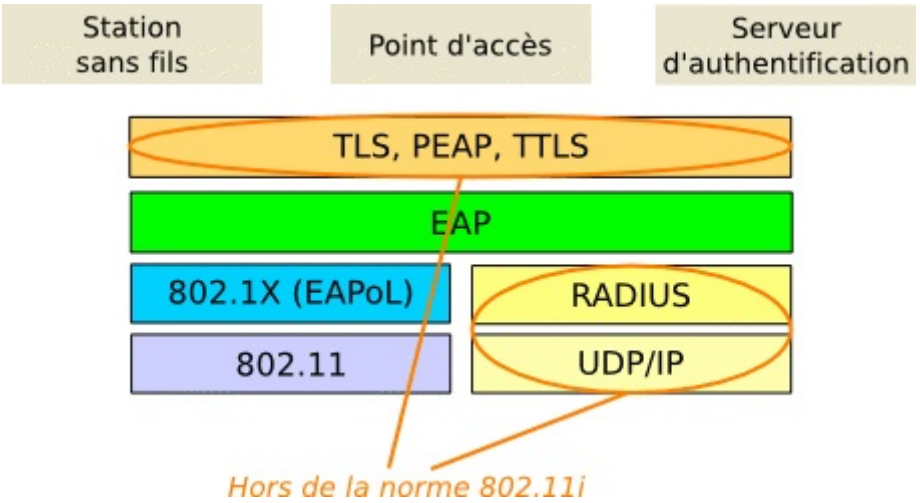
3-4-2 - Risques à éviter

Dans une connexion sans fil, il convient de prendre un soin particulier à être sûr que :

- l'on s'adresse au bon serveur d'authentification ;
- le point d'accès est bien celui que l'on croit ;
- le client est bien autorisé ;
- le client est bien celui qu'il prétend être ;
- il n'y a pas de « man in the middle » ;
- un point d'accès « voleur » ne peut prendre en cours de session la place du point d'accès « officiel » ;
- un espion ne pourra pas, dans un délai raisonnable, déchiffrer les informations qui circulent dans l'air.

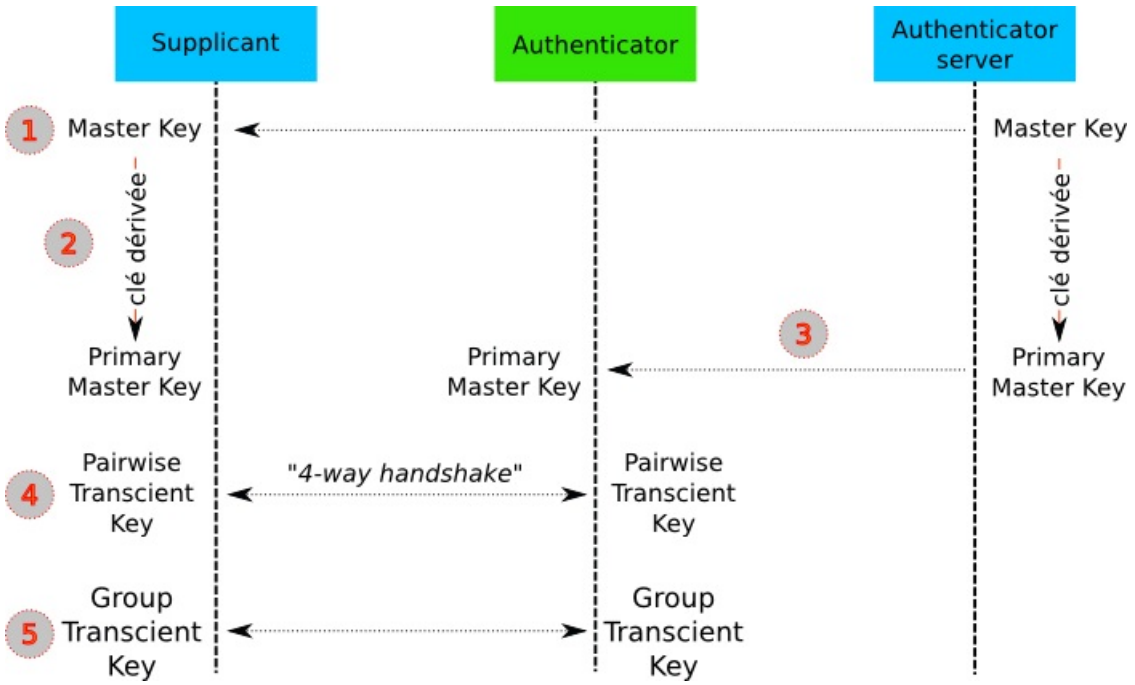
Il y a donc du travail.

3-4-3 - Principe de fonctionnement



Les protocoles d'authentification (TLS, PEAP, TTLS...) ne sont pas du domaine de 802.1X, RADIUS non plus.

Cependant, compte tenu de ce qui a été déjà vu, il est clair qu'il n'y a pas d'alternative.

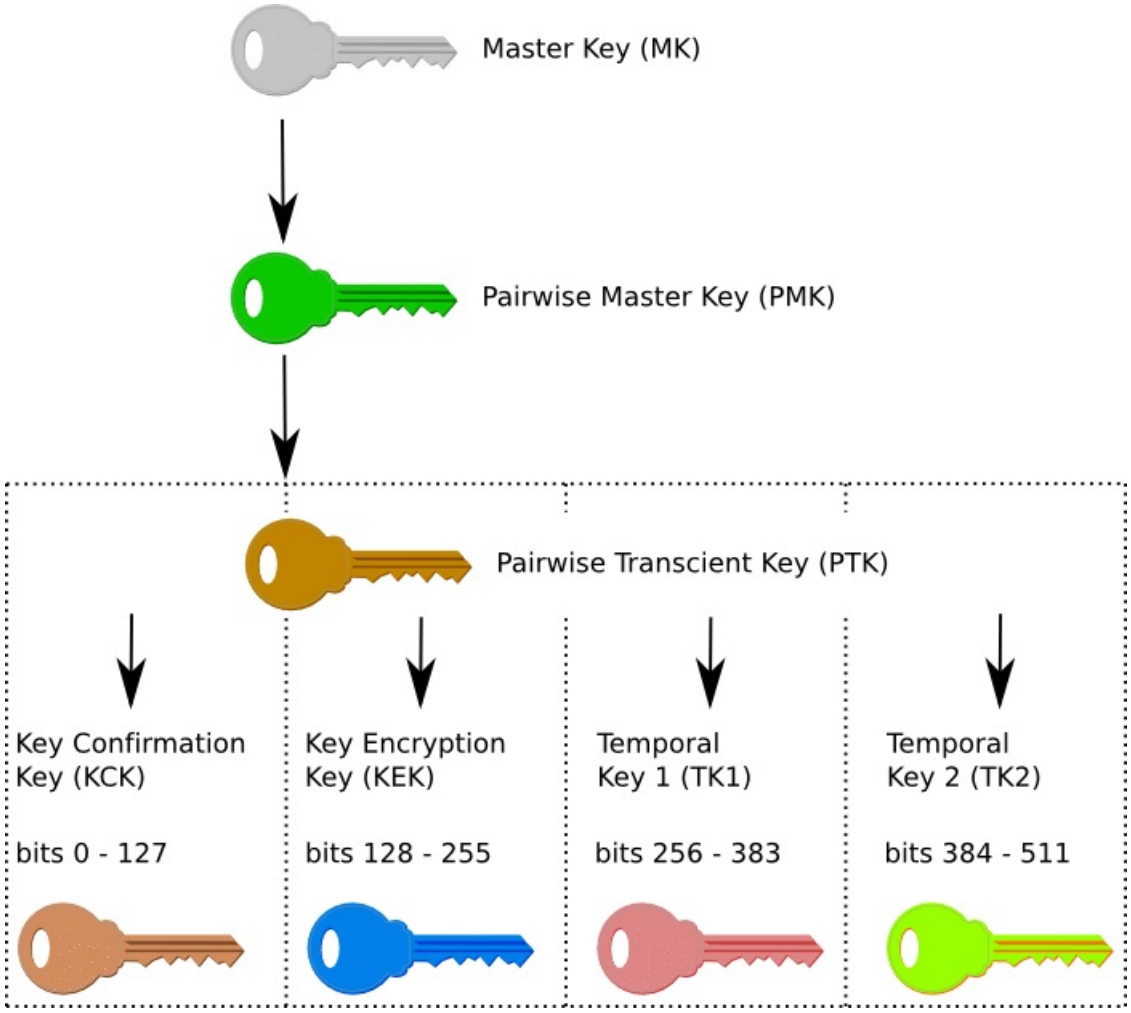


- Quand le client (supplicant) et le serveur d'authenticication (AS) s'authentifient, un des derniers messages envoyés par le serveur, contient la « Master Key » (MK). Cette clé n'est connue que par le client et le serveur. La MK n'est valable que pour cette session, entre ce client et ce serveur.
- Le client et le serveur calculent, à partir de la MK, une nouvelle clé, nommée « Pairwise Master Key » (PMK).
- La PMK est déplacée du serveur vers le point d'accès. Seuls le client et le serveur savent calculer cette PMK, sinon, le point d'accès pourrait prendre des décisions de contrôle d'accès à la place du serveur. La PMK est une nouvelle clé symétrique, valable uniquement pour cette session, entre ce client et ce point d'accès.
- La PMK et une poignée de main en quatre passes (« 4-way handshake ») sont utilisées entre le client et le point d'accès pour calculer et vérifier une « Pairwise Transcient Key » (PTK), qui sera utilisée uniquement dans cette session entre ce client et ce point d'accès. La PTK est en réalité un trousseau de trois clés :

1. la « Key Confirmation Key » (KCK) qui, comme son nom l'indique, est utilisée pour prouver la possession de la PMK et pour attacher la PMK au point d'accès ;
2. la « Key Encryption Key » (KEK) est utilisée pour distribuer la « Group Transcient Key » (GTK), décrite plus loin ;
3. les « Temporal Key 1 & 2 » (TK1/TK2) sont utilisées pour le chiffrement.

- la KEK et un autre « 4-way handshake » sont alors utilisés pour envoyer une « Group Transcient Key » (GTK) du point d'accès vers le client. La GTK est partagée par tous les clients connectés au même point d'accès. Elle est utilisée uniquement pour sécuriser le multicast et le broadcast.

En résumé :



- La MK représente la décision d'accès (positive).
- La PMK représente l'autorisation d'accès au medium 802.11.
- La PTK, contient :
  1. la KCK, utilisée pour attacher la PMK au point d'accès et pour prouver que le point d'accès dispose lui aussi de la PMK (authentification mutuelle entre le point d'accès et le client) ;
  2. la KEK, utilisée pour distribuer la GTK ;
  3. les TK, utilisées pour sécuriser le transfert de données.

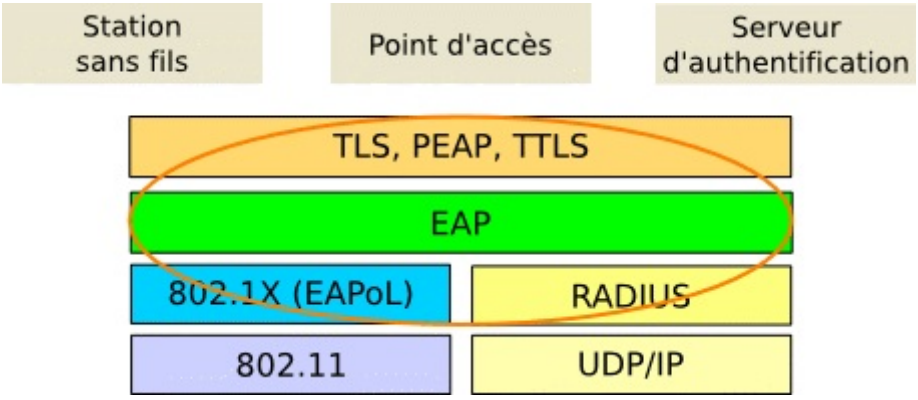
Finalement, nous avons bien :

- le client authentifié par le serveur RADIUS ;
- le serveur authentifié par le client,
- le client authentifié par le point d'accès ;

- le point d'accès authentifié par le client ;
- les données sont sécurisées. Chaque paquet (et même chaque fragment de paquet) est authentifié, si bien qu'il n'est normalement pas possible d'injecter frauduleusement des paquets forgés.

Le point d'accès et le serveur sont reliés par le réseau filaire. A priori, il y a moins de dangers. L'authentification mutuelle se fait par un « secret partagé », comme nous le verrons dans la mise en œuvre.

3-5 - Authentifications



3-5-1 - Autorité de certification

Il en existe de très officielles, leurs services coûtent généralement une fortune. Nous nous contenterons, au sein de notre système, de créer notre propre CA. Pratiquement, il faut créer un certificat de CA, qui permettra d'authentifier tous les autres certificats que nous devrons créer.

3-5-2 - EAP-TLS

TLS (Transport Layer Security) n'est autre que SSL v3 (Secure Socket Layer). Ici, nous aurons besoin d'un certificat x509 sur le serveur et sur le client. L'authentification se fait au moyen des clés privées (un message chiffré par une clé privée authentifie son auteur).

Avec cette méthode, il faut donc créer un certificat pour le serveur mais aussi un certificat par client. Le client n'a pas besoin de disposer d'un « login/password », le certificat suffit à l'authentifier.

3-5-3 - EAP-TTLS

Ici, seul le certificat du serveur suffit. Il permettra de construire un « tunnel chiffré » dans lequel voyagera le mot de passe du client. Une interception du trafic ne permettra que de récupérer un mot de passe chiffré.

3-5-4 - EAP-PEAP

Très similaire à TTLS, mais en plus le mot de passe ne circule pas dans le tunnel. Il s'agit ici d'un « challenge », réalisé au moyen du protocole MSCHAP v2 (d'origine Microsoft). PEAP est réputé encore plus sûr que TTLS.

Cette méthode peut être employée si l'on dispose d'un domaine Microsoft, avec ActiveDirectory, et que l'on souhaite que les utilisateurs s'authentifient avec leur « login/password » du domaine. Il « suffira » dans ce cas de configurer convenablement le serveur FreeRADIUS pour qu'il puisse consulter l'annuaire de ActiveDirectory, ce qui, en contrepartie, dispense de créer un certificat par client.

Il existe d'autres méthodes supportées par 802.1x, mais ce sont ces trois méthodes réputées les plus sécurisées.

3-6 - Les certificats

Nous allons créer tous les certificats nécessaires à l'utilisation de EAP-TLS. Il nous faudra :

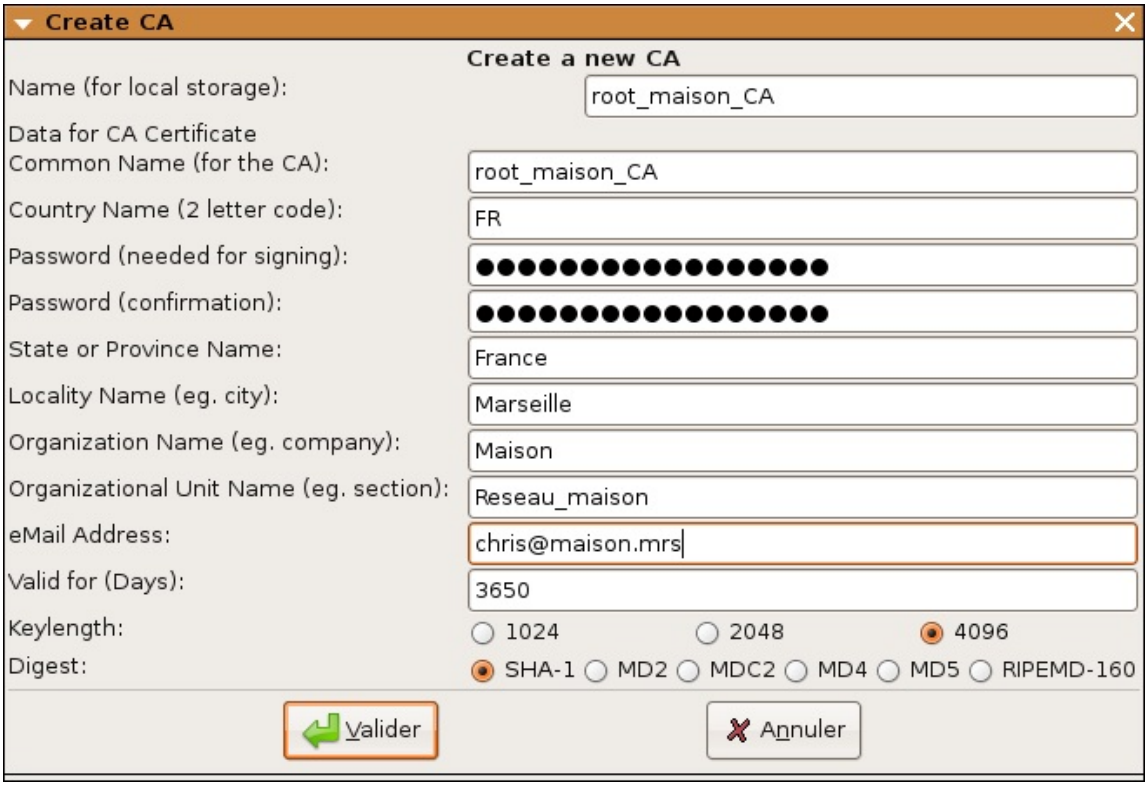
- un certificat qui représentera notre autorité de certification, dont la partie publique devra être disponible pour le serveur RADIUS et pour tous les clients. Ce certificat doit permettre d'authentifier tous les autres certificats que nous devrons créer ;
- un certificat pour le serveur RADIUS. La partie publique de ce certificat sera envoyée par le serveur à tous les supplicants lorsqu'ils voudront s'attacher au point d'accès ;
- un certificat par client, dont la partie publique sera envoyée au serveur RADIUS, en guise de code d'accès.

Pour réaliser tout ceci, nous faisons appel à OpenSSL. Mais la ligne de commande n'est pas bien ergonomique. Fort heureusement, il existe un outil qui va nous aider grandement : TinyCA.

TinyCA est une application (Perl/GTK) qui permet de manipuler plus facilement OpenSSL. Nous pourrons réaliser et gérer avec les divers certificats qui nous seront nécessaires. Il n'est ni utile ni même souhaitable d'effectuer cette opération sur le serveur FreeRADIUS. Au contraire, il est même vivement conseillé de réaliser les opérations qui suivent sur une machine inaccessible depuis le réseau.

