

Table des matières

1	Introduction	3
2	Vulnérabilités de WEP	4
2.1	Le faille de RC4	4
2.2	IV collision	6
2.3	Réalisation de l'attaque	7
2.3.1	Airsnort	7
2.3.2	Déroulement de l'attaque	7
2.3.3	Problèmes d'installation	12
2.3.4	Résultats	13
3	Première approche	15
3.1	Déterminer ses besoins	15
3.2	Problèmes liés aux WLAN	15
3.3	Problèmes liés à la voie et à la vidéo	16
4	RADIUS	17
4.1	Les messages	17
4.2	Critères d'authentification	20
4.3	Base de données mySQL	20
5	802.1x	21
5.1	Port-based Network Access Control	21
6	EAP	23
6.1	Authentification basée sur certificats	23
6.1.1	EAP-TLS	23
6.1.2	PEAP	24
6.1.3	EAP-TTLS	24
6.2	Authentification basée sur mots de passe	24
6.3	Messages EAP	25
7	Schéma d'authentification	27
8	WPA	29
8.1	TKIP	29
9	802.11i	31
9.1	AES	31
9.2	CCMP	32
10	Les solutions des couches supérieures	33
10.1	IPsec	33
10.2	SSL	33

11	<i>Mise en œuvre de la plateforme d'authentification</i>	35
11.1	Matériel	35
11.2	Schéma de la plateforme	36
11.3	Serveur freeRADIUS	36
11.3.1	Installation	37
11.3.2	Configuration	37
11.3.3	Installation MySQL serveur	42
11.3.4	Interface graphique	45
11.4	802.1x et EAP	46
11.5	Côté utilisateur	50
11.6	Mécanismes d'encryption	56
11.7	IPsec	59
12	<i>Mesures des performances</i>	62
12.1	Méthodes	62
12.2	Plateforme	63
12.3	Logiciel de mesure	64
12.4	Types de trafic	65
12.5	Présentation des résultats	65
13	<i>Travail à poursuivre</i>	70
14	<i>Conclusion</i>	71
15	<i>Remerciements</i>	72
16	<i>Références</i>	72
17	<i>Table des figures</i>	74

Annexe 1 : Rapport de semestre

Annexe 2 : Echange des trames RADIUS et EAP pour un challenge MD5

Annexe 3 : Démarrage de radiusd en mode debug

Annexe 4 : Schéma de la base de données RADIUS

1 Introduction

Ce document relate l'entier de mon travail de diplôme. Son but est de décrire et d'expliquer la totalité du travail effectué durant la période du 15 septembre au 18 décembre 2003, soit 12 semaines consécutives. Le travail de diplôme est la suite du projet de semestre et résout les problèmes énoncés lors de ce précédent travail. Il est conseillé de lire au préalable le rapport de semestre en annexe 1, si vous ne vous sentez pas à l'aise au sujet de la norme 802.11 et de ses problèmes liés à la sécurité. Il faut noter que l'étude d'un schéma de sécurité pour Bluetooth a été abandonnée pour des raisons de temps.

Comme il a été vu au court du projet de semestre, les WLAN sont plus vulnérables que les réseaux câblés par le fait que la diffusion des ondes n'est que peu ou pas maîtrisable, et donc qu'un pirate peut facilement écouter le trafic, ou même se connecter au réseau. Il faut donc trouver des solutions afin de limiter les risques d'intrusion et protéger les données sensibles par des mécanismes d'encryption. C'est l'un des buts de ce projet. L'autre objectif correspond à l'étude des performances d'un réseau 802.11 sur lequel une plateforme de sécurité a été mise en œuvre. En effet, l'implémentation des mécanismes de sécurité contribue à la baisse des performances du réseau en question. Il va donc falloir comparer ces diminutions de débit entre différents protocoles d'encryption afin de déterminer lequel est le plus avantageux en matière de performance, mais également en matière de sécurité.