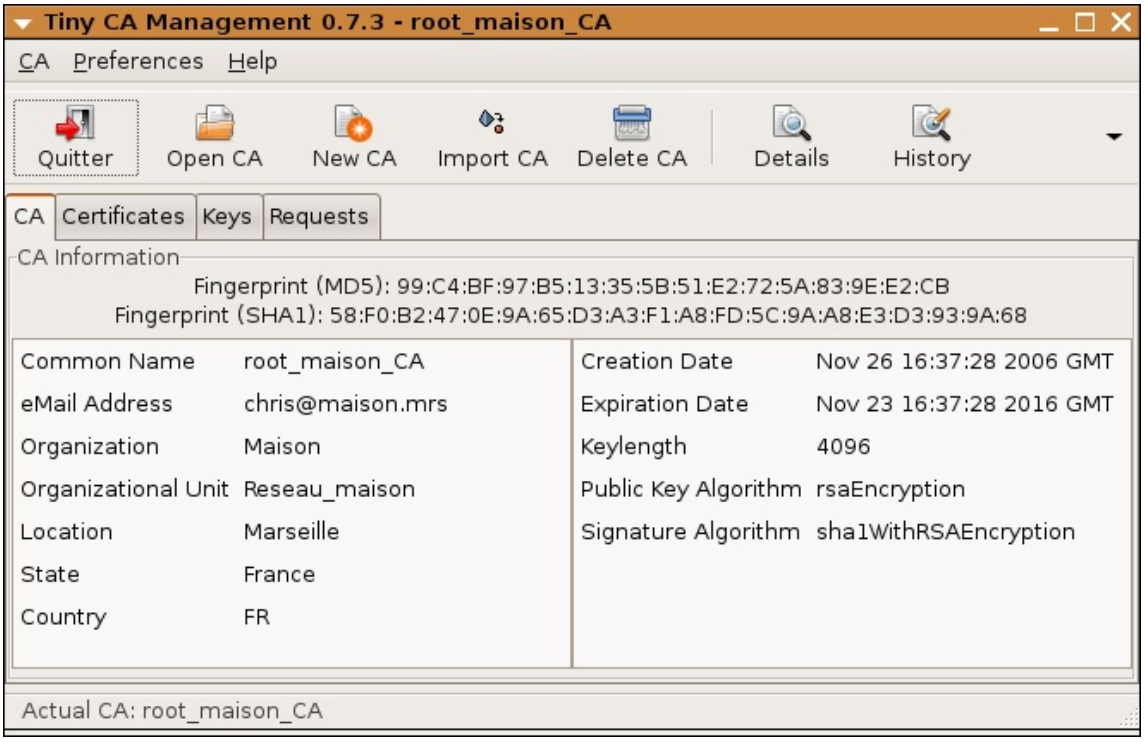
3-6-1 - Création d'une CA

C'est bien sûr la première opération à réaliser. D'ailleurs la fenêtre qui suit est la première que vous verrez au premier lancement de tinyca :



Puis vient la configuration. Pour les tests, nous garderons les valeurs par défaut :

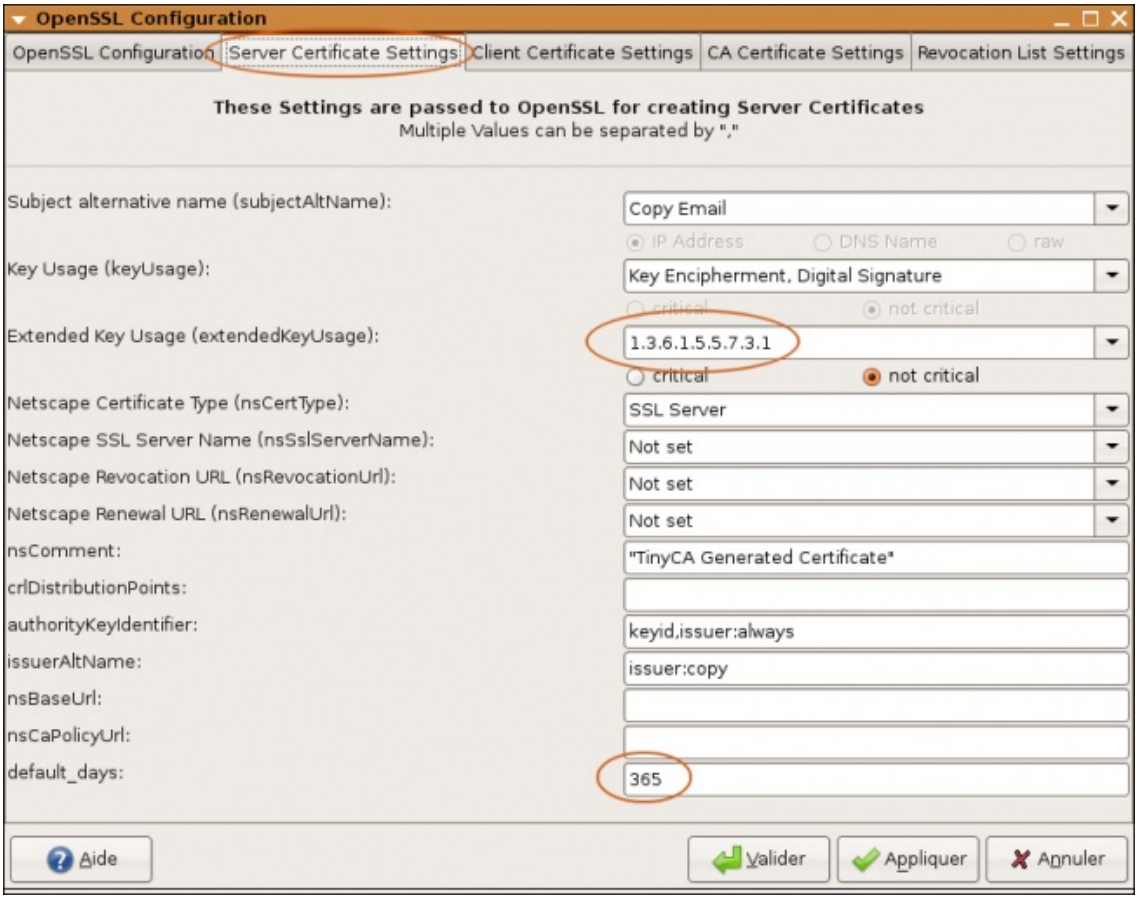
Le temps que les clés se construisent, et nous obtenons :



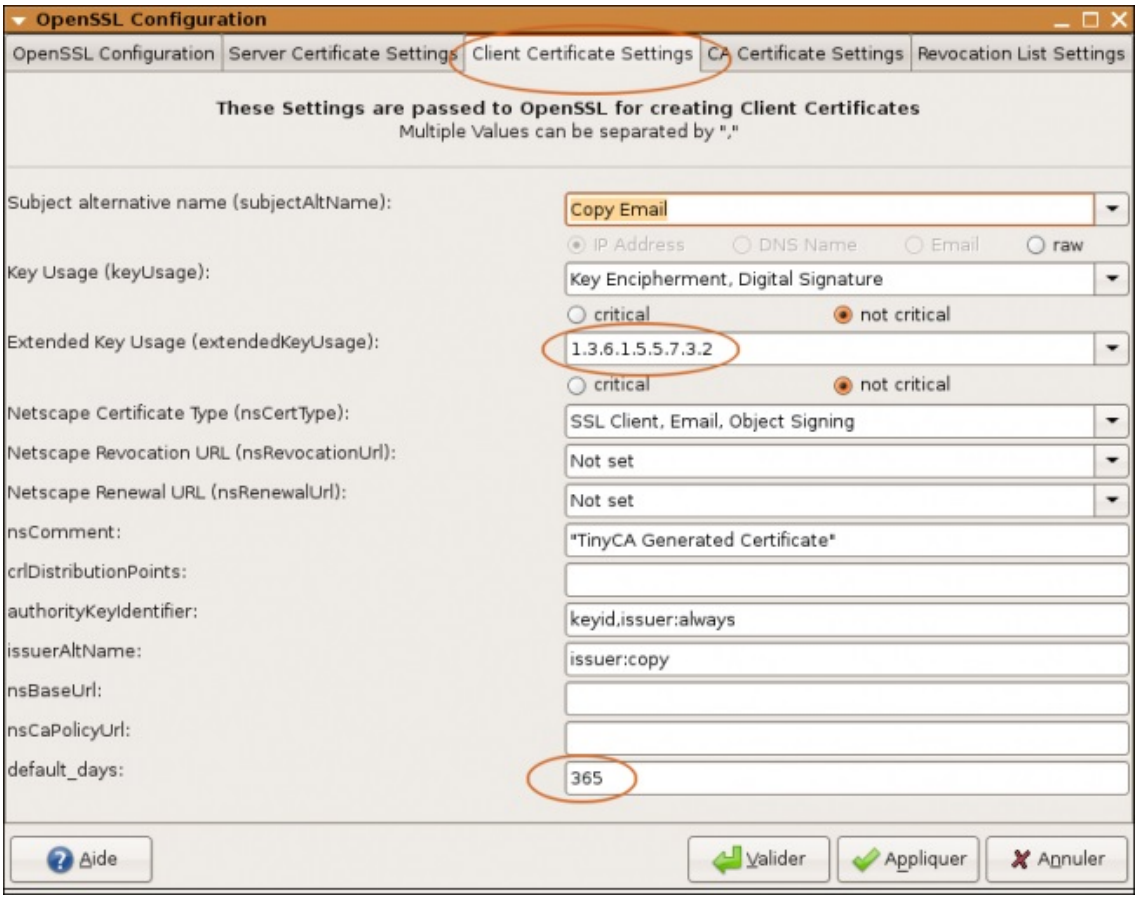
3-6-2 - Configuration des préférences

Le menu « préférences » va nous permettre de donner quelques directives fondamentales pour ce que nous avons à faire.

Pour les certificats du serveur, nous devons ajouter l'extension 1.3.6.1.5.5.7.3.1, ce qui, en français, signifie que le certificat est destiné à un serveur d'authentification. De même, choisissez la durée de validité du certificat (ici, un an). À l'échéance, il faudra penser faire un nouveau certificat.

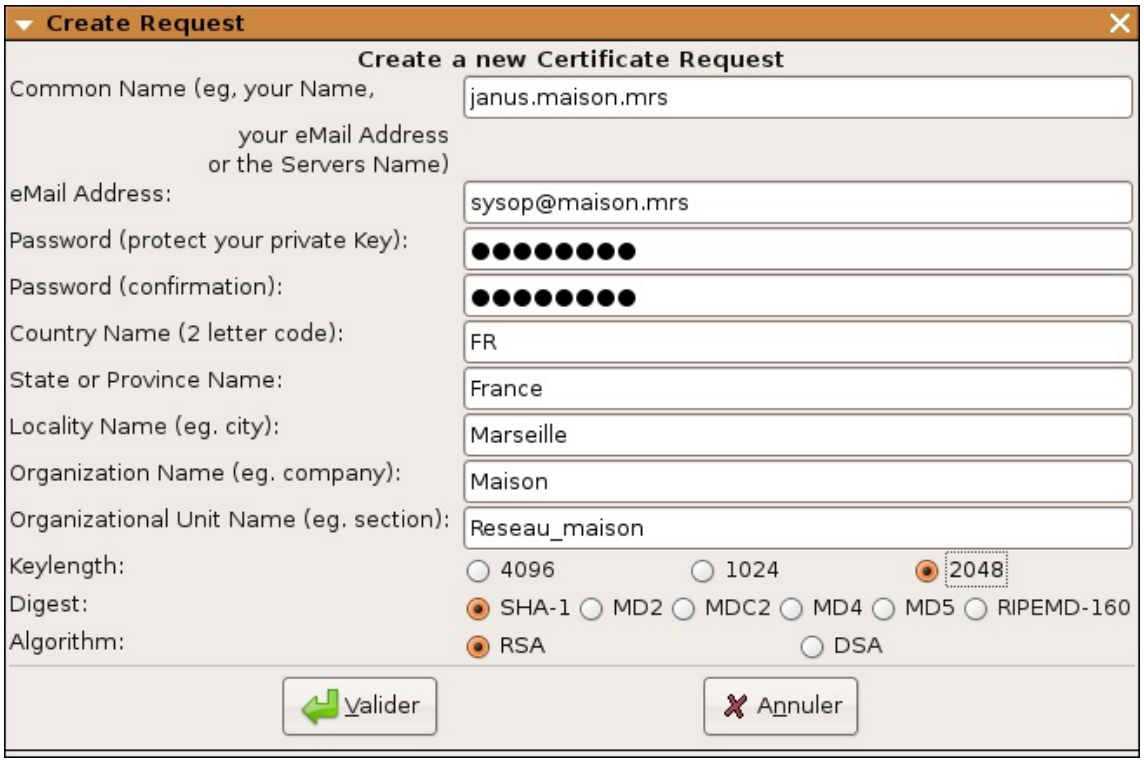


Pour les certificats des clients, nous devons ajouter l'extension 1.3.6.1.5.5.7.3.2, qui signifie que les certificats sont destinés à authentifier les clients. Pensez aussi à leur durée de validité.



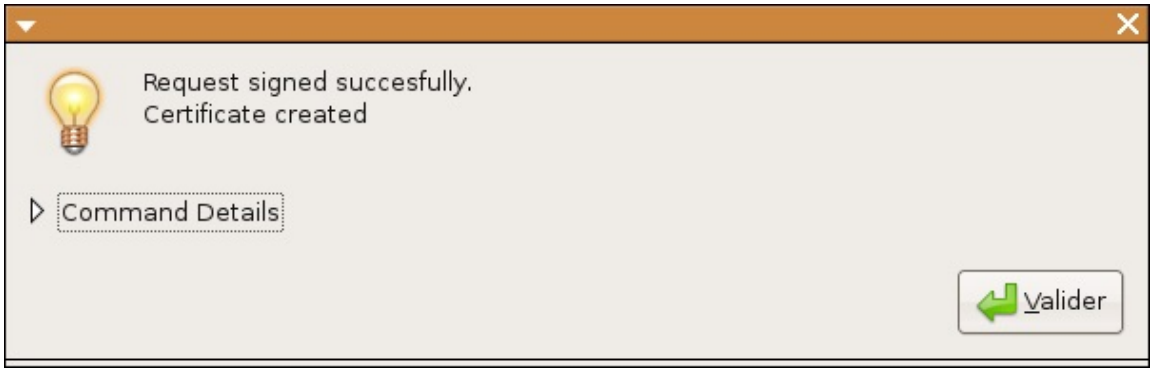
3-6-3 - Création du certificat du serveur

La création des divers certificats (serveur et clients) suit la même procédure, en utilisant le paramétrage par défaut effectué plus haut. Il suffit de compléter les champs vides :

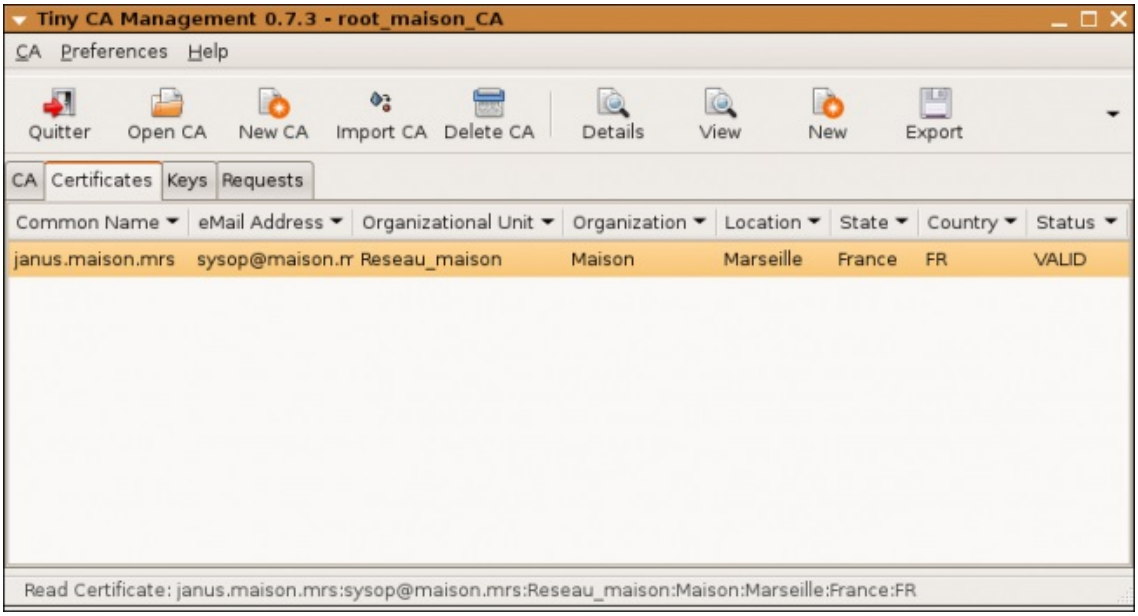


En réalité, nous n'avons créé qu'une requête de certificat. Cette requête doit maintenant être validée par la CA, ce qui se traduit par l'apposition de la signature de la CA. Il s'agit du mot de passe choisi lors de la création de la CA :

Notez que lors de cette opération, il est possible de modifier la durée de validité. C'est important surtout pour les certificats clients. Il est en effet possible d'envisager la création de certificats à durée très courte, dans le cas de clients de passage, par exemple, ou pour des autorisations ponctuelles.

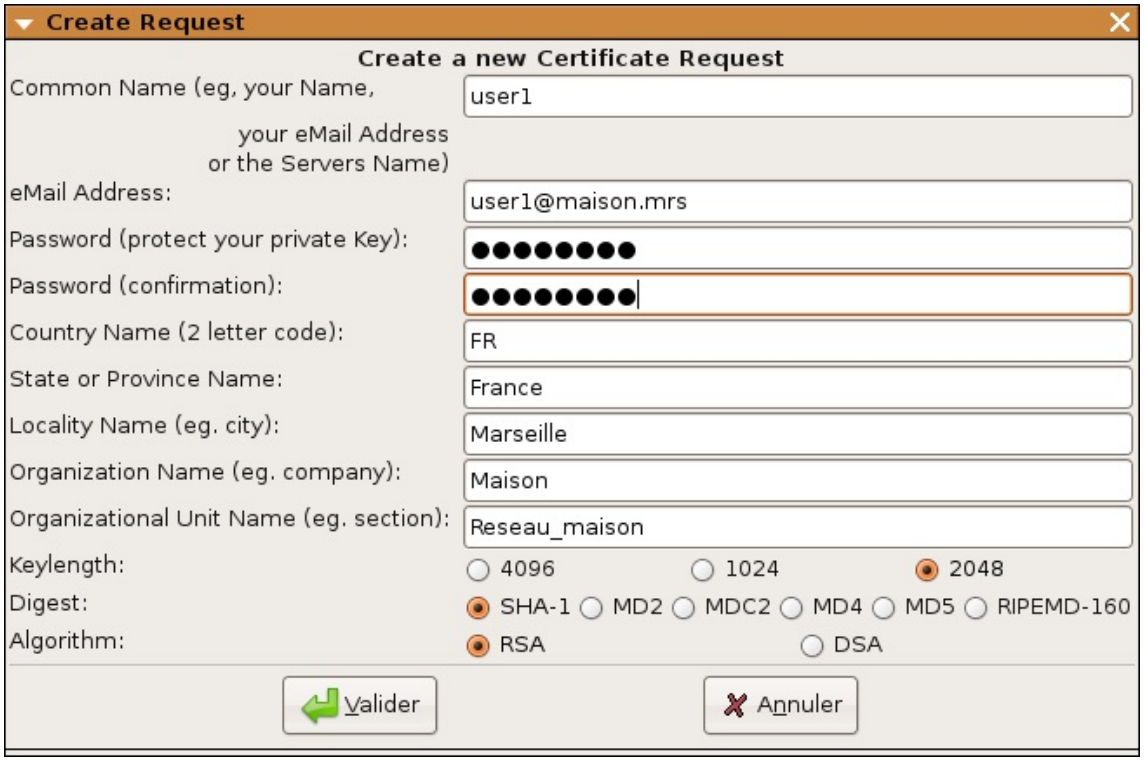


À l'issue de la signature, assurez-vous que le compte-rendu est positif. Nous devons retrouver le certificat inscrit dans la liste :

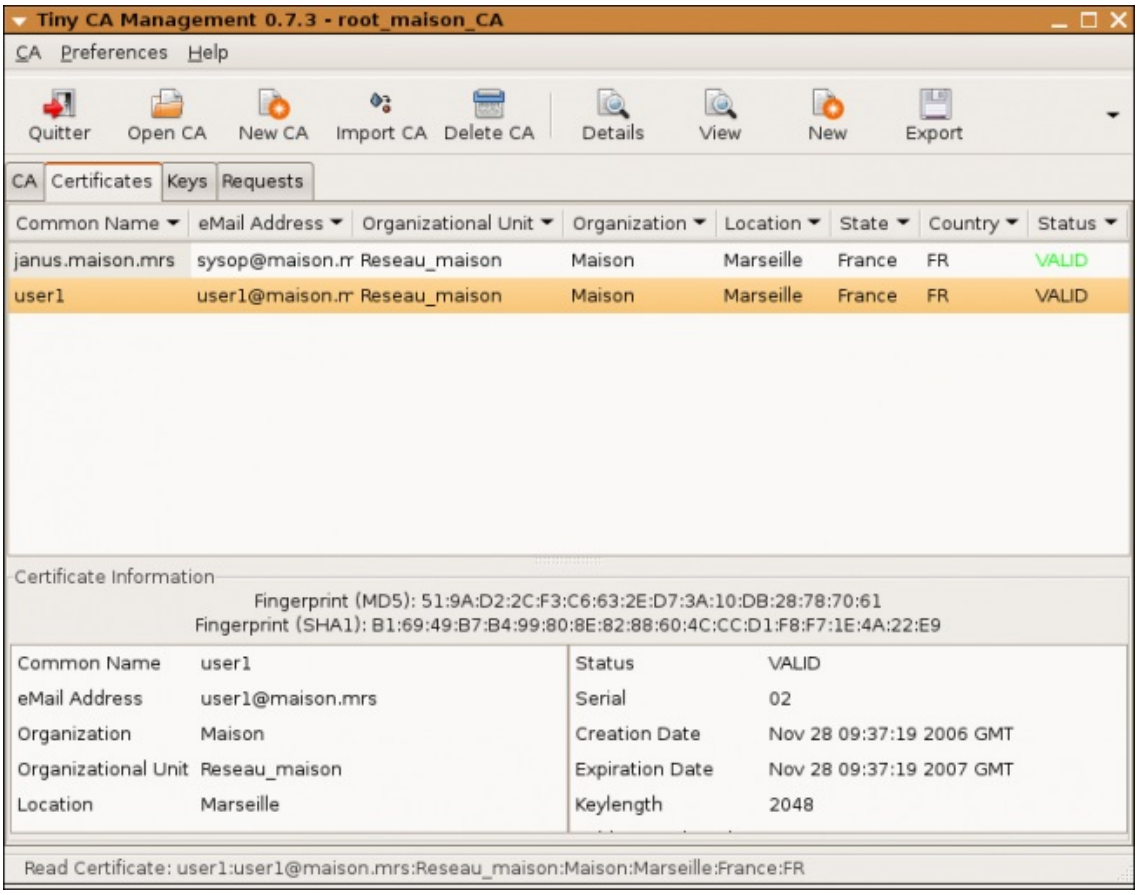


3-6-4 - Création d'un certificat client :

Procédure identique :



Je passe les détails de signature. Finalement, nous avons :



3-7 - Exportation des certificats

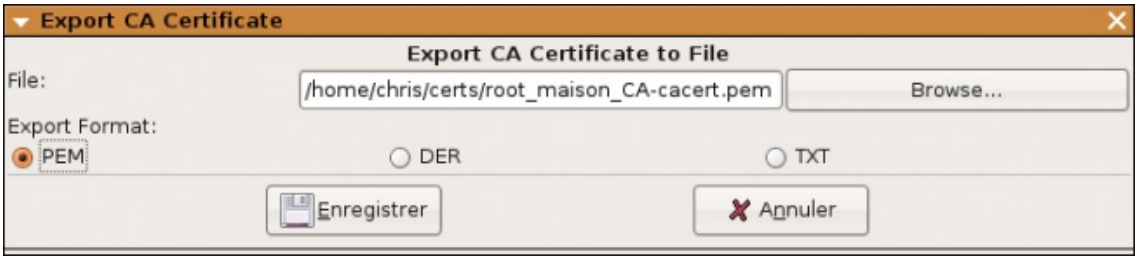
Il nous faut maintenant exporter les certificats à installer sur le serveur et sur le(s) client(s).