Mais avant tout, une démonstration des vulnérabilités du protocole actuellement proposé par la norme 802.11 va être réalisé, afin de justifier, d'une certaine manière, la raison d'être de ce travail de diplôme.

2 Vulnérabilités de WEP

Les réseaux 802.11 proposent un système d'encryption au niveau liaison appelé WEP (Wired Equivalency Protocol). Son but est de transmettre les données de manière sécurisée entre deux clients WLAN. Il a été démontré que l'algorithme RC4, sur lequel WEP se base, comportait des failles et rendait ainsi ce mécanisme insuffisant dès qu'il s'agit de protéger ses données de manière sûre. La première partie de ce travail de diplôme consiste à réaliser une attaque se basant sur les vulnérabilités de WEP.

2.1 La faille de RC4

RC4 est l'un des algorithmes à flux le plus utilisé dans les applications. Il a été conçu par Ron Rivest en 1987 et est resté secret jusqu'en 1994. L'attaque qui est décrite ici a été découverte par S.Fluher, I.Mantin et A.Shamir en 2001 [1] et concerne l'algorithme KSA (Key Scheduling Algorithm).

RC4 possède un tableau interne secret S qui est rempli par tous les $N = 2^n$ mots possibles de n bits, avec $n = 8$ dans la pratique donc un tableau de 256 positions. RC4 est composé de deux parties. Le KSA, qui sert à permuter le tableau initial (figure 2.2) à partir d'une clé K de longueur variable et l'algorithme Pseudo Random Generator Algorithm (PRGA) qui utilise ces permutations pour générer une séquence de bit pseudo aléatoire. La figure 2.1 représente les deux algorithmes qui manipulent le tableau.

KSA(K) Initialisation : For $i = 0..N-1$ $S[i] = i$ $j = 0$ Scrambling : For $i = 0..N-1$ $j = j + S[i] + K[i \bmod L]$ swap($S[i], S[j]$)	PRGA(K) Initialisation: $i = 0$ $j = 0$ Generator loop: $i = i + 1$ $j = j + S[i]$ Swap($S[i], S[j]$) Output $z = S[S[i] + S[j]]$
---	--

Fig. 2.1 Algorithme KSA et algorithme PRGA

Le KSA initialise S par les mots N dans un ordre croissant et j à 0 comme nous le montre la figure 2.2.

0	1	2	3	4	...	252	253	254	255
---	---	---	---	---	-----	-----	-----	-----	-----

Fig. 2.2 Tableau RC4 initial

Ensuite, dans l'opération de scrambling, KSA permute les valeurs du tableau qui sont pointées par les indices i et j en prenant soin, à chaque boucle, d'incrémenter i et de mettre à jour j en lui ajoutant $S[i]$ et le prochain mot de la clé K (dans un ordre cyclique, L étant le nombre de mots de K). La figure 2.3 montre le tableau S une fois KSA exécuté.

18	231	126	6	99	...	4	14	67	211
----	-----	-----	---	----	-----	---	----	----	-----

Fig. 2.3 Exemple d'un tableau RC4 permuté après KSA

Le PRGA initialise les deux indices i et j , puis boucle autour de 4 opérations triviales. Tout d'abord, i est incrémenté comme un compteur, ensuite j est incrémenté pseudo aléatoirement. La troisième opération permute les deux valeurs de S pointées par i et j . Finalement, la sortie (clé de flux) n'est rien d'autre que la valeur de S pointée par $S[i] + S[j]$. La figure 2.4 représente les étapes de RC4.

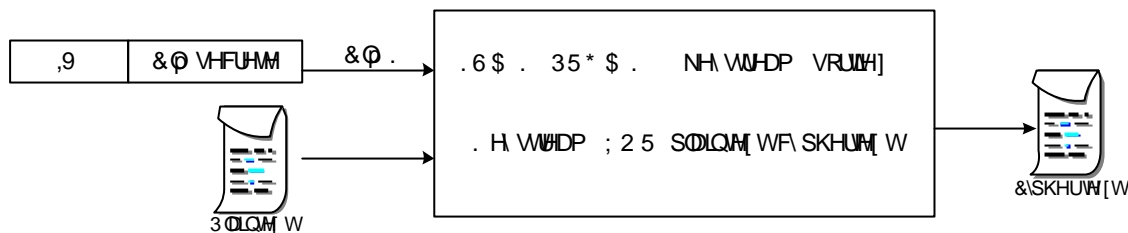


Fig. 2.4 Schéma de RC4

Comme déjà cité, RC4 n'est pas totalement fiable. Considérons que le pirate a accès à la valeur de tous les bits de certains mots de la clé. Nous faisons référence ici à l'Initialisation Vector (IV) qui est concaténé à la clé secrète pour créer la clé K . Si la même clé secrète est utilisée avec plusieurs IV différents et que le pirate peut obtenir le premier mot de la sortie de RC4 correspondant à chaque IV, il peut dès lors aisément recouvrer la clé secrète.

Le nombre d'IV requis dépend de l'ordre de la concaténation de celui-ci par rapport à la clé secrète, de sa taille et quelques fois de la valeur de la clé secrète. Cette attaque est donc particulièrement intéressante dans notre cas, car l'on peut facilement connaître le premier mot du payload (plaintext) pour chaque paquet comme nous le montre la figure 2.5, qui représente un paquet WEP, avec en bleu la partie cryptée.

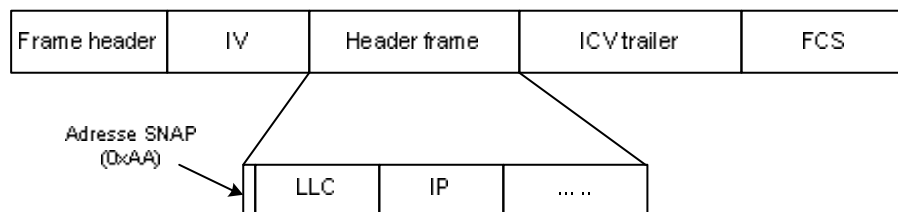


Fig. 2.5 Trame WEP et en-tête LLC

En effet, dans beaucoup de cas, le premier mot du payload correspond à l'adresse SNAP de l'en-tête LLC, qui n'est rien d'autre que 0xAA pour la majeure partie des protocoles. Donc, le pirate connaît le premier mot du plaintext (plaintext[0]) et le premier mot du cyphertext (cyphertext[0]). Il peut ainsi retrouver par un simple XOR le premier mot de la clé de flux (keystream[0]).

Le fait que nous ne nous intéressons qu'au premier mot de la clé de flux, simplifie le modèle de la sortie z . Le premier mot de la clé de flux ne dépend que de trois éléments spécifiques de permutation. La figure 2.6 nous montre l'état du tableau directement après KSA. Ceci est expliqué sans démonstration. Il est recommandé de lire le document [1] pour plus de détails, ou alors une deuxième lecture, « Using the Fluher, Mantin and Shamir Attack to Break WEP » [2], un peu plus vulgarisé pour les personnes qui n'aiment pas trop les mathématiques.

Lorsque trois mots sont disposés comme le montre le modèle ci-dessous, la valeur du premier mot de sortie (keystream[0]) vaut simplement z.