3-7-1 - Certificat de la racine de confiance (CA)

Ce certificat, qui représente la racine de confiance, devra être présent sur le serveur RADIUS et sur tous les clients. Nous devons donc l'exporter dans des formats compatibles avec OpenSSL (pem) et Windows (der).

Rappelons que c'est ce certificat qui permettra :

- au serveur de s'assurer que les certificats présentés par les clients sont bien authentiques ;
- aux clients de s'assurer que le certificat présenté par le serveur est bien authentique.

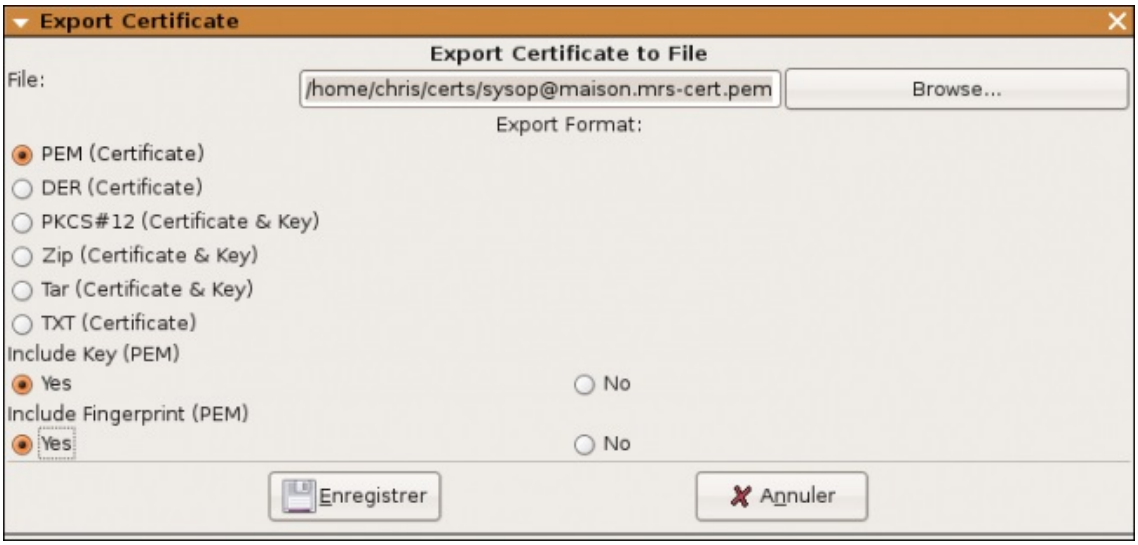


Vérifions que l'exportation est réussie :

Et recommençons l'opération, mais au format .der (je passe les détails).

3-7-2 - Certificat du serveur

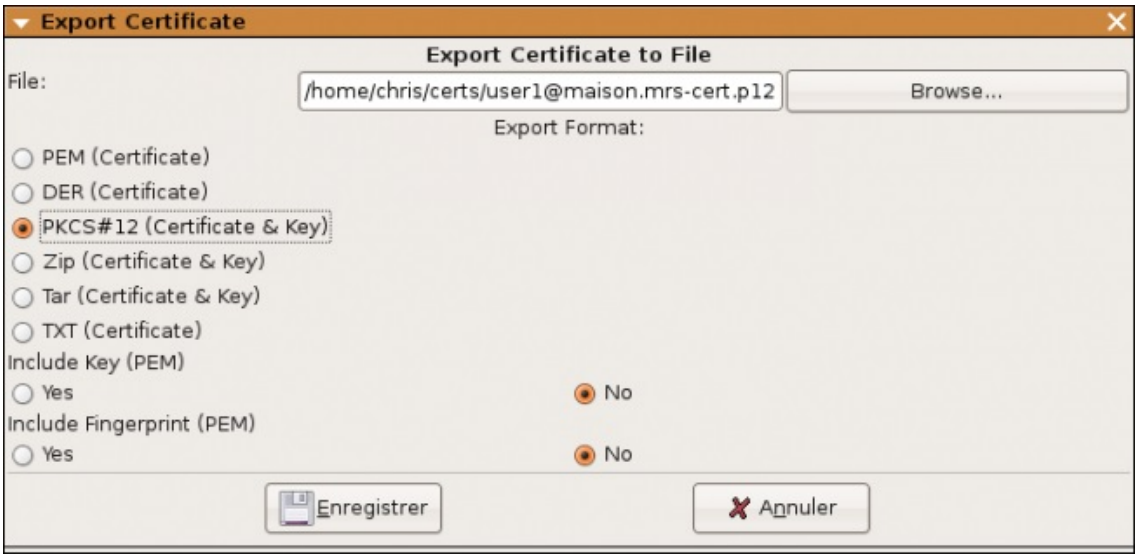
Ce certificat ne sera installé que sur le serveur FreeRADIUS, donc le format pem (avec la clé privée et l'empreinte intégrées) suffira :



3-7-3 - Certificat client

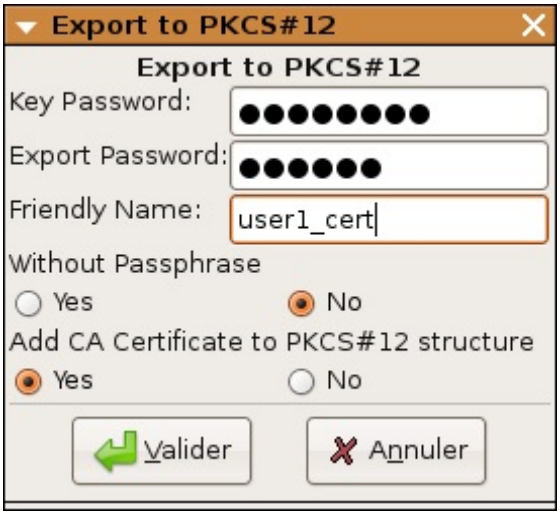
Ce certificat servira à « user1 ». Suivant qu'il utilisera Windows ou GNU/Linux, il lui faudra un .pem ou un .pkcs#12 (clé privée intégrée). C'est un utilisateur bicéphale, son portable est en « dual-boot ».

L'exportation au format .pem se fait de la même manière que pour le serveur, passons les détails. En revanche, pour le format pkcs#12, c'est un petit peu plus compliqué :



Deux mots de passe sont demandés :

- le premier (Key password) correspond à celui qui a été saisi lors de la création du certificat ;
- le second (Export Password) sera nécessaire lors de l'installation du certificat chez le client.



Finalement, nous avons dans notre répertoire d'exportation :

- root_maison_CA-cacert.der ;
- root_maison_CA-cacert.pem (les certificats de l'autorité) ;
- sysop@maison.mrs-cert.pem (le certificat du serveur) ;
- user1@maison.mrs-cert.pem ;
- user1@maison.mrs-cert.p12 (le certificat du client, dans les deux formats nécessaires).

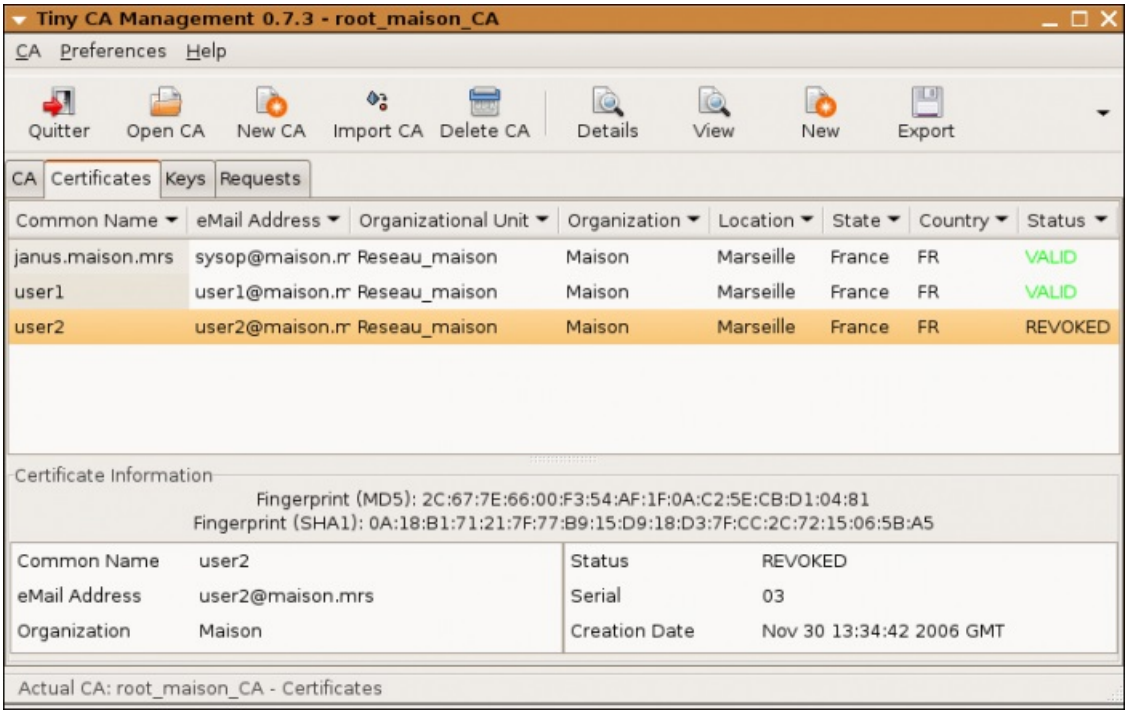
3-7-4 - Et la révocation

Il peut se faire, pour de multiples raisons, que l'on doive rendre inopérant un certificat avant sa date d'expiration :

- la clé privée est compromise ;
- le possesseur du certificat n'en a plus besoin plus tôt que prévu ;
- le possesseur du certificat a enfreint les règles et doit se voir banni du réseau ;
- et tant d'autres raisons.

Il faut donc disposer d'un moyen pour que le serveur RADIUS puisse rejeter des certificats non expirés, mais désormais interdits d'accès.

Dans un tel cas, il faut alors pouvoir créer un certificat de révocations qu'il sera nécessaire d'installer sur le serveur, et le configurer pour qu'il le consulte à chaque authentification. Tinyca sait réaliser de tels certificats. Ici, nous avons révoqué le certificat de « user2 » :



Il nous faut maintenant exporter un certificat des révocations :

Attention à la durée de validité d'un tel certificat. Bien entendu, ce certificat ne doit jamais expirer.

Nous retrouvons dans le répertoire d'exportation le certificat :

root_maison_CA-crl.pem.

4 - RADIUS

Remote Authentication Dial-In User Service est un système qui fait de « l'AAA » (Authentication, Authorization and Accounting). Utilisé depuis longtemps déjà par bon nombre de fournisseurs d'accès à l'internet pour authentifier leurs clients et leur communiquer une configuration IP.

RADIUS est également très utile pour sécuriser un réseau Wi-Fi, ou même un réseau filaire, dans certaines conditions.

Dans ce sous-chapitre, nous verrons comment installer et configurer un serveur RADIUS libre : FreeRADIUS, puis nous le mettrons en œuvre dans deux situations bien distinctes :

- la sécurisation d'un réseau Wi-Fi (authentification WPA2-TLS) ;
- la sécurisation d'un réseau filaire (authentification des stations par adresse MAC).

Ce type de solution s'avère fort intéressant si l'on dispose d'un réseau proposant un accès filaire et Wi-Fi, avec des utilisateurs susceptibles de venir y connecter leur ordinateur portable. Par définition, ce type de machine échappe totalement au contrôle de l'administrateur et peut être la source de bien des soucis.

4-1 - FreeRADIUS

(2.0.4 sur Debian Lenny).

4-1-1 - Avant de commencer...

RADIUS (Remote Authentication Dial-In User Service) est un vaste programme. Pour essayer de faire simple (donc schématique et incomplet), ce service est capable :

- d'authentifier un utilisateur distant, suivant de multiples modes plus ou moins sécurisés, en s'appuyant sur une base de connaissances allant du simple fichier texte à l'annuaire LDAP, en passant par une base de données de type SQL ;
- d'enregistrer des informations sur chaque « login » ;
- de renvoyer au demandeur des paramètres variés pouvant, suivant le cas, être une configuration IP, un numéro de LAN virtuel, etc.

Étudier dans le détail toutes les possibilités de RADIUS est hors de la portée de cet exposé. (c'est, de toute façon, également hors de ma propre portée). Nous nous contenterons ici de le mettre en œuvre dans les deux cas qui nous intéressent :

- authentification depuis leur adresse MAC des stations « connues » sur notre réseau filaire, en utilisant un système de type « login/password », avec le protocole CHAP (Challenge-Handshake Authentication Protocol), éventuellement en assignant un numéro de VLAN suivant la machine ;
- authentification avec un certificat x.509 sur le réseau Wi-Fi, en utilisant EAP-TLS.

Installer et surtout configurer un serveur radius pour la première fois a quelque chose d'assez rebutant, voire repoussant. Nous allons passer un peu de temps à détailler cette opération, ceci aidera probablement ceux qui n'ont encore jamais tenté l'aventure. Nous utilisons FreeRADIUS sur une Debian Lenny.

FreeRadius peut fonctionner en s'appuyant uniquement sur des fichiers texte. Ce n'est pas forcément ce qu'il y a de plus simple à gérer, si l'on doit manipuler un grand nombre de clients. Ici, nous utiliserons MySQL pour stocker les adresses MAC des clients. Outre la souplesse qu'apportent des outils comme phpmyadmin pour gérer la liste des clients, cette solution offre l'avantage de ne pas nécessiter de redémarrage de FreeRADIUS à chaque modification de la base.

4-1-2 - Installation de Freeradius

Pour des raisons de compatibilité de licences, FreeRadius est compilé par défaut sur Debian (Lenny) sans le support de TLS. TLS nous servira pour le WPA2. Nous allons donc reconstruire un paquet binaire à partir du paquet source, en tenant compte de cet usage.

Notons que ceci n'a plus lieu d'être sur Squeeze, ce qui simplifiera notre tâche, l'installation du binaire standard suffira à nos besoins.

4-1-2-1 - Préparatifs

Nous aurons besoin de quelques outils de compilation et de gestion des paquets source :

```
# aptitude install build-essential
...
# aptitude install apt-src
```

Puis nous devons mettre à jour la liste des paquets source :

```
# apt-src update
```

Enfin, nous installons le paquet source de FreeRadius dans un répertoire que nous aurons créé dans ce but. La commande apt-src install offre, entre autres, l'avantage d'installer automatiquement les dépendances.

```
# mkdir ~/build_freeradius
# cd ~/build_freeradius
# apt-src install freeradius
```

Nous devons retrouver dans notre répertoire :

```
# ls -l
total 2908
drwxr-xr-x 15 root root    4096 mar  4 15:14 freeradius-2.0.4+dfsg
-rw-r--r--  1 root root    4860 sep  7 20:02 freeradius_2.0.4+dfsg-6.diff.gz
-rw-r--r--  1 root root    1476 sep  7 20:02 freeradius_2.0.4+dfsg-6.dsc
-rw-r--r--  1 root root 2953674 mai 19  2008 freeradius_2.0.4+dfsg.orig.tar.gz
```

4-1-2-2 - Configuration de la compilation

Dans le répertoire ~/build_freeradius/freeradius-2.0.4+dfsg/debian nous avons un fichier nommé « rules », qui contient les directives de compilation. Nous allons devoir modifier ce fichier.

Voici la première partie qui nous intéresse :

```
        ./configure $(confflags) \
                --prefix=/usr \
                --exec-prefix=/usr \
                --mandir=$(mandir) \
                --sysconfdir=/etc \
                --libdir=$(libdir) \
                --datadir=/usr/share \
```

```
                --localstatedir=/var \
                --with-raddbdir=$(raddbdir) \
                --with-logdir=/var/log/$(package) \
                --enable-ltdl-install=no --enable-strict-dependencies \
                --with-large-files --with-udpfromto --with-edir \
                --enable-developer \
                --config-cache \
                --without-rlm_eap_tls \
                --without-rlm_eap_ttls \
                --without-rlm_eap_peap \
                --without-rlm_eap_tnc \
                --without-rlm_otp \
                --with-rlm_sql_postgresql_lib_dir=`pg_config --libdir` \
                --with-rlm_sql_postgresql_include_dir=`pg_config --includedir` \
                --without-openssl \
                --without-rlm_eap_ikev2 \
                --without-rlm_sql_oracle \
                --without-rlm_sql_unixodbc \
                --with-system-libtool
```

Les lignes surlignées sont celles qu'il faut supprimer pour obtenir le support de TLS. Nous devons donc aboutir à ceci :

```
    ./configure $(confflags) \
        --prefix=/usr \
        --exec-prefix=/usr \
        --mandir=$(mandir) \
        --sysconfdir=/etc \
        --libdir=$(libdir) \
        --datadir=/usr/share \
        --localstatedir=/var \
        --with-raddbdir=$(raddbdir) \
        --with-logdir=/var/log/$(package) \
        --enable-ltdl-install=no --enable-strict-dependencies \
        --with-large-files --with-udpfromto --with-edir \
        --enable-developer \
        --config-cache \
        --without-rlm_eap_tnc \
        --with-rlm_sql_postgresql_lib_dir=`pg_config --libdir` \
        --with-rlm_sql_postgresql_include_dir=`pg_config --includedir` \
        --without-rlm_eap_ikev2 \
        --without-rlm_sql_oracle \
        --without-rlm_sql_unixodbc \
        --with-system-libtool
```

Ce serait tout si le concepteur du paquet n'avait pas mis un petit test pour bloquer la compilation s'il traine un lien vers OpenSSL lors de l'édition des liens. Il faut repérer ce test et l'inhiber. Le voici :

```
    for pkg in ${pkgs} ; do \
      if dh_shlibdeps -p $$pkg -- -O 2>/dev/null | grep -q libssl; then \
        echo "$$pkg links to openssl" ;\
        exit 1 ;\
      fi ;\
    done
```

Il suffit de commenter la ligne exit 1;\ comme ceci :

```
    for pkg in ${pkgs} ; do \
      if dh_shlibdeps -p $$pkg -- -O 2>/dev/null | grep -q libssl; then \
        echo "$$pkg links to openssl" ;\
#        exit 1 ;\
      fi ;\
    done
```

Du côté de la compilation, nous sommes parés.