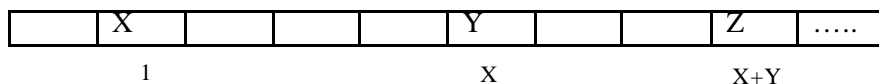


Fig. 2.6 Tableau RC4 après KSA

Si, lors de la création de la clé, à un moment donné $i = 1$, $X = S_i[1]$ et $X+Y = S_i[1] + S_i[S_i[1]]$, alors, avec une probabilité plus grande que 0.05, aucuns éléments référencés par ces trois valeurs ne participeront à aucunes permutations futures. Dans ce cas, la valeur $S[S[1] + S[S[1]]]$ sera le premier mot de la clé de flux (keystream[0], que le pirate connaît déjà). On considère cette situation comme condition résolue, et les paquets WEP qui se trouvent dans cette situation sont nommés paquets *intéressants*. A partir de là et grâce à la connaissance d'une partie des mots de la clé K, le pirate est à même de recouvrer la clé secrète. Tout se passe à la ligne $j = j+S[i]+K[i \bmod l]$ lors de l'opération de *scrambling*. Nous n'allons pas rentrer plus dans les détails, car l'explication remplit facilement quelques pages de développements mathématiques qui sortent du sujet de ce rapport.

2.2 IV collision

Il existe encore une autre attaque contre WEP qui mérite d'être citée. Elle se nome IV collision. Collision, pour signa ler qu'un IV est réutilisé plusieurs fois à plusieurs points de la transmission et que la clé secrète n'est que rarement changée. Le problème avec les algorithmes à flux est que deux paquets encryptés avec le même IV peuvent facilement être décryptés. Les équations suivantes prouvent ce fait. C correspond au cyphertext et P au plaintext, alors que v représente l'IV et k la clé secrète.

$$\begin{aligned} C_1 &= P_1 \otimes RC4(v, k) \\ C_2 &= P_2 \otimes RC4(v, k) \\ C_1 \otimes C_2 &= (P_1 \otimes RC4(v, k)) \otimes (P_2 \otimes RC4(v, k)) \\ C_1 \otimes C_2 &= P_1 \otimes P_2 \end{aligned}$$

Tout d'abord, 24 bits pour l'IV est trop peu, et entraîne la réutilisation régulière du même IV. Un AP qui transmet 1500 bytes à 11 Mbps épuise tous les IV en un peu moins de 5 heures. Deuxièmement, certaines cartes réinitialisent l'IV à 0 à chaque fois que la carte est initialisée et l'incrémente de 1 pour chaque paquet. Finalement, la clé secrète est très rarement changée, par le fait que cela implique de passer vers tous les postes clients et de réaliser la mise à jour manuellement. De plus, il faudrait le faire toutes les 5 heures. Le fait d'une clé de 128 bits (24 pour l'IV et 104 pour la clé secrète) ne fait que retarder l'échéance. Donc, une collision a toutes les chances de se produire de manière régulière.

Il existe peut-être d'autres attaques qui ne sont pas encore découvertes ou non répertoriées. Dans le cadre de ce diplôme, c'est la faille de l'algorithme KSA qui va être exploitée pour démontrer la non-fiabilité de WEP. Il est temps maintenant d'expliquer le déroulement complet de l'attaque.

2.3 Réalisation de l'attaque

Avant de commencer la pratique, il faut se poser certaines questions. Quelle plateforme doit être utilisée, quels sont les outils (logiciels) à ma disposition ou encore quel matériel est requis ?

Dans le cadre de cette attaque, le matériel requis n'est pas très conséquent, ce qui la rend réalisable par tous. En résumé, il faut un PC portable muni d'une carte wireless capable de capturer les paquets, c'est-à-dire une carte qui peut travailler en mode *monitor* (promiscuous mode). Ensuite, il faut un logiciel qui va capturer les trames et un petit utilitaire qui va craquer la clé secrète sur la base des paquets intéressants capturés. Une fois que le réseau ciblé est localisé, il ne reste plus qu'à attendre. Pour déterminer la plateforme, c'est le logiciel qui va craquer la clé qui dicte ce choix. L'outil qui va être utilisé ayant été développé pour Linux, il faut donc déployer l'attaque depuis une plateforme Linux.