Voici donc le fichier « rules » tel qu'il doit finalement se présenter :

```
#!/usr/bin/make -f
# -*- makefile -*-
# Sample debian/rules that uses debhelper.
#
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37 of dh-make.
#
# Modified to make a template file for a multi-binary package with separated
# build-arch and build-indep targets  by Bill Allombert 2001

# Uncomment this to turn on verbose mode.
export DH_VERBOSE=1

.NOTPARALLEL:

SHELL           =/bin/bash

package         = freeradius
freeradius_dir  = $(CURDIR)/debian/tmp/

mandir          = /usr/share/man
libdir          = /usr/lib/$(package)
logdir          = /var/log/$(package)
pkgdocdir       = /usr/share/doc/$(package)
raddbdir        = /etc/$(package)

modulelist=krb5 ldap sql_mysql sql_iodbc sql_postgresql
pkgs=$(shell dh_listpackages)

# This has to be exported to make some magic below work.
export DH_OPTIONS

# These are used for cross-compiling and for saving the configure cript
# from having to guess our platform (since we know it already)
export DEB_HOST_GNU_TYPE  ?= $(shell dpkg-architecture -qDEB_HOST_GNU_TYPE)
export DEB_BUILD_GNU_TYPE ?= $(shell dpkg-architecture -qDEB_BUILD_GNU_TYPE)

ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
        CFLAGS += -O0
else
        CFLAGS += -O2
endif

ifeq ($(DEB_BUILD_GNU_TYPE), $(DEB_HOST_GNU_TYPE))
        confflags += --build $(DEB_HOST_GNU_TYPE)
else
        confflags += --build $(DEB_BUILD_GNU_TYPE) --host $(DEB_HOST_GNU_TYPE)
endif

config.status: configure
        dh_testdir

ifeq (config.sub.dist,$(wildcard config.sub.dist))
        rm config.sub
else
        mv config.sub config.sub.dist
endif
ifeq (config.guess.dist,$(wildcard config.guess.dist))
        rm config.guess
else
        mv config.guess config.guess.dist
endif
        ln -s /usr/share/misc/config.sub config.sub
        ln -s /usr/share/misc/config.guess config.guess


        ./configure $(confflags) \
                --prefix=/usr \
                --exec-prefix=/usr \
                --mandir=$(mandir) \
```

```
                --sysconfdir=/etc \
                --libdir=$(libdir) \
                --datadir=/usr/share \
                --localstatedir=/var \
                --with-raddbdir=$(raddbdir) \
                --with-logdir=/var/log/$(package) \
                --enable-ltdl-install=no --enable-strict-dependencies \
                --with-large-files --with-udpfromto --with-edir \
                --enable-developer \
                --config-cache \
                --without-rlm_eap_tnc \
                --with-rlm_sql_postgresql_lib_dir=`pg_config --libdir` \
                --with-rlm_sql_postgresql_include_dir=`pg_config --includedir` \
                --without-rlm_eap_ikev2 \
                --without-rlm_sql_oracle \
                --without-rlm_sql_unixodbc \
                --with-system-libtool

#Architecture
build: build-arch build-indep

build-arch: build-arch-stamp
build-arch-stamp: config.status
        $(MAKE)
        touch $@

build-indep: build-indep-stamp
build-indep-stamp: config.status
        touch $@

clean:
        dh_testdir
        dh_testroot
        rm -f build-arch-stamp build-indep-stamp
        rm -f config.cache config.log

        [ ! -d src/modules/lib ] || rm -fr src/modules/lib || true
        [ ! -d src/binary ] || rm -fr src/binary || true

        # Add here commands to clean up after the build process.
ifeq (Make.inc,$(wildcard Make.inc))
        $(MAKE) distclean
endif
ifeq (config.sub.dist,$(wildcard config.sub.dist))
        rm -f config.sub
        mv config.sub.dist config.sub
endif
ifeq (config.guess.dist,$(wildcard config.guess.dist))
        rm -f config.guess
        mv config.guess.dist config.guess
endif
        dh_clean

install: install-indep install-arch
install-indep: build-indep-stamp
        dh_testdir
        dh_testroot
        dh_installdirs -i

        $(MAKE) -C dialup_admin DIALUP_PREFIX=/usr/share/freeradius-dialupadmin \
                                DIALUP_DOCDIR=/usr/share/doc/freeradius-dialupadmin \
                                DIALUP_CONFDIR=/etc/freeradius-dialupadmin \
                                R=$(freeradius_dir) install

        mv $(freeradius_dir)/usr/share/freeradius-dialupadmin/bin/dialup_admin.cron \
               $(freeradius_dir)/usr/share/freeradius-dialupadmin/bin/freeradius-dialupadmin.cron
        mv $(freeradius_dir)/usr/share/doc/freeradius-dialupadmin/Changelog \
               $(freeradius_dir)/usr/share/doc/freeradius-dialupadmin/changelog

        install -m0644 debian/apache2.conf $(freeradius_dir)/etc/freeradius-dialupadmin/

        dh_install -i --sourcedir=$(freeradius_dir)
```

```
        dh_installdocs -p freeradius-dialupadmin dialup_admin/README

install-arch: build-arch-stamp
        dh_testdir
        dh_testroot
        dh_installdirs -s
        test -d $(freeradius_dir)/usr/lib/freeradius || mkdir -p $(freeradius_dir)/usr/lib/freeradius
        ln -s rlm_sql.so $(freeradius_dir)/usr/lib/freeradius/librlm_sql.so
        $(MAKE) install R=$(freeradius_dir)

        # rename radius binary to play nicely with others
        mv $(freeradius_dir)/usr/sbin/radiusd $(freeradius_dir)/usr/sbin/$(package)
        mv $(freeradius_dir)/$(mandir)/man8/radiusd.8 $(freeradius_dir)/$(mandir)/man8/$(package).8

        dh_install --sourcedir=$(freeradius_dir) -p libfreeradius2
        dh_install --sourcedir=$(freeradius_dir) -p libfreeradius-dev

        for mod in ${modulelist}; do \
          pkg=$${mod##sql_} ; \
          dh_install --sourcedir=$(freeradius_dir) -p freeradius-$$pkg ; \
          rm -f $(freeradius_dir)/usr/lib/freeradius/rlm_$$mod*.so ; \
        done

        dh_install --sourcedir=$(freeradius_dir) -p freeradius-utils
        dh_install --sourcedir=$(freeradius_dir) -p freeradius

        dh_strip -a --dbg-package=freeradius-dbg

        dh_makeshlibs -a -n
        for pkg in ${pkgs} ; do \
          if dh_shlibdeps -p $$pkg -- -O 2>/dev/null | grep -q libssl; then \
            echo "$$pkg links to openssl" ;\
#           exit 1 ;\
          fi ;\
        done
        dh_shlibdeps

binary-common:
        dh_testdir
        dh_testroot
        dh_installchangelogs
        dh_installdocs
        dh_installexamples
        dh_installlogrotate
        dh_installpam --name=radiusd
        dh_installinit --noscripts
        dh_installman
        dh_lintian
        dh_link
        dh_compress -Xexamples
        dh_fixperms
        dh_installdeb
        dh_gencontrol
        dh_md5sums
        dh_builddeb

# Build architecture independant packages using the common target.
binary-indep: build-indep install-indep
        $(MAKE) -f debian/rules DH_OPTIONS=-i binary-common

# Build architecture dependant packages using the common target.
binary-arch: build-arch install-arch
        $(MAKE) -f debian/rules DH_OPTIONS=-s binary-common

binary: binary-arch binary-indep
.PHONY: build clean binary-indep binary-arch binary install install-indep install-arch
```

4-1-2-3 - Le fichier « control »

Nous devons ici ajouter la dépendance à la bibliothèque libssl-dev :

```
Source: freeradius
Build-Depends: autotools-dev, debhelper (>= 6.0.7), libgdbm-dev, libiodbc2-dev, libkrb5-dev, libldap2-
dev, libltdl3-dev, libmysqlclient15-dev | libmysqlclient-dev, libpam0g-dev, libpcap-dev, libperl-dev,
 libpq-dev, libsasl2-dev, libsnmp-dev, libtool, python-dev, libssl-dev
Section: net
Priority: optional
Maintainer: Stephen Gran
Uploaders: Mark Hymers
Standards-Version: 3.7.3
...
```

Et nous assurer qu'elle est bien présente sur notre système :

```
aptitude install libssl-dev
```

4-1-2-4 - construction des binaires

Il nous reste à construire les paquets binaires :

```
cd ~/build_freeradius
apt-src build freeradius
```

Une fois la compilation terminée, normalement sans message d'erreur, nous obtenons la liste des paquets qui suit. Nous n'avons pas besoin de les installer tous ici, puisque nous n'utiliserons que MySQL :

```
ls -l *.deb
-rw-r--r-- 1 root root 513228 f&#233;v 22 16:20 freeradius_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root 205030 f&#233;v 22 16:21 freeradius-common_2.0.4+dfsg-6_all.deb
-rw-r--r-- 1 root root 949458 f&#233;v 22 16:20 freeradius-dbg_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root 132748 f&#233;v 22 16:21 freeradius-dialupadmin_2.0.4+dfsg-6_all.deb
-rw-r--r-- 1 root root  17184 f&#233;v 22 16:20 freeradius-iodbc_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root  18082 f&#233;v 22 16:20 freeradius-krb5_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root  34426 f&#233;v 22 16:20 freeradius-ldap_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root  24874 f&#233;v 22 16:20 freeradius-mysql_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root  35364 f&#233;v 22 16:20 freeradius-postgresql_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root  71282 f&#233;v 22 16:20 freeradius-utils_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root  85212 f&#233;v 22 16:20 libfreeradius2_2.0.4+dfsg-6_i386.deb
-rw-r--r-- 1 root root 103672 f&#233;v 22 16:20 libfreeradius-dev_2.0.4+dfsg-6_i386.deb
```

4-1-2-5 - Installation des paquets utiles

La commande dpkg :

```
dpkg -i libfreeradius2_2.0.4+dfsg-6_i386.deb freeradius-common_2.0.4+dfsg-6_all.deb
 freeradius_2.0.4+dfsg-6_i386.deb freeradius-mysql_2.0.4+dfsg-6_i386.deb freeradius-
utils_2.0.4+dfsg-6_i386.deb
```

4-1-2-6 - Se protéger des mises à jour de « aptitude »

Si nous compilons maintenant le paquet binaire, nous obtiendrons des paquets ayant le même nom (version comprise), que les binaires de la distribution, et les mises à jour futures ne manqueront pas de nous remplacer notre construction à la première occasion.

Une solution consiste à utiliser l'outil dpkg pour gérer la sélection des paquets installés. Faisons d'abord un :

```
:~# dpkg --get-selections > packages
```

De manière à obtenir le fichier packages qu'il nous faudra éditer. Si nous recherchons les paquets relatifs à Freeradius qui sont installés (au cas où nous aurions des trous dans la mémoire) :

```
aptitude search freeradius | grep ^i
i   freeradius                      - a high-performance and highly configurable
i   freeradius-common               - FreeRadius common files
i   freeradius-mysql                - MySQL module for FreeRADIUS server
i   freeradius-utils                - FreeRadius client utilities
i   libfreeradius2                  - FreeRADIUS shared library
```

Nous retrouvons dans le fichier packages que nous avons créé, les mêmes paquets :

```
cat packages | grep radius
freeradius                      install
freeradius-common                  install
freeradius-mysql                install
freeradius-utils                install
libfreeradius2                    install
```

Nous devons éditer ce fichier en remplaçant l'information install par hold, qui indiquera à apt que ces paquets ne doivent pas être touchés lors des mises à jour. Après édition, nous devons avoir :

```
cat packages | grep radius
freeradius                      hold
freeradius-common                  hold
freeradius-mysql                hold
freeradius-utils                hold
libfreeradius2                    hold
```

Il nous reste à entrer ces nouvelles informations dans la base de données des paquets installés :

```
:~# dpkg --set-selections < packages
```

Si tout s'est bien passé, la vérification suivante doit donner :

```
aptitude search freeradius | grep ^i
ih  freeradius                      - a high-performance and highly configurable
ih  freeradius-common               - FreeRadius common files
ih  freeradius-mysql                - MySQL module for FreeRADIUS server
ih  freeradius-utils                - FreeRadius client utilities
ih  libfreeradius2                  - FreeRADIUS shared library
```

Le h qui suit le i indique bien que ces paquets sont désormais protégés des mises à jour.

4-1-3 - Configuration

4-1-3-1 - Création de la base MySQL

Nous voulons ici faire quelque chose de « simple ». Il sera toujours temps de compliquer les choses une fois que la solution minimale aura été validée. Notre base MySQL contiendra la liste des « utilisateurs » (chez nous des adresses MAC), la liste des « authenticators » (nos switches et notre borne WI-FI) et éventuellement une liste d'utilisateurs autorisés à utiliser le WI-FI, nous verrons plus loin pourquoi.

FreeRADIUS n'aura donc à manipuler cette base qu'en lecture, il n'aura rien à écrire dedans. Nous allons donc créer une base radius et un utilisateur radius qui ne pourra que faire des SELECT dans cette base. De cette manière, si notre serveur FreeRADIUS venait à être compromis, il ne pourrait pas facilement transformer la base en foutoir.

Nous utilisons ici :

```
# mysql -V
mysql  Ver 14.12 Distrib 5.0.51a, for debian-linux-gnu (i486) using readline 5.2
Cr&#233;ation de la base :
```

```
# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 42
Server version: 5.0.51a-24 (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE radius;
Query OK, 1 row affected (0.01 sec)
```

Création de l'utilisateur :

```
CREATE USER 'radius'@'localhost' IDENTIFIED BY 'epikoi';
GRANT SELECT ON radius . * TO 'radius'@'localhost';
```

Pour la création des tables, FreeRADIUS propose des fichiers SQL qui vont nous aider ici. Ils se trouvent dans /etc/freeradius/sql/mysql/. Le principal fichier s'appelle schema.sql et crée les tables les plus importantes :

- radacct (inutilisée dans notre cas) ;
- radcheck ;
- radgroupcheck ;
- radgroupreply (inutilisée dans notre cas) ;
- radreply (inutilisée dans notre cas) ;
- radusergroup ;
- radpostauth (inutilisée dans notre cas).

Le second s'appelle nas.sql et n'ajoute qu'une seule table nas. Cette table est destinée à contenir les authenticators (nas). Nous pourrions nous en passer et utiliser un simple fichier texte à la place, d'autant que cette table n'est lue qu'une seule fois au démarrage de FreeRADIUS (contrairement à toutes les autres) et n'apporte donc pas grand-chose de plus qu'un fichier texte.

Finalement, pour vérification :

```
# mysql -uroot -pepikoi radius
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 83
Server version: 5.0.51a-24-log (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show tables;
+------------------+
| Tables_in_radius |
+------------------+
| nas              |
| radacct          |
| radcheck         |
| radgroupcheck    |
| radgroupreply    |
| radpostauth      |
| radreply         |
| radusergroup     |
+------------------+
8 rows in set (0.00 sec)
```

4-1-3-2 - Configuration de FreeRADIUS

C'est maintenant que nous allons vraiment commencer à nous amuser. En effet il y a pas mal de fichiers qui ont été placés dans /etc/freeradius et beaucoup vont devoir être modifiés.

Dans /etc/freeradius/, il y a radiusd.conf qui est le fichier principal. Il est copieusement documenté et fait par défaut appel à quelques autres dont nous n'aurons pas besoin. Voyons ceci :

```
# cat radiusd.conf | egrep -v -e '[[:blank:]]*#|^$' | grep \$INCLUDE
$INCLUDE proxy.conf
$INCLUDE clients.conf
$INCLUDE snmp.conf
$INCLUDE eap.conf
$INCLUDE policy.conf
$INCLUDE sites-enabled/
```

Dans notre cas très simple :

- proxy.conf ne nous servira pas ;
- clients.conf de même. En effet, ce fichier contient les « nas » et nous allons les mettre dans la base MySQL ;
- snmp.conf ne nous sera d'aucune utilité ;
- policy.conf probablement non plus, mais nous le garderons « d'usine ».

En revanche :

- eap.conf nous servira à configurer le mode « eap » ;
- sites-enabled/ va contenir des liens symboliques vers des fichiers placés dans sites-availables/ à la mode d'Apache2. Dans notre cas (très simple, rappelons-le), un seul site sera « available » et s'appellera poétiquement default.

Il reste enfin le fichier sql.conf qui sera nécessaire pour l'usage de MySQL et qui fait lui-même appel à /etc/freeradius/sql/mysql/dialup.conf que nous nous garderons de toucher, bien que dans notre cas (très simple), il serait possible de le dégraisser quelque peu.

Finalement, nous avons à voir et à modifier :

- /etc/freeradius/radiusd.conf ;
- /etc/freeradius/eap.conf ;
- /etc/freeradius/sql.conf ;
- /etc/freeradius/sites-available/default ;

Et nous supprimerons dans /etc/freeradius/sites-enabled/ tout lien qui ne pointera pas sur /etc/freeradius/sites-available/default.

Lorsque je vous disais qu'il y a de quoi s'amuser...

4-1-3-2-1 - radiusd.conf

Première chose à faire :

```
cd /etc/freeradius
mv radiusd.conf radiusd.conf.dist
```

Deuxième chose à faire : **lire le contenu de radiusd.conf.dist, histoire de comprendre un peu ce que l'on va faire par la suite...**

Troisième chose :

```
cat radiusd.conf.dist | egrep -v -e '^[[:blank:]]*#|^$' > radiusd.conf
```

De manière à ne pas se bousiller les yeux à chercher les lignes « utiles » dans la forêt de commentaires.

Nous obtenons quelque chose qui ressemble à :

```
prefix = /usr
exec_prefix = /usr
sysconfdir = /etc
localstatedir = /var
sbindir = ${exec_prefix}/sbin
logdir = /var/log/freeradius
raddbdir = /etc/freeradius
radacctdir = ${logdir}/radacct
confdir = ${raddbdir}
run_dir = ${localstatedir}/run/freeradius
db_dir = $(raddbdir)
libdir = /usr/lib/freeradius
pidfile = ${run_dir}/freeradius.pid
user = freerad
group = freerad
max_request_time = 30
cleanup_delay = 5
max_requests = 1024
listen {
        type = auth
        ipaddr = *
        port = 0
}
listen {
        ipaddr = *
        port = 0
        type = acct
}
hostname_lookups = no
allow_core_dumps = no
regular_expressions     = yes
extended_expressions    = yes
log {
        destination = files
        file = ${logdir}/radius.log
        syslog_facility = daemon
        stripped_names = no
        auth = no
        auth_badpass = no
        auth_goodpass = no
}
checkrad = ${sbindir}/checkrad
security {
        max_attributes = 200
        reject_delay = 1
        status_server = yes
}
#proxy_requests  = yes
#$INCLUDE proxy.conf
#$INCLUDE clients.conf
snmp    = no
#$INCLUDE snmp.conf
thread pool {
        start_servers = 5
        max_servers = 32
        min_spare_servers = 3
        max_spare_servers = 10
        max_requests_per_server = 0
}
modules {
#       pap {
#               auto_header = no
#       }
        chap {
                authtype = CHAP
        }
#       pam {
#               pam_auth = radiusd
#       }
#       unix {
```

```
#               radwtmp = ${logdir}/radwtmp
#       }
$INCLUDE eap.conf
        mschap {
        }
        ldap {
                server = "ldap.your.domain"
                basedn = "o=My Org,c=UA"
                filter = "(uid=%{Stripped-User-Name:-%{User-Name}})"
                ldap_connections_number = 5
                timeout = 4
                timelimit = 3
                net_timeout = 1
                tls {
                        start_tls = no
                }
                dictionary_mapping = ${confdir}/ldap.attrmap
                edir_account_policy_check = no
        }
        realm IPASS {
                format = prefix
                delimiter = "/"
        }
        realm suffix {
                format = suffix
                delimiter = "@"
        }
        realm realmpercent {
                format = suffix
                delimiter = "%"
        }
        realm ntdomain {
                format = prefix
                delimiter = "\\"
        }
        checkval {
                item-name = Calling-Station-Id
                check-name = Calling-Station-Id
                data-type = string
        }

        preprocess {
                huntgroups = ${confdir}/huntgroups
                hints = ${confdir}/hints
                with_ascend_hack = no
                ascend_channels_per_line = 23
                with_ntdomain_hack = no
                with_specialix_jetstream_hack = no
                with_cisco_vsa_hack = no
        }
        files {
                usersfile = ${confdir}/users
                acctusersfile = ${confdir}/acct_users
                preproxy_usersfile = ${confdir}/preproxy_users
                compat = no
        }
        detail {
                detailfile = ${radacctdir}/%{Client-IP-Address}/detail-%Y%m%d
                detailperm = 0600
                header = "%t"
        }
        acct_unique {
                key = "User-Name, Acct-Session-Id, NAS-IP-Address, Client-IP-Address, NAS-Port"
        }
        $INCLUDE ${confdir}/sql.conf
        radutmp {
                filename = ${logdir}/radutmp
                username = %{User-Name}
                case_sensitive = yes
                check_with_nas = yes
                perm = 0600
                callerid = "yes"
```

```
        }
        radutmp sradutmp {
                filename = ${logdir}/sradutmp
                perm = 0644
                callerid = "no"
        }
        attr_filter attr_filter.post-proxy {
                attrsfile = ${confdir}/attrs
        }
        attr_filter attr_filter.pre-proxy {
                attrsfile = ${confdir}/attrs.pre-proxy
        }
        attr_filter attr_filter.access_reject {
                key = %{User-Name}
                attrsfile = ${confdir}/attrs.access_reject
        }
        attr_filter attr_filter.accounting_response {
                key = %{User-Name}
                attrsfile = ${confdir}/attrs.accounting_response
        }
        counter daily {
                filename = ${db_dir}/db.daily
                key = User-Name
                count-attribute = Acct-Session-Time
                reset = daily
                counter-name = Daily-Session-Time
                check-name = Max-Daily-Session
                reply-name = Session-Timeout
                allowed-servicetype = Framed-User
                cache-size = 5000
        }
        always fail {
                rcode = fail
        }
        always reject {
                rcode = reject
        }
        always noop {
                rcode = noop
        }
        always handled {
                rcode = handled
        }
        always updated {
                rcode = updated
        }
        always notfound {
                rcode = notfound
        }
        always ok {
                rcode = ok
                simulcount = 0
                mpp = no
        }
        expr {
        }
        digest {
        }
        expiration {
                reply-message = "Password Has Expired\r\n"
        }
        logintime {
                reply-message = "You are calling outside your allowed timespan\r\n"
                minimum-timeout = 60
        }
        exec {
                wait = yes
                input_pairs = request
                shell_escape = yes
                output = none
        }
        exec echo {
```

```
                wait = yes
                program = "/bin/echo %{User-Name}"
                input_pairs = request
                output_pairs = reply
                shell_escape = yes
        }
        ippool main_pool {
                range-start = 192.168.1.1
                range-stop = 192.168.3.254
                netmask = 255.255.255.0
                cache-size = 800
                session-db = ${db_dir}/db.ippool
                ip-index = ${db_dir}/db.ipindex
                override = no
                maximum-timeout = 0
        }
        policy {
               filename = ${confdir}/policy.txt
        }
}
instantiate {
        exec
        expr
        expiration
        logintime
}
$INCLUDE policy.conf
$INCLUDE sites-enabled/
```

Il y a dans ce fichier plein de choses que nous pourrions enlever, car elles ne nous servent à rien (dans notre cas...). Les lignes surlignées montrent ce qu'il est nécessaire de modifier pour nos besoins.

4-1-3-2-2 - sites-available/default

Assez peu de choses dans ce fichier, compte tenu de la simplicité de nos besoins :

```
authorize {
        preprocess
        eap {
                ok = return
        }
        sql
}
authenticate {
        Auth-Type CHAP {
                chap
        }
        eap
}
session {
        sql
}
```

4-1-3-2-3 - eap.conf

Il s'agit d'indiquer que nous utiliserons tls et donner le chemin d'accès aux certificats de la racine et du serveur, ainsi que la clé privée du serveur, qui peut ici être protégée par un mot de passe.

Nous avons décidé, en préparant notre système Wi-Fi, d'utiliser EAP-TLS pour l'authentification des utilisateurs. Lors de la création des certificats pour WPA2, nous avons créé :

- root_maison_CA-cacert.pem qui est le certificat de notre racine de confiance ;
- sysop@maison.mrs-cert.pem qui est le certificat du serveur FreeRADIUS. Nous l'avons créé de manière à ce qu'il contienne la clé privée du serveur.

Nous allons utiliser ici ces deux certificats, qu'il faut placer dans le répertoire /etc/freeradius/certs.

De même, nous pouvons y créer le fichier dh (pour l'échange « Diffie-Hellman ») :

```
openssl dhparam -check -text -5 512 -out dh
```

Ainsi qu'un fichier random :

```
dd if=/dev/urandom of=random count=2
```

Ce répertoire devrait finalement contenir :

```
/etc/freeradius/certs# ls -l
total 12
-rw-r----- 1 root freerad    0 2007-03-12 11:11 dh
-rw-r----- 1 root freerad 3242 2007-03-12 15:38 maison.mrs-cert.pem
-rw-r----- 1 root freerad 1024 2007-03-12 11:11 random
-rw-r----- 1 root freerad 2610 2007-03-12 15:25 root_maison_CA-cacert.pem
```

Faites attention aux droits d'accès des fichiers de ce répertoire. Il suffit maintenant de modifier eap.conf de la sorte :

```
        eap {
                default_eap_type = tls
                timer_expire     = 60
                ignore_unknown_eap_types = no
                cisco_accounting_username_bug = no
                tls {
                        private_key_password = epikoi
                        private_key_file = ${raddbdir}/certs/maison.mrs-cert.pem
                        certificate_file = ${raddbdir}/certs/maison.mrs-cert.pem
                        CA_file = ${raddbdir}/certs/Root_maison_CA-cacert.pem
                        CA_path = ${raddbdir}/certs/
                        dh_file = ${raddbdir}/certs/dh
                        random_file = ${raddbdir}/certs/random
                        fragment_size = 1024
                        include_length = yes
                        check_crl = no
                }

                mschapv2 {
                }
        }
```

4-1-3-2-4 - sql.conf

Nous avons une base MySQL radius et un utilisateur du même nom, capable de lire dans cette base :

```
sql {
    database = "mysql"
    driver = "rlm_sql_${database}"
    server = "localhost"
    login = "radius"
    password = "epikoi"
    radius_db = "radius"
    acct_table1 = "radacct"
    acct_table2 = "radacct"
    postauth_table = "radpostauth"
    authcheck_table = "radcheck"
    authreply_table = "radreply"
    groupcheck_table = "radgroupcheck"
    groupreply_table = "radgroupreply"
    usergroup_table = "radusergroup"
    deletestalesessions = yes
    sqltrace = no
```

```
    sqltracefile = ${logdir}/sqltrace.sql
    num_sql_socks = 5
    connect_failure_retry_delay = 60
    nas_table = "nas"
    $INCLUDE sql/${database}/dialup.conf
    readclients = yes
}
```

Le fichier inclus dialup.conf pourrait être copieusement dégraissé dans notre cas, mais nous n'y toucherons pas.

4-1-4 - Vérifions...

4-1-4-1 - Essai chap

Nous créons un « authenticator » de test dans la table « nas » :

```
echo "INSERT INTO nas(nasname,shortname,secret) VALUES ('127.0.0.1','localhost','naspassword');" |
 mysql -u root -p radius
```

Nous créons un utilisateur de test dans « radcheck » :

```
echo "INSERT INTO radcheck(UserName,Attribute,op,Value) VALUES ('test0','Cleartext-
Password',':=','userpassword');" | mysql -u root -p radius
```

Enfin nous démarrons freeradius en mode « debug », dans une console avec :

```
freeradius -X
```

Le mode « debug » s'avère très volubile, mais instructif :

```
FreeRADIUS Version 2.0.4, for host i486-pc-linux-gnu, built on Feb 22 2009 at 16:19:09
Copyright (C) 1999-2008 The FreeRADIUS server project and contributors.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
You may redistribute copies of FreeRADIUS under the terms of the
GNU General Public License.
Starting - reading configuration files ...
including configuration file /etc/freeradius/radiusd.conf
including configuration file /etc/freeradius/eap.conf
including configuration file /etc/freeradius/sql.conf
including configuration file /etc/freeradius/sql/mysql/dialup.conf
including configuration file /etc/freeradius/policy.conf
including files in directory /etc/freeradius/sites-enabled/
including configuration file /etc/freeradius/sites-enabled/default
including dictionary file /etc/freeradius/dictionary
main {
        prefix = "/usr"
        localstatedir = "/var"
        logdir = "/var/log/freeradius"
        libdir = "/usr/lib/freeradius"
        radacctdir = "/var/log/freeradius/radacct"
        hostname_lookups = no
        max_request_time = 30
        cleanup_delay = 5
        max_requests = 1024
        allow_core_dumps = no
        pidfile = "/var/run/freeradius/freeradius.pid"
        user = "freerad"
        group = "freerad"
        checkrad = "/usr/sbin/checkrad"
        debug_level = 0
        proxy_requests = yes
 security {
        max_attributes = 200
```

```
        reject_delay = 1
        status_server = yes
 }
}
radiusd: #### Loading Realms and Home Servers ####
radiusd: #### Instantiating modules ####
 instantiate {
 Module: Linked to module rlm_exec
 Module: Instantiating exec
  exec {
        wait = yes
        input_pairs = "request"
        shell_escape = yes
  }
 Module: Linked to module rlm_expr
 Module: Instantiating expr
 Module: Linked to module rlm_expiration
 Module: Instantiating expiration
  expiration {
        reply-message = "Password Has Expired  "
  }
 Module: Linked to module rlm_logintime
 Module: Instantiating logintime
  logintime {
        reply-message = "You are calling outside your allowed timespan  "
        minimum-timeout = 60
  }
 }
radiusd: #### Loading Virtual Servers ####
server {
 modules {
 Module: Checking authenticate {...} for more modules to load
 Module: Linked to module rlm_eap
 Module: Instantiating eap
  eap {
        default_eap_type = "tls"
        timer_expire = 60
        ignore_unknown_eap_types = no
        cisco_accounting_username_bug = no
  }
 Module: Linked to sub-module rlm_eap_tls
 Module: Instantiating eap-tls
   tls {
        rsa_key_exchange = no
        dh_key_exchange = yes
        rsa_key_length = 512
        dh_key_length = 512
        verify_depth = 0
        CA_path = "/etc/freeradius/certs/"
        pem_file_type = yes
        private_key_file = "/etc/freeradius/certs/radius.eme.org-cert.pem"
        certificate_file = "/etc/freeradius/certs/radius.eme.org-cert.pem"
        CA_file = "/etc/freeradius/certs/Root_EME_CA-cacert.pem"
        private_key_password = "ph34rl3r4dius"
        dh_file = "/etc/freeradius/certs/dh"
        random_file = "/etc/freeradius/certs/random"
        fragment_size = 1024
        include_length = yes
        check_crl = no
   }
 Module: Linked to sub-module rlm_eap_mschapv2
 Module: Instantiating eap-mschapv2
   mschapv2 {
        with_ntdomain_hack = no
   }
 Module: Linked to module rlm_chap
 Module: Instantiating chap
 Module: Checking authorize {...} for more modules to load
 Module: Linked to module rlm_preprocess
 Module: Instantiating preprocess
  preprocess {
        huntgroups = "/etc/freeradius/huntgroups"
```

```
        hints = "/etc/freeradius/hints"
        with_ascend_hack = no
        ascend_channels_per_line = 23
        with_ntdomain_hack = no
        with_specialix_jetstream_hack = no
        with_cisco_vsa_hack = no
        with_alvarion_vsa_hack = no
  }
 Module: Linked to module rlm_sql
 Module: Instantiating sql
  sql {
        driver = "rlm_sql_mysql"
        server = "localhost"
        port = ""
        login = "radius"
        password = "epikoi"
        radius_db = "radius"
        read_groups = yes
        sqltrace = yes
        sqltracefile = "/var/log/freeradius/sqltrace.sql"
        readclients = yes
        deletestalesessions = yes
        num_sql_socks = 5
        sql_user_name = "%{User-Name}"
        default_user_profile = ""
        nas_query = "SELECT id, nasname, shortname, type, secret FROM nas"
        authorize_check_query = "SELECT id, username, attribute, value, op FROM radcheck WHERE username
 = '%{SQL-User-Name}' ORDER BY id"
        authorize_reply_query = "SELECT id, username, attribute, value, op FROM radreply WHERE username
 = '%{SQL-User-Name}' ORDER BY id"
        authorize_group_check_query = "SELECT id, groupname, attribute, value, op FROM radgroupcheck
 WHERE groupname = '%{Sql-Group}' ORDER BY id"
        authorize_group_reply_query = "SELECT id, groupname, attribute, value, op FROM radgroupreply
 WHERE groupname = '%{Sql-Group}' ORDER BY id"
        accounting_onoff_query = ""
        accounting_update_query = ""
        accounting_update_query_alt = ""
        accounting_start_query = ""
        accounting_start_query_alt = ""
        accounting_stop_query = ""
        accounting_stop_query_alt = ""
        group_membership_query = "SELECT groupname FROM radusergroup WHERE username = '%{SQL-User-
Name}' ORDER BY priority"
        connect_failure_retry_delay = 60
        simul_count_query = ""
        simul_verify_query = ""
        postauth_query = "INSERT INTO radpostauth (username, pass, reply, authdate) VALUES ('%{User-
Name}','%{%{User-Password}:-%{Chap-Password}}','%{reply:Packet-Type}', '%S')"
        safe-characters = "@abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.-_: /"
  }
rlm_sql (sql): Driver rlm_sql_mysql (module rlm_sql_mysql) loaded and linked
rlm_sql (sql): Attempting to connect to radius@localhost:/radius
rlm_sql (sql): starting 0
rlm_sql (sql): Attempting to connect rlm_sql_mysql #0
rlm_sql_mysql: Starting connect to MySQL server for #0
rlm_sql (sql): Connected new DB handle, #0
rlm_sql (sql): starting 1
rlm_sql (sql): Attempting to connect rlm_sql_mysql #1
rlm_sql_mysql: Starting connect to MySQL server for #1
rlm_sql (sql): Connected new DB handle, #1
rlm_sql (sql): starting 2
rlm_sql (sql): Attempting to connect rlm_sql_mysql #2
rlm_sql_mysql: Starting connect to MySQL server for #2
rlm_sql (sql): Connected new DB handle, #2
rlm_sql (sql): starting 3
rlm_sql (sql): Attempting to connect rlm_sql_mysql #3
rlm_sql_mysql: Starting connect to MySQL server for #3
rlm_sql (sql): Connected new DB handle, #3
rlm_sql (sql): starting 4
rlm_sql (sql): Attempting to connect rlm_sql_mysql #4
rlm_sql_mysql: Starting connect to MySQL server for #4
rlm_sql (sql): Connected new DB handle, #4
```

```
rlm_sql (sql): Processing generate_sql_clients
rlm_sql (sql) in generate_sql_clients: query is SELECT id, nasname, shortname, type, secret FROM nas
rlm_sql (sql): Reserving sql socket id: 4
rlm_sql_mysql: query:  SELECT id, nasname, shortname, type, secret FROM nas
rlm_sql (sql): Read entry nasname=127.0.0.1,shortname=localhost,secret=naspassword
rlm_sql (sql): Adding client 127.0.0.1 (localhost, server=) to clients list
rlm_sql (sql): Released sql socket id: 4
 Module: Checking session {...} for more modules to load
 }
}
radiusd: #### Opening IP addresses and Ports ####
listen {
        type = "auth"
        ipaddr = *
        port = 0
}
listen {
        type = "acct"
        ipaddr = *
        port = 0
}
main {
        snmp = no
        smux_password = ""
        snmp_write_access = no
}
Listening on authentication address * port 1812
Listening on accounting address * port 1813
Listening on proxy address * port 1814
Ready to process requests.
```

Notez la liste de tous les fichiers inclus, en cas de problème. Notez également que le serveur, si rien n'a coincé dans la configuration, est prêt à recevoir des requêtes. Notez enfin que tout ceci n'est pas très propre, puisque nous n'utilisons pas ici ni l'accounting ni le proxy, mais que FreeRADIUS va tout de même ouvrir ces ports. Il y aurait pas mal de « tuning » à faire.

Dans une autre console, nous allons essayer une authentification avec l'utilitaire radtest :

```
radtest test0 userpassword 127.0.0.1 0 naspassword
```

Nous devrions obtenir la réponse :

```
Sending Access-Request of id 197 to 127.0.0.1 port 1812
        User-Name = "test0"
        User-Password = "userpassword"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 0
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=197, length=20
```

Dans la console où s'exécute FreeRADIUS en mode « debug » :

```
rad_recv: Access-Request packet from host 127.0.0.1 port 50494, id=197, length=57
        User-Name = "test0"
        User-Password = "userpassword"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 0
+- entering group authorize
++[preprocess] returns ok
  rlm_eap: No EAP-Message, not doing EAP
++[eap] returns noop
        expand: %{User-Name} -> test0
rlm_sql (sql): sql_set_user escaped user --> 'test0'
rlm_sql (sql): Reserving sql socket id: 3
        expand: SELECT id, username, attribute, value, op FROM radcheck WHERE username = '%{SQL-User-
Name}'  ORDER BY id -> SELECT id, username, attribute, value, op FROM radcheck WHERE username = 'test0'
 ORDER BY id
```

```
rlm_sql_mysql: query:  SELECT id, username, attribute, value, op FROM radcheck WHERE username = 'test0'
 ORDER BY id
rlm_sql (sql): User found in radcheck table
        expand: SELECT id, username, attribute, value, op FROM radreply WHERE username = '%{SQL-User-
Name}' ORDER BY id -> SELECT id, username, attribute, value, op FROM radreply  WHERE username = 'test0'
 ORDER BY id
rlm_sql_mysql: query:  SELECT id, username, attribute, value, op FROM radreply WHERE username = 'test0'
 ORDER BY id
        expand: SELECT groupname FROM radusergroup WHERE username = '%{SQL-User-Name}' ORDER BY
 priority -> SELECT groupname FROM radusergroup WHERE username = 'test0' ORDER BY priority
rlm_sql_mysql: query:  SELECT groupname FROM radusergroup WHERE username = 'test0' ORDER BY priority
rlm_sql (sql): Released sql socket id: 3
++[sql] returns ok
auth: type Local
auth: user supplied User-Password matches local User-Password
Login OK: [test0/userpassword] (from client localhost port 0)
Sending Access-Accept of id 197 to 127.0.0.1 port 50494
Finished request 0.
Going to the next request
Waking up in 4.9 seconds.
Cleaning up request 0 ID 197 with timestamp +6
Ready to process requests.
```

Notre authentification par nom d'utilisateur et mot de passe (chap) fonctionne correctement. Il ne nous reste qu'à ajouter dans la table « nas » nos switches , nos points d'accès Wi-Fi, et dans la table « radcheck » toutes nos adresses MAC en guise d'utilisateurs pour le réseau filaire (« UserName » et valeur de l'attribut « Cleartext-Password » identiques).

4-2 - Pour les VLAN, gestion des adresses MAC des clients

Il existe plusieurs méthodes pour collecter les adresses MAC de vos postes clients, par exemple les logiciels d'inventaire de parc (**OCS Inventory NG** , en est un).

4-2-1 - La table « radcheck »

La table qui doit accueillir les « utilisateurs » s'appelle « radcheck ». Voyons de plus près la structure de cette table :

```
mysql> describe radcheck;
+-----------+------------------+------+-----+---------+----------------+
| Field     | Type             | Null | Key | Default | Extra          |
+-----------+------------------+------+-----+---------+----------------+
| id        | int(11) unsigned | NO   | PRI | NULL    | auto_increment |
| UserName  | varchar(64)      | NO   | MUL |         |                |
| Attribute | varchar(32)      | NO   |     |         |                |
| op        | char(2)          | NO   |     | ==      |                |
| Value     | varchar(253)     | NO   |     |         |                |
+-----------+------------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

Entrer dans le détail des nombreuses possibilités de FreeRADIUS nous mènerait beaucoup trop loin. Dans le cadre de notre projet, nous devons créer une ligne par client de la manière suivante :

- Username contiendra l'adresse MAC du client, au format xx:yy:zz:aa:bb:cc, en lettres minuscules ;
- Attribute contiendra le texte : « Cleartext-Password » ;
- op contiendra « := » ;
- Value contiendra également l'adresse MAC du client, au format xx:yy:zz:aa:bb:cc, en lettres minuscules.

Comme pour gérer dans la suite une telle collection de valeurs, il faudra savoir à quel client correspond une adresse MAC, je vous conseille d'ajouter dans cette table une rubrique supplémentaire destinée à contenir, par exemple, le nom du client associé. De plus, les valeurs par défaut de Attribute et de op gagnent à être créées/modifiées comme suit :

```
mysql> describe radcheck;
+-----------+------------------+------+-----+--------------------+----------------+
| Field     | Type             | Null | Key | Default            | Extra          |
+-----------+------------------+------+-----+--------------------+----------------+
| id        | int(11) unsigned | NO   | PRI | NULL               | auto_increment |
| UserName  | varchar(64)      | NO   | MUL |                    |                |
| Attribute | varchar(32)      | NO   |     | Cleartext-Password |                |
| op        | char(2)          | NO   |     | :=                 |                |
| Value     | varchar(253)     | NO   |     |                    |                |
| COMMENT   | varchar(50)      | YES  |     | NULL               |                |
+-----------+------------------+------+-----+--------------------+----------------+
6 rows in set (0.01 sec)
```

Il « suffit » ensuite de peupler cette base avec vos adresses MAC par la méthode qui vous conviendra le mieux. Ceci devrait aboutir à quelque chose de la forme :

```
mysql> select *  from radcheck;
+----+-------------------+--------------------+----+-------------------+-----------------------+
| id | UserName          | Attribute          | op | Value             | COMMENT               |
+----+-------------------+--------------------+----+-------------------+-----------------------+
|  1 | test0             | Cleartext-Password | := | userpassword      | NULL                  |
|  5 | 00:05:5d:df:f4:5b | Cleartext-Password | := | 00:05:5d:df:f4:5b | Poste 1               |
|  7 | 00:13:46:2f:93:f5 | Cleartext-Password | := | 00:13:46:2f:93:f5 | Poste 2               |
|  8 | 00:13:46:2f:93:ef | Cleartext-Password | := | 00:13:46:2f:93:ef | Poste 3               |
|  9 | 00:15:e9:f1:2d:e6 | Cleartext-Password | := | 00:15:e9:f1:2d:e6 | Poste 4               |
....
```

Il n'est pas nécessaire de relancer FreeRADIUS après modification de cette table.

4-2-2 - La table « nas »

Il ne faut pas oublier non plus d'ajouter dans la table « nas » les informations concernant vos switches. La table « nas » est de structure suivante :

```
mysql> describe nas;
+-------------+--------------+------+-----+---------------+----------------+
| Field       | Type         | Null | Key | Default       | Extra          |
+-------------+--------------+------+-----+---------------+----------------+
| id          | int(10)      | NO   | PRI | NULL          | auto_increment |
| nasname     | varchar(128) | NO   | MUL |               |                |
| shortname   | varchar(32)  | YES  |     | NULL          |                |
| type        | varchar(30)  | YES  |     | other         |                |
| ports       | int(5)       | YES  |     | NULL          |                |
| secret      | varchar(60)  | NO   |     | secret        |                |
| community   | varchar(50)  | YES  |     | NULL          |                |
| description | varchar(200) | YES  |     | RADIUS Client |                |
+-------------+--------------+------+-----+---------------+----------------+
8 rows in set (0.00 sec)
```

Nous devons avoir dans cette table, quelque chose qui ressemble à ceci :

```
mysql> select *  from nas;
+----+---------------+-----------+-------+-------+--------------+-----------+----------------------+
| id | nasname       | shortname | type  | ports | secret       | community | description          |
+----+---------------+-----------+-------+-------+--------------+-----------+----------------------+
|  1 | 127.0.0.1     | localhost | other |  NULL | naspassword  | NULL      | RADIUS Client        |
|  2 | 192.168.10.11 | sw1       | other |  NULL | chutt        | NULL      | switch 1             |
...
```

Notez qu'une modification de la table « nas » nécessite (pour l'instant) un redémarrage de FreeRADIUS.

Normalement, tout devrait fonctionner correctement. Le client d'adresse MAC 00:05:5d:df:f4:5b devrait se retrouver sur le VLAN d'IP 2 (PARADIS_VLAN) alors qu'un client d'adresse MAC ne figurant pas dans la table « radcheck » se retrouvera dans le VLAN d'ID 3 (ENFER_VLAN).

Si vous rencontrez des problèmes, la première chose à faire est d'arrêter FreeRADIUS, puis de le relancer en « foreground » (avant-plan) en mode « debug », par la commande :

```
freeradius -X
```

ou, pour encore plus de détails :

si, lors de la connexion d'un client, vous n'observez rien, c'est tout simplement que le switch ne dialogue pas avec le radius. Il faut alors en trouver la raison ;

si le dialogue démarre, il vous faudra déchiffrer le discours pour trouver la raison du dysfonctionnement. Les raisons les plus probables étant :

- le radius ne reconnait pas le switch (problème de secret partagé, de configuration du radius sur le switch, de configuration du switch dans la table « nas » du radius),
- le radius ne trouve pas le client dans sa base (erreur dans la table « radcheck »).

4-3 - Pour WPA2, configuration de eap

4-3-1 - La table « nas »

Il nous faut ajouter notre borne Wi-Fi dans la table « nas ». Ici, nous utilisons une borne de type Netgear GW302, qui aura dans notre réseau l'adresse IP 192.168.10.3, comme nom « netgear » et qui utilisera « re-chutt » comme secret partagé avec le serveur radius. Nous devrions avoir dans notre table, quelque chose de ce genre :

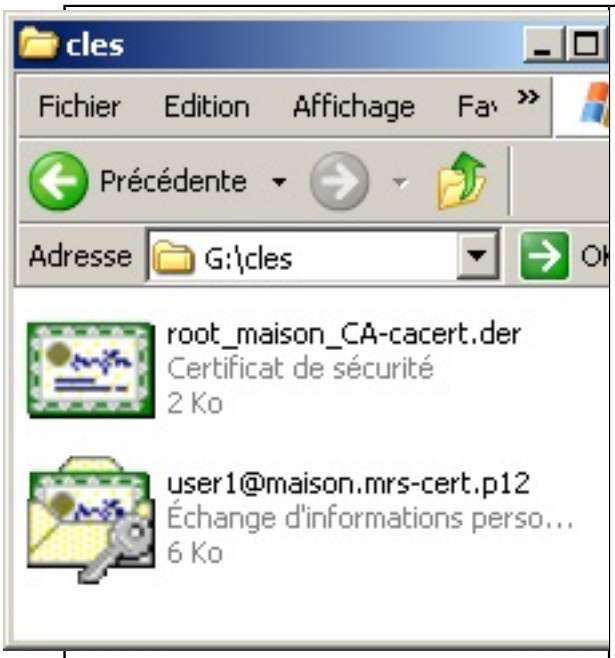
```
mysql> select *  from nas;
+----+---------------+-----------+-------+-------+--------------+-----------+---------------------+
| id | nasname       | shortname | type  | ports | secret       | community | description         |
+----+---------------+-----------+-------+-------+--------------+-----------+---------------------+
|  1 | 127.0.0.1     | localhost | other |  NULL | password     | NULL      | RADIUS Client       |
|  2 | 192.168.10.11 | sw1       | other |  NULL | chutt        | NULL      | switch 1            |
|  3 | 192.168.10.3  | netgear   | other |  NULL | re-epikoi    | NULL      | Borne wi-fi         |
...
```

4-3-2 - Gestion des certificats des clients

Ici, la configuration va être extrêmement simple. Comme nous avons choisi d'utiliser WPA2-TLS, qui nécessite un certificat chez le client, il n'y aura a priori pas de base de noms d'utilisateurs à construire. Le serveur RADIUS va se contenter de vérifier l'authenticité du certificat présenté par le client. Si un client dispose d'un certificat valide, c'est bien qu'il est autorisé à se connecter.

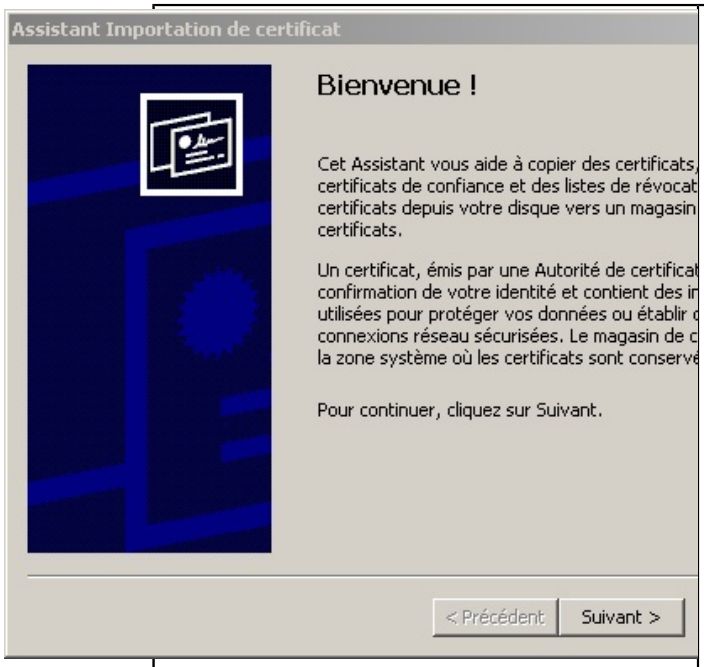
4-3-2-1 - Installation des certificats sur une machine Windows

4-3-2-1-1 - Certificat de l'autorité

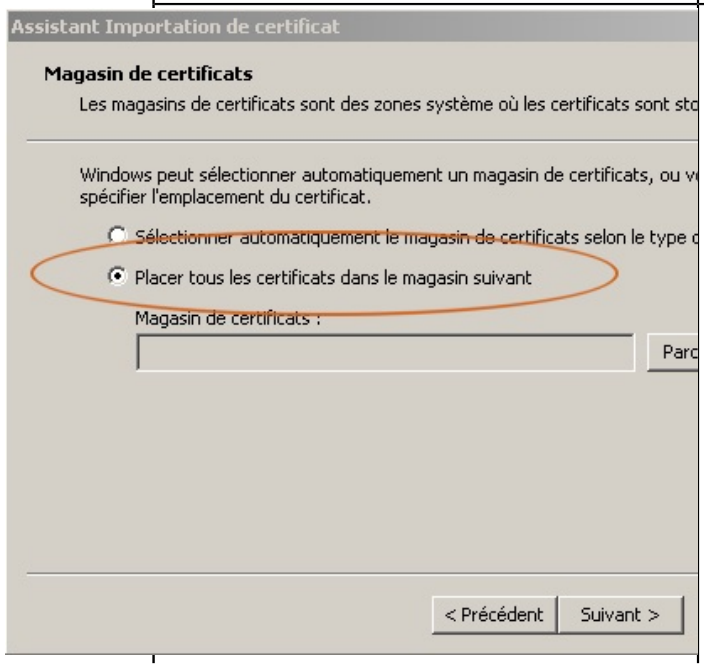


Dans le cas de Windows XP, nous avons besoin du certificat de notre racine de confiance (root_maison_CA) ainsi que du certificat client (user1). Ces certificats sont à installer par l'utilisateur et serviront à authentifier l'utilisateur. Entendez par là deux points importants :
- l'attachement à la borne Wi-Fi ne pourra se faire qu'après ouverture d'une session ;
- seule la session de l'utilisateur qui aura installé les certificats pourra permettre un attachement. Commençons par le certificat de la racine de confiance. Double-clic dessus, ce qui nous amène à :

Cette Autorité de certification étant inconnue de Microsoft, il nous appartient de décider si oui ou non, nous l'acceptons. Nous allons bien sûr installer ce certificat, non sans être certain que c'est bien le bon. Autrement dit, il nous appartient de vérifier que ce certificat provient bien de chez nous. « Installer ce certificat... » va nous conduire à un assistant :



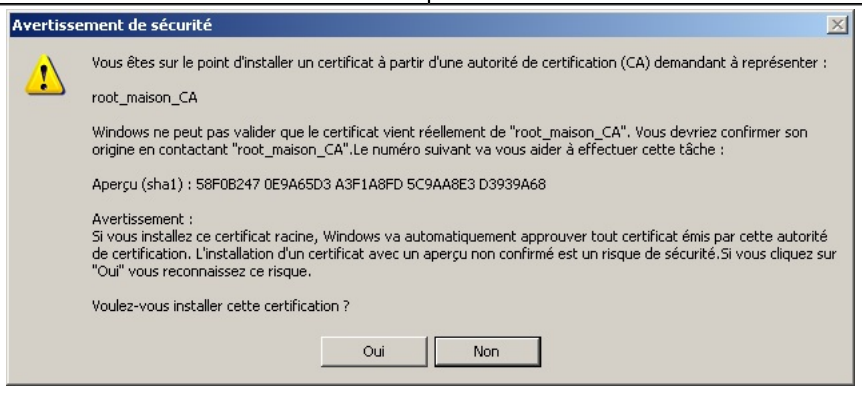
dont le texte explicatif ne nécessite aucun commentaire. Suivant :



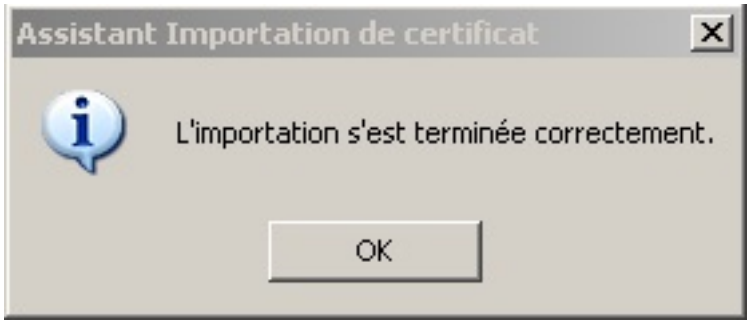
Nous allons choisir nous-même l'emplacement où ce certificat sera stocké. « Parcourir... »

Nous choisissons logiquement les Autorités de certification racines de confiance.

L'assistant a fini son travail.

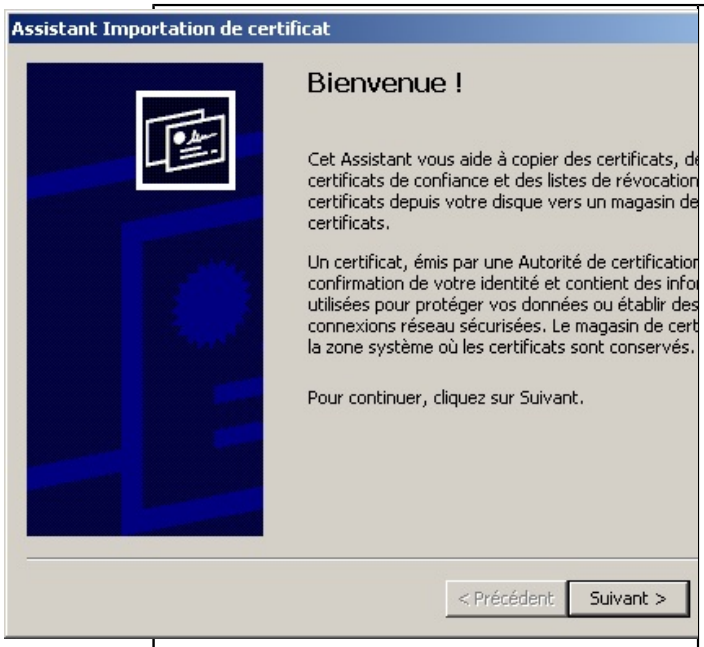


Mais, tout de même, nous avons encore droit à un bon gros avertissement de sécurité.Encore une fois, si nous sommes certains de l'origine de ce certificat, nous pouvons y aller.

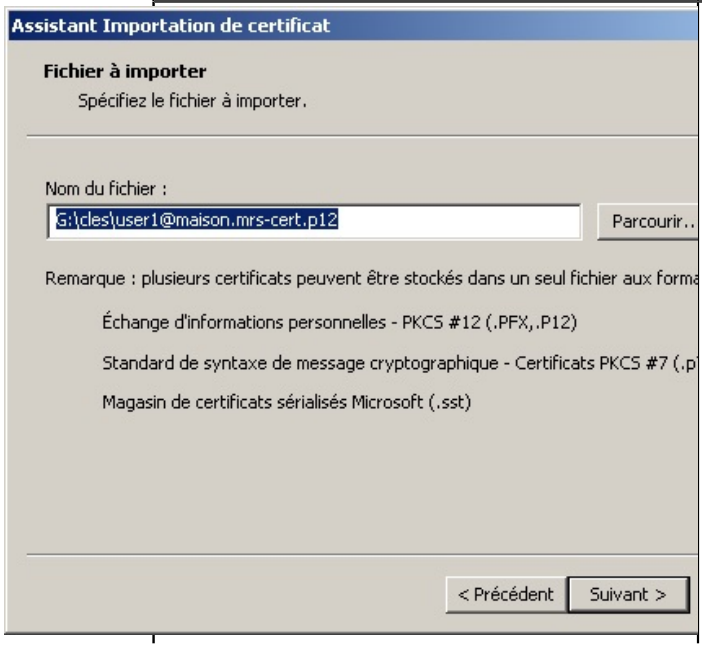


4-3-2-1-2 - Certificat du client

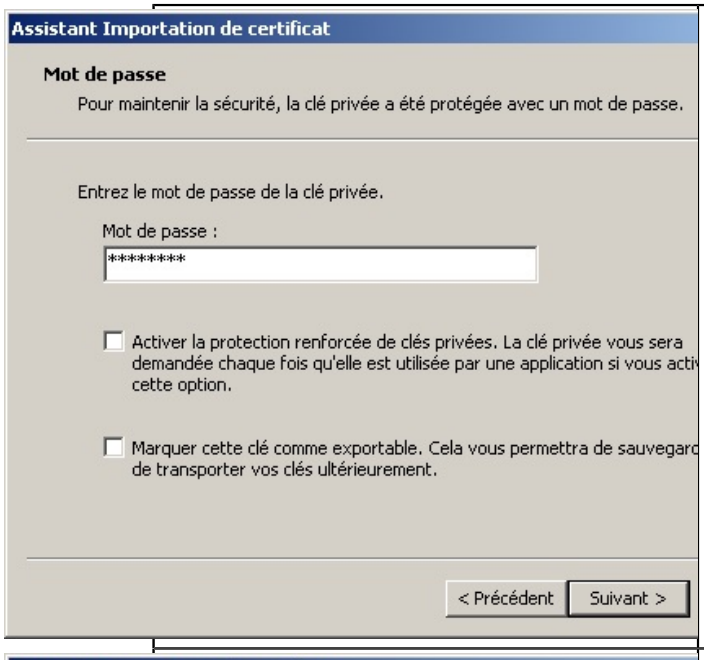
Double-clic sur le certificat client.



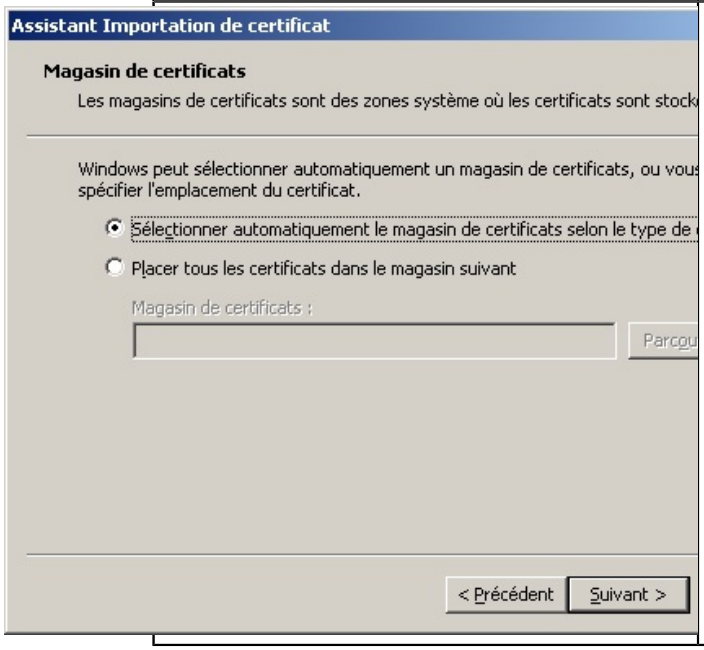
Ceci démarre un nouvel assistant. Suivant >



La seconde permet de vérifier le chemin d'accès au certificat que nous voulons enregistrer. Suivant >



Le mot de passe demandé ici est celui qui a été spécifié lors de l'exportation au format PKCS#12 avec TinyCA (« Export password »).



Ici, nous pouvons choisir de sélectionner automatiquement le magasin. Suivant >

Le reste ne présente pas d'intérêt particulier, l'installation doit se terminer sans encombre.

4-3-2-1-3 - WPA2 et Windows XP

Windows XP, même avec le SP2, ne gère pas WPA2. Il faut lui ajouter un correctif à télécharger chez Microsoft (validation d'intégrité « **Genuine Advantage** » requise). Ce correctif s'appelle : **WindowsXP-KB893357-v2-x86-FRA.exe**. À l'heure où ces lignes sont écrites, ce lien est valide. Sans ce correctif, vous devrez vous contenter de WPA. Ceci n'est pas nécessaire pour les heureux possesseurs de Vista ™.

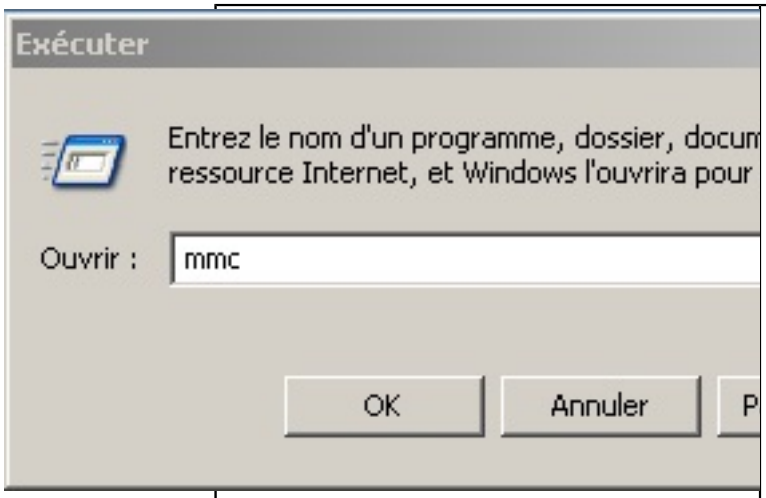
Une fois tout ceci réalisé, avec un peu de chance, la connexion devrait s'établir automatiquement pour l'utilisateur qui a installé les certificats.

4-3-2-1-4 - En cas de problèmes

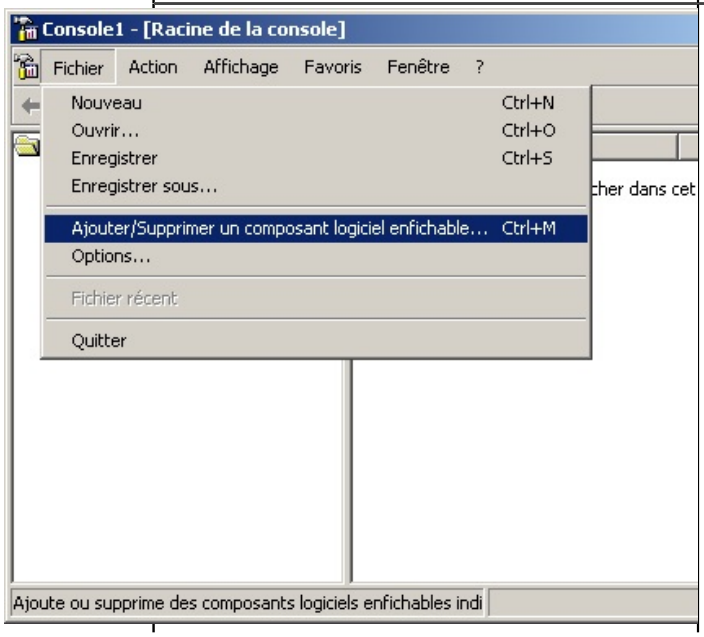
Voici quelques éléments qui devraient vous aider à trouver la solution.

4-3-2-1-5 - Voir les certificats installés

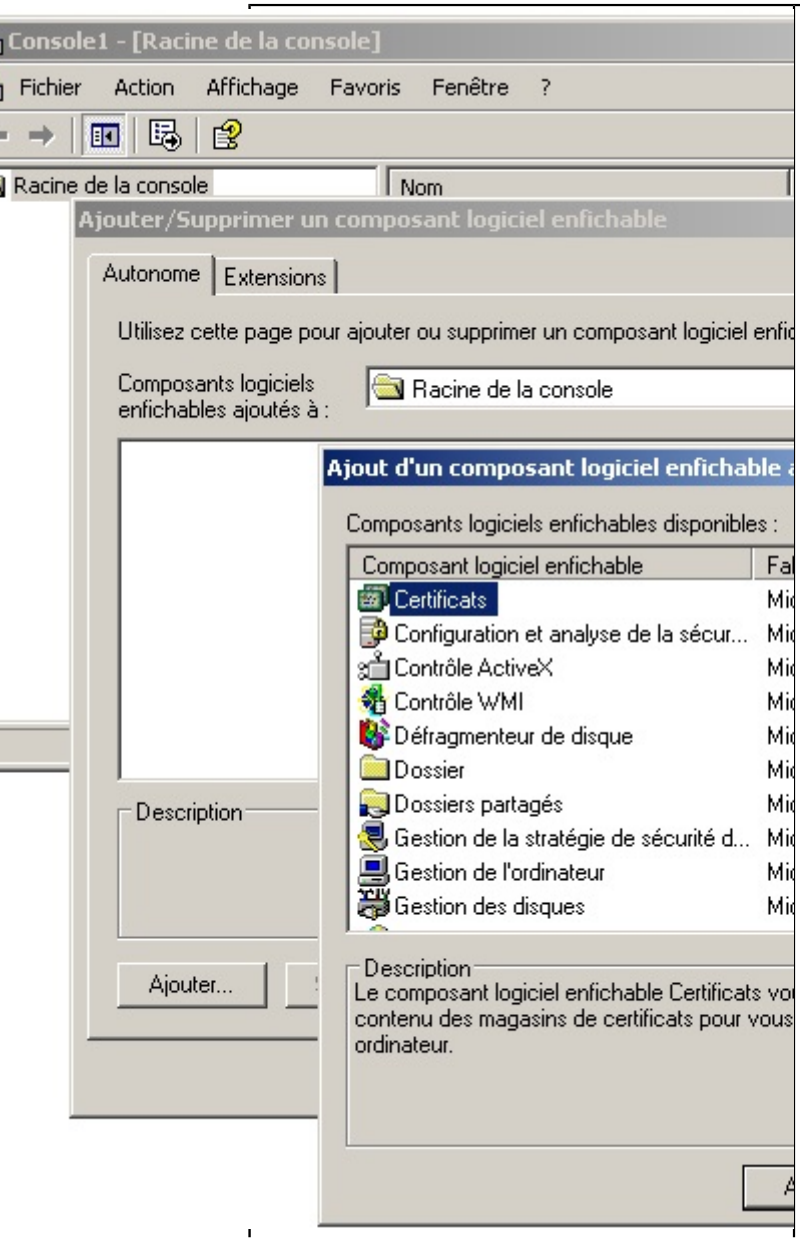
Une console « mmc » va nous venir en aide. Créons une console de gestion des certificats.



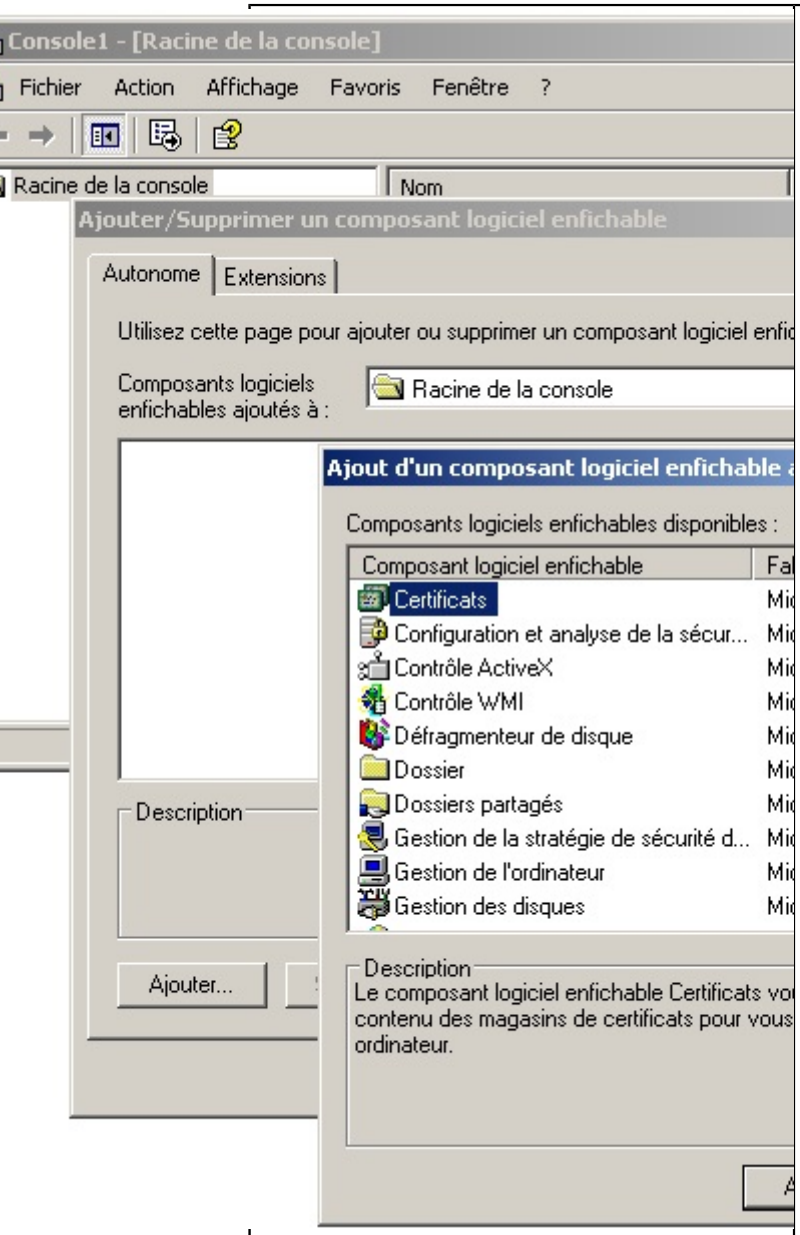
exécuter mmc...



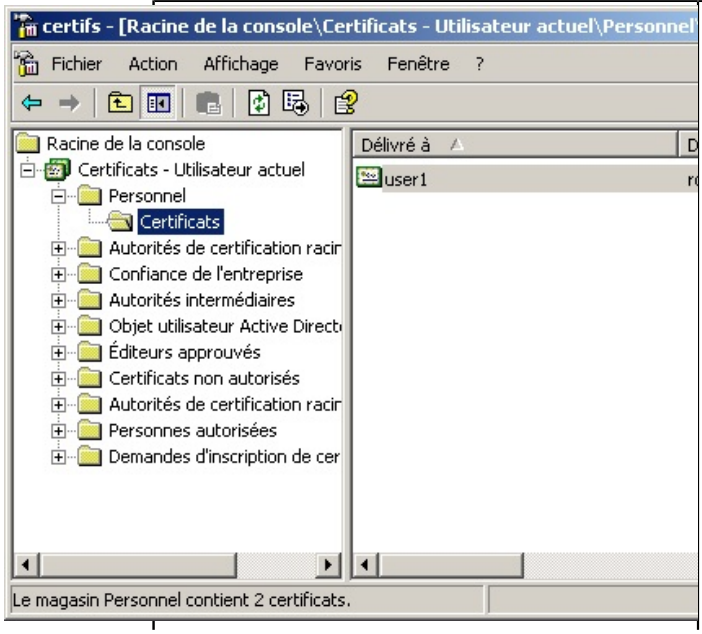
Ajouter un composant logiciel enfichable...



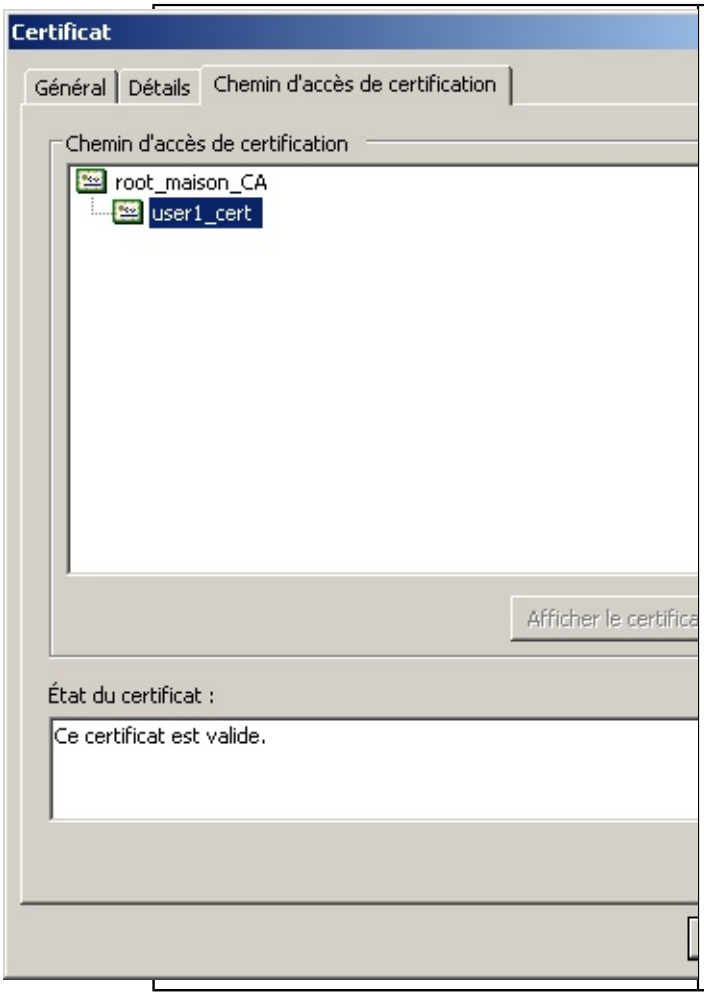
Choisir le composant « certificats »...
« Fermer »



Si vous avez un compte d'administrateur (ce qui, rappelons-le, n'est pas une bonne idée, même si c'est le comportement par défaut de Windows XP), vous aurez droit à choisir s'il s'agit de vos certificats (utilisateur) ou d'autres certificats, comme ceux qui concernent l'ordinateur. Si votre compte est celui d'un simple utilisateur, vous n'aurez pas ce choix et ne pourrez gérer que vos propres certificats, ce qui est suffisant dans notre cas. « Terminer »



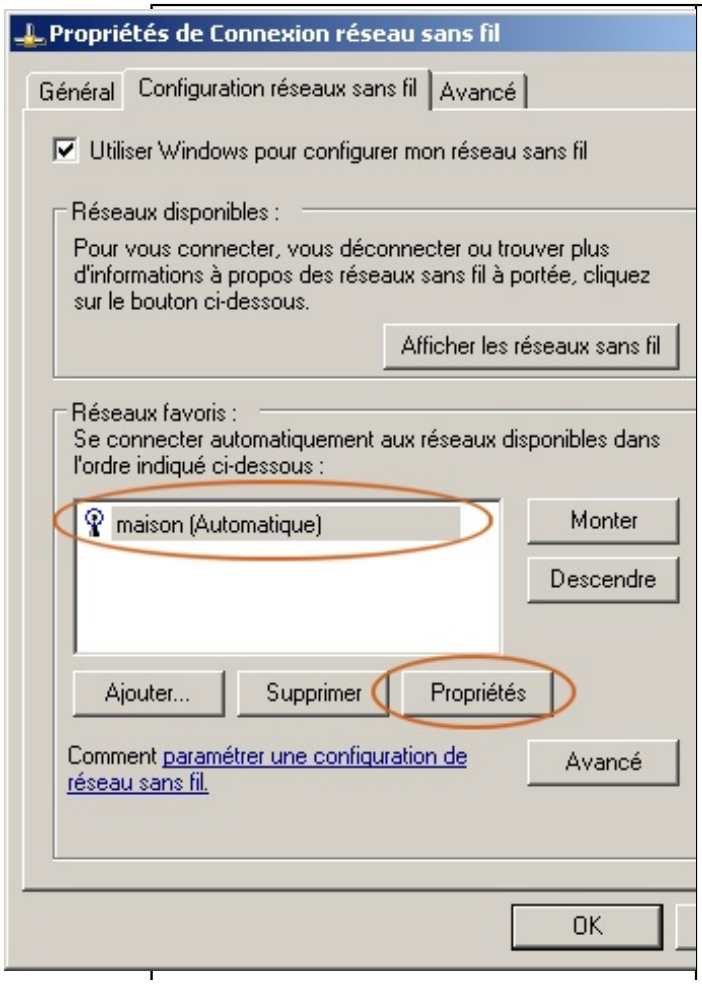
Si nous double-cliquons sur le certificat :



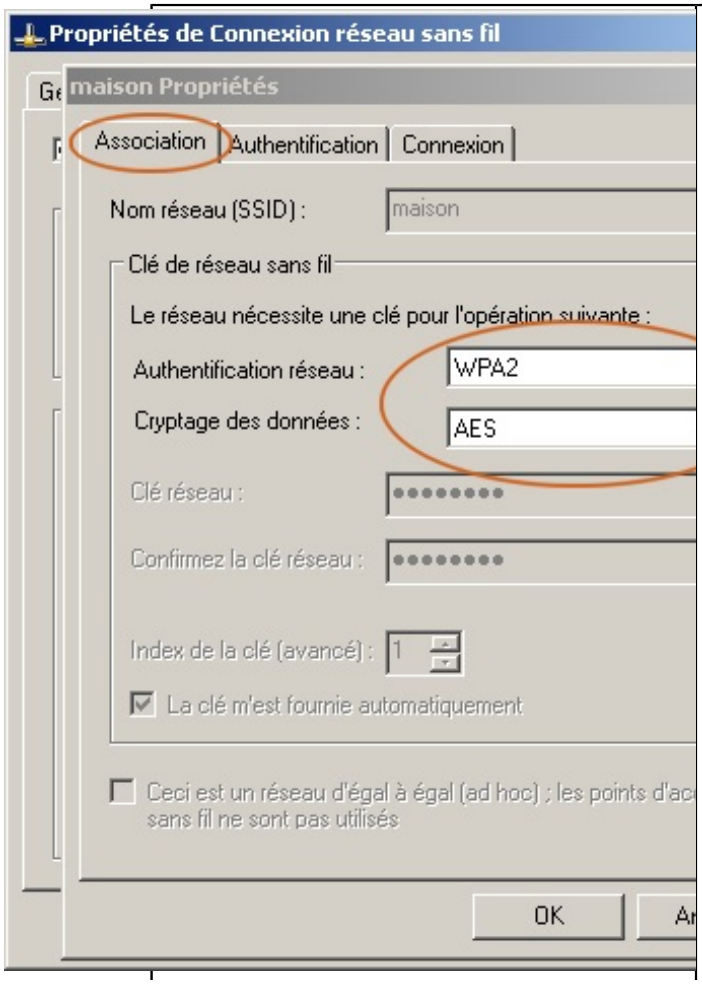
Nous voyons apparaitre la racine de confiance associée.

La suite sous-entend que les connexions WI-FI sont gérées par les outils Windows. Même si l'on peut admettre qu'ils sont plutôt rébarbatifs, peu ergonomiques à première vue, mon expérience personnelle ne m'en a pas fait découvrir de meilleurs parmi les nombreux « gadgets » fournis par les constructeurs de matériel Wi-Fi.

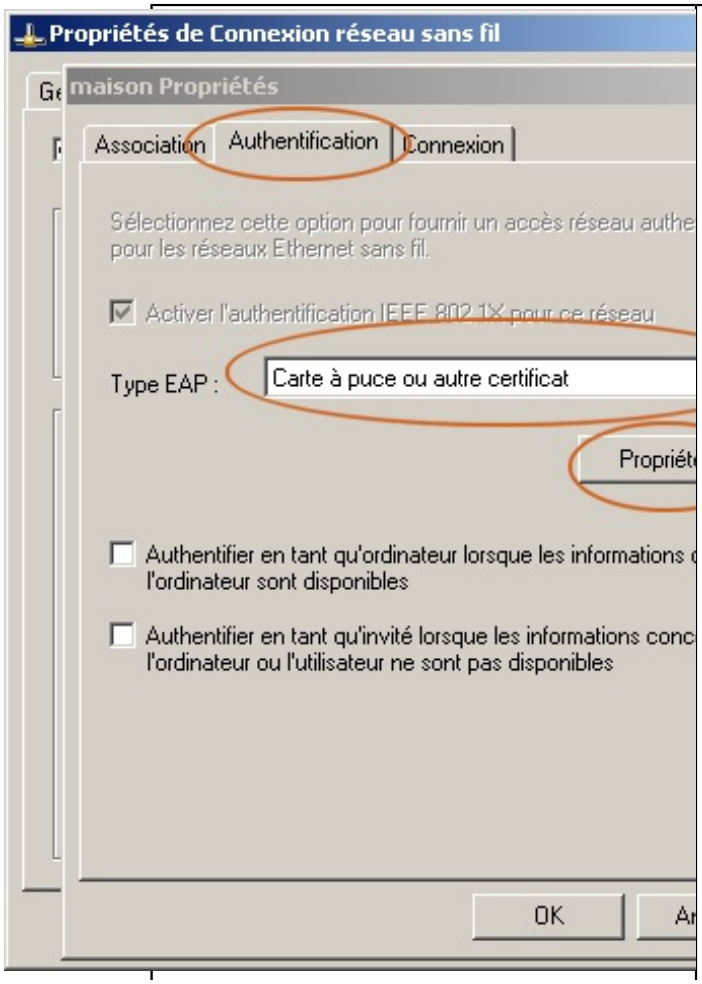
Faites apparaitre les propriétés des connexions réseau sans fil :



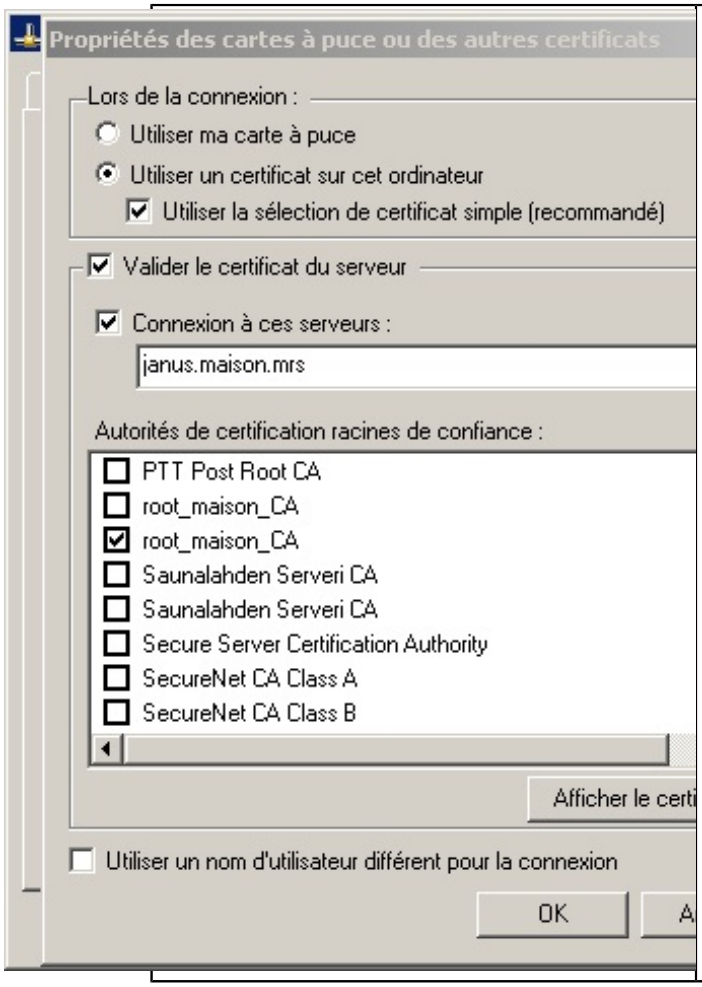
Sélectionnez votre réseau et affichez ses propriétés,



Sur l'onglet « Association », assurez-vous que vous êtes bien sur le mode WPA2, avec un chiffrement AES,



Sur l'onglet « Authentification », le type EAP doit être « Carte à puce ou autre certificat ». Affichez ses propriétés,



Utilisez la sélection de certificats recommandée (simple), indiquez le nom complet du serveur RADIUS, tel qu'il a été défini dans son certificat, sélectionnez enfin le certificat de la racine d'autorité. Ceci devrait enlever toute ambiguïté, en cas de troubles.

4-3-2-2 - Installation du certificat sur une machine Linux

Contrairement à Windows, le moyen le plus simple d'obtenir l'attachement à un réseau Wi-Fi est de le faire au niveau du système. Entendez par là que l'authentification se fera lors du montage du réseau, au démarrage du système, ou lors de l'activation de l'interface Wi-Fi. Autrement dit, ce ne sera pas l'utilisateur qui sera authentifié, mais l'administrateur de la machine.

La manipulation est faite sur une distribution Ubuntu 6.10, où wpa-supplicant est normalement installé par défaut. Il y a sur cette distribution, deux interventions à faire. La première est typique aux distributions basées sur Debian et il vous faudra trouver comment adapter à une autre distribution.

L'interface Wi-Fi est une carte PCMCIA (cardbus) D-Link DWL-G650, qui utilise un chipset Atheros (Driver Madwifi).

4-3-2-2-1 - Copie des certificats

L'installation du paquet wpasupplicant a créé un répertoire /etc/wpa_supplicant/. Nous pouvons créer dedans un répertoire certs et y mettre dedans nos deux certificats :

- root_maison_CA-cacert.pem pour l'autorité de certification ;
- user1@maison.mrs-cert.pem pour le client.

Comme le contenu de ce répertoire est utilisé lors du démarrage, il n'y a aucune raison qu'il soit accessible par quiconque d'autre que root.

4-3-2-2-2 - Configuration de l'interface Wi-Fi

Sur les Distributions Debian et dérivées, il faut agir sur le fichier /etc/network/interfaces :

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loopback network interface
auto lo
iface lo inet loopback

  auto ath0
  iface ath0 inet dhcp
        wpa-driver madwifi
        wpa-conf /etc/wpa_supplicant.conf
  ...
```

4-3-2-2-3 - Configuration de wpa-supplicant

wpa-supplicant mériterait à lui seul tout un chapitre. Son fonctionnement n'est pas d'une évidence extrême, et sa configuration non plus.

Il faut créer un fichier /etc/wpa_supplicant.conf, comme indiqué dans /etc/network/interfaces, qui indiquera, réseau par réseau (ici, un seul suffira), les paramètres nécessaires à l'attachement.

Voici un exemple de configuration dans notre contexte :

```
ctrl_interface=/var/run/wpa_supplicant
ap_scan=1

network={
        ssid="maison"
        scan_ssid=0
        key_mgmt=WPA-EAP
        eap=TLS
        proto=WPA2
        pairwise=CCMP TKIP
        group=CCMP TKIP
        identity="user1"
        ca_cert="/etc/wpa_supplicant/certs/root_maison_CA-cacert.pem"
        client_cert="/etc/wpa_supplicant/certs/user1@maison.mrs-cert.pem"
        private_key="/etc/wpa_supplicant/certs/user1@maison.mrs-cert.pem"
        private_key_passwd="epikoi"
}
```

Il n'y a rien d'incompréhensible dans ce fichier, la difficulté réside surtout dans le fait d'utiliser les bons paramètres. Pour vous aider, vous avez dans le répertoire /usr/share/doc/wpasupplicant/examples/ quelques exemples de configuration ainsi qu'un fichier wpa_supplicant.conf.gz très largement documenté, qui passe en revue tous les paramètres possibles.

4-3-3 - C'est bien, mais...

Cette solution ne nous permet pas de gérer les « coups durs ». Comment faire en effet si l'on apprend qu'un certificat encore valide a été compromis ? Car dans cette configuration, une fois un certificat installé, il sera accepté par FreeRADIUS tant qu'il n'aura pas expiré.

4-4 - Révocations

4-4-1 - Comment gérer les impondérables ?

Dans notre solution globale, nous gérons le réseau filaire par une authentification de type « login/password » où l'adresse MAC joue le rôle des deux composants. Nous disposons d'une base de données MySQL (ou Postgresql) qui recense toutes les adresses MAC connues. Nous avons pu tester son bon fonctionnement en utilisant un « login/password » introduit dans la base de données.

En ce qui concerne le réseau Wi-Fi, nous avons confié à EAP l'authentification, via le protocole TLS. Nous avons vu que, dans ce cas, tout se passe par l'intermédiaire des certificats, et qu'il n'y a aucune information dans la base de données.

Comment faire alors si pour une raison ou une autre, nous devions être amenés à bloquer un utilisateur disposant d'un certificat en cours de validité ? Il peut y avoir plusieurs raisons qui pourraient amener à cette résolution, comme :

- un vol déclaré de la machine sur laquelle le certificat a été installé ;
- un utilisateur autorisé pour une certaine durée, mais qui pour une raison ou une autre, n'a plus rien à faire sur notre réseau, temporairement ou définitivement (non-respect de la charte des utilisateurs, démission, etc.).

Dans notre configuration actuelle, ce type de situation ne peut être géré, il nous faut trouver une solution, si possible pas trop complexe à maintenir.

Deux voies sont à explorer :

- l'utilisation de certificats de révocation ;
- trouver un moyen pour qu'en plus de l'authentification par certificats, le nom d'utilisateur doive être présent dans la base de données pour autoriser l'attachement.

4-4-1-1 - Certificat de révocation

TinyCA sait générer simplement des certificats de révocation, et FreeRADIUS peut être configuré assez simplement pour en tenir compte. Cette solution offre cependant deux gros défauts pour la maintenance :

- à chaque nouvelle suspension de compte, il faut révoquer le certificat correspondant au compte suspendu, recréer un nouveau certificat de révocations, l'exporter, l'installer sur le serveur puis redémarrer le serveur ;
- en cas de suspension provisoire, un certificat révoqué doit être recréé puis réinstallé sur la machine cliente à la fin de la suspension du compte.

Cette solution n'est clairement pas facile à gérer.

4-4-1-2 - Usage de la base de données

4-4-1-2-1 - État des lieux

Actuellement, eap est un mode d'authentification par défaut, si bien que tout certificat présenté, s'il est authentique et non révoqué, sera accepté sans autre contrainte. Si nous trouvons un moyen pour que eap ne soit utilisé que si l'utilisateur est référencé dans la base, les choses deviendraient beaucoup plus simples, il suffirait d'ajouter ou de supprimer une ligne dans la base pour suspendre, temporairement ou non, un compte d'utilisateur, même si le certificat est encore en cours de validité.

Nous verrons que lorsque le client est configuré pour utiliser eap-tls, il présente son certificat, dans tous les cas de figure. Dans ce certificat, il y a le nom de l'utilisateur. Exemple :

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 20 (0x14)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FR, ST=France, L=Marseille, O=Maison, OU=Reseau_maison, CN=root_maison_CA/
emailAddress=chris@maison.mrs
        Validity
            Not Before: Jan  6 12:17:58 2009 GMT
            Not After : Nov 13 12:17:58 2016 GMT
        Subject: C=FR, ST=France, L=Marseille, O=Maison, OU=Reseau_maison, CN=user1/
emailAddress=user1@maison.mrs
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (4096 bit)
...
```

Si nous trouvons un moyen d'expliquer à notre FreeRADIUS que seuls les utilisateurs référencés dans un groupe particulier pourront utiliser EAP, ceci devrait résoudre notre problème.

Pour rappel, nous avons actuellement dans /etc/freeradius/sites-avalable/default les directives suivantes :

```
authorize {
    preprocess
    eap {
        ok = return
    }
    sql
}
authenticate {
    Auth-Type CHAP {
        chap
    }
    eap
}
```

4-4-1-3 - Authentification EAP sélective

Nous allons simplifier /etc/freeradius/sites-avalable/default comme suit :

```
authorize {
    preprocess
    sql
}
...
```

De cette manière, eap ne sera plus autorisé par défaut et, sans informations supplémentaires, plus aucun certificat ne sera accepté puisque eap-tls ne sera plus reconnu comme un moyen d'authentification valide.

Voici ce qu'il va se passer lorsque user1 va présenter son certificat :

```
rad_recv: Access-Request packet from host 192.168.1.254 port 1285, id=0, length=199
        Message-Authenticator = 0xcb9362792c3a93b83a28b6153462c2db
        Service-Type = Framed-User
        User-Name = "user1\000"
        Framed-MTU = 1488
        Called-Station-Id = "00-0F-3D-AB-66-E8:maisonwifi"
        Calling-Station-Id = "00-1F-3C-4B-07-9C"
        NAS-Identifier = "D-Link Access Poi"
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 54Mbps 802.11g"
```

```
        EAP-Message = 0x0200000e016c61702d70726f6673
        NAS-IP-Address = 192.168.1.254
        NAS-Port = 1
        NAS-Port-Id = "STA port # 1"
+- entering group authorize
++[preprocess] returns ok
        expand: %{User-Name} -> user1
rlm_sql (sql): sql_set_user escaped user --> 'user1'
rlm_sql (sql): Reserving sql socket id: 3
        expand: SELECT id, username, attribute, value, op FROM radcheck WHERE username = '%{SQL-User-
Name}' ORDER BY id -> SELECT id, username, attribute, value, op FROM radcheck WHERE username = 'user1'
 ORDER BY id
        expand: SELECT groupname FROM radusergroup WHERE username = '%{SQL-User-Name}' ORDER BY
 priority -> SELECT groupname FROM radusergroup WHERE username = 'user1' ORDER BY priority
rlm_sql (sql): Released sql socket id: 3
rlm_sql (sql): User user1 not found
++[sql] returns notfound
auth: No authenticate method (Auth-Type) configuration found for the request: Rejecting the user
auth: Failed to validate the user.
Login incorrect: [user1\000/] (from client dwl2100ap port 1 cli 00-1F-3C-4B-07-9C)
Delaying reject of request 10 for 1 seconds
Going to the next request
Waking up in 0.9 seconds.
Sending delayed reject for request 10
Sending Access-Reject of id 0 to 192.168.1.254 port 1285
Waking up in 3.9 seconds.
```

Il n'y a pas de mot de passe, il n'y a pas d'autre méthode d'authentification, le client est rejeté.

La notion de groupes existe dans radius. Si nous créons un groupe d'utilisateurs qui ont EAP comme méthode d'authentification, et que nous plaçons dans ce groupe les noms des clients qui ont un certificat en cours de validité, ça devrait fonctionner.

4-4-1-3-1 - Le groupe wifiGroup

Dans la table radgroupcheck, nous créons un groupe que nous appellerons « wifiGroup » auquel nous accordons le type d'authentification EAP :

```
mysql> INSERT INTO radgroupcheck(GroupName,Attribute,op,Value) VALUES ('wifiGroup','Auth-
Type',':=','EAP');
Query OK, 1 row affected (0.00 sec)
```

Il nous reste maintenant à associer dans la table usergroup l'utilisateur « user1 » au groupe « wifiGroup » :

```
mysql> INSERT INTO usergroup(UserName,GroupName) VALUES ('user1','wifiGroup');
Query OK, 1 row affected (0.00 sec)
```

Et à vérifier que notre méthode fonctionne :

```
ad_recv: Access-Request packet from host 192.168.1.254 port 1287, id=5, length=1703
        Message-Authenticator = 0x5a717acfbd03837c04baa4ff01be6399
        Service-Type = Framed-User
        User-Name = "user1\000"
        Framed-MTU = 1488
        State = 0x411d49f94518440d3eeebef8b48bd82c
        Called-Station-Id = "00-0F-3D-AB-66-E8:maisonwifi"
        Calling-Station-Id = "00-1F-3C-4B-07-9C"
        NAS-Identifier = "D-Link Access Poi"
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 54Mbps 802.11g"
        EAP-Message =
 0x020505d20dc000000a901603010a600b00075000074d00074a308207463082052ea003020102020114300d06092a864886f70d01010505
        EAP-Message =
 0x55040613024652310f300d060355040813064672616e6365311230100603550407130 94d61727365696c6c65310c300a060355040a1303
```

```
        EAP-Message =
 0x597d42670715fb9f0dc758c3efca14ea6efdc1ace976f9dc5346205c04e0b42de56aa643dec29cd0e8aa37223b78748490e08979096222
        EAP-Message =
 0x8b9b1ecb74502381f8ce29f3b94ef2808da6438ec64cca64a2644ea659e747b6287bd674b5fcaa4efbcedb4d4b8146c2d831ab675844e2
        EAP-Message =
 0x65301d0603551d0e0416041447069b70658e7d9f0f4ffe350b37543da59b7f1f3081ce0603551d230481c63081c380143cabad8ef6977b
        EAP-Message =
 0x20736572766963652e696e666f40656d652d656e736569676e656d656e742e667230190603551d1104123010810e63616c65636140656d
        NAS-IP-Address = 192.168.1.254
        NAS-Port = 1
        NAS-Port-Id = "STA port # 1"
+- entering group authorize
++[preprocess] returns ok
        expand: %{User-Name} -> user1
rlm_sql (sql): sql_set_user escaped user --> 'user1'
rlm_sql (sql): Reserving sql socket id: 1
        expand: SELECT id, username, attribute, value, op FROM radcheck WHERE username = '%{SQL-User-
Name}' ORDER BY id -> SELECT id, username, attribute, value, op FROM radcheck WHERE username = 'user1'
 ORDER BY id
        expand: SELECT groupname FROM radusergroup WHERE username = '%{SQL-User-Name}' ORDER BY
 priority -> SELECT groupname FROM radusergroup WHERE username = 'user1' ORDER BY priority
        expand: SELECT id, groupname, attribute, Value, op FROM radgroupcheck WHERE groupname = '%{Sql-
Group}' ORDER BY id -> SELECT id, groupname, attribute, Value, op FROM radgroupcheck WHERE groupname =
 'wifiUsers' ORDER BY id
rlm_sql (sql): User found in group wifiUsers
        expand: SELECT id, groupname, attribute, value, op FROM radgroupreply WHERE groupname = '%{Sql-
Group}' ORDER BY id -> SELECT id, groupname, attribute, value, op FROM radgroupreply WHERE groupname =
 'wifiUsers' ORDER BY id
rlm_sql (sql): Released sql socket id: 1
++[sql] returns ok
 rad_check_password: Found Auth-Type EAP
auth: type "EAP"
+- entering group authenticate
 rlm_eap: Request found, released from the list
 rlm_eap: EAP/tls
 rlm_eap: processing type tls
 rlm_eap_tls: Authenticate
 rlm_eap_tls: processing TLS
....
```

Passons sur la suite, notre utilisateur user1 va maintenant disposer de l'authentification de type EAP et le reste va se passer convenablement pour lui.

Cette méthode finalement fort simple nous permet en agissant sur le contenu de la table usergroup d'autoriser ou non un utilisateur présentant un certificat valide sur la bonne foi du nom (CN) présenté par le client, CN qui se trouve dans le certificat. Il nous est désormais possible d'interdire l'accès à un utilisateur disposant d'un certificat en cours de validité, sans avoir besoin de passer par un certificat de révocations.

4-4-2 - Conclusion

Nous sommes loin d'avoir vu tout ce qu'il est possible de faire avec RADIUS, mais nous avons réalisé ce que nous voulions faire.

Dans le cas du réseau filaire, il est également possible d'attribuer un ID de VLAN différent suivant l'utilisateur authentifié, de même qu'il est possible de remplacer EAP-TLS par PEAP, si l'on dispose par exemple d'un annuaire ActiveDirectory et que l'on souhaite que les clients Wi-Fi soient authentifiés avec leur « login/password » du réseau Microsoft, plutôt que par un certificat. La machine qui héberge FreeRadius doit alors être intégrée au domaine Microsoft et pouvoir interroger l'annuaire ActiveDirectory. À mon sens, EAP-TLS reste largement préférable, ne serait-ce qu'à cause des fuites (toujours possibles) de login/passwords.

5 - Remerciements Developpez

Vous pouvez retrouver l'article original ici : **L'Internet Rapide et Permanent**. Christian Caleca a aimablement autorisé l'équipe « **Réseaux** » de **Developpez.com** à reprendre son article. Retrouvez tous les articles de Christian Caleca **sur cette page**.

Nos remerciements à **ClaudeLELOUP** pour sa relecture orthographique.

N'hésitez pas à commenter cet article !