2.3.1 Aircnort

Il s'agit de l'outil principal de cette attaque. Il réalise toutes les étapes du processus. C'est le groupe « The Shmoo Group » [3] qui a développé cet utilitaire. Dans un premier temps, Aircnort sniffe les paquets et détermine tous les BSSID d'où proviennent ces paquets. Déjà à ce niveau, un premier filtre élimine toutes les données non cryptées. Ensuite, les paquets qui se composent d'un IV intéressant sont sauvegardés et les autres effacés. A chaque fois que dix nouveaux paquets intéressants ont été trouvés, la clé d'encryption est recalculée. Une fois que suffisamment de paquets ont été récupérés, la clé d'encryption est déterminée en quelques secondes puis affichée. Aircnort n'est donc pas un jouet. Son unique but est de casser WEP. Bien qu'Aircnort ait été développé pour Linux, une version pour windows existe, mais n'est pas encore aboutie à l'heure de mon diplôme. Le site officiel de Aircnort [3] comporte tous les renseignements nécessaires.

2.3.2 Déroulement de l'attaque

Passons maintenant à la mise en œuvre à proprement parlé de cette attaque. Cette marche à suivre est relativement peu expliquée sur le site de Aircnort [3], donc voici toutes les différentes étapes de l'installation.

Description de la plateforme de test

Tout d'abord, il faut commencer par installer la plateforme, c'est-à-dire le réseau de test ainsi que l'ordinateur qui va sniffer et déterminer la clé à l'aide de Aircnort. Je parle d'un réseau de test, car il n'est pas question d'expérimenter cette attaque sur le réseau du voisin... J'ai fait le choix de travailler avec la norme 802.11b. Il est vrai que la norme 802.11g émerge, mais le débit n'est pas une priorité dans cette étape. La figure 2.7 représente la plateforme de test.

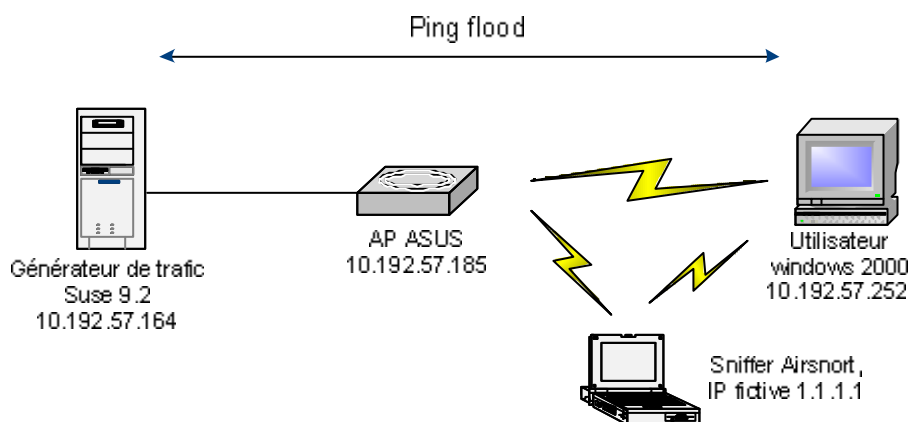


Fig. 2.7 Plateforme de test WEP

Dans mon cas, le réseau se compose d'une machine Linux qui génère le trafic en direction de l'utilisateur windows 2000. Les caractéristiques des machines ne sont pas importantes (sauf celles du sniffer que nous verrons plus bas). L'AP ASUS est un point d'accès traditionnel avec WEP 64 et 128 bits. Pour plus de détails sur les caractéristiques du matériel utilisé pour cette attaque, se référer au chapitre 11.1.

Comme l'on ne s'intéresse qu'au premier byte du payload, celui que nous connaissons (0xAA), la taille des paquets importe peu. C'est le nombre de paquets qui est important. Donc, le trafic généré se résume à la diffusion de *ping*. Il s'agit de *ping flood* plus exactement, pour réduire le temps de récupération des paquets intéressants. Le choix de Linux pour générer le ping flood est pratique, car il suffit de taper la commande `ping -f 10.192.57.252` depuis le générateur. Chaque 2 millisecondes environ, un *ping request* est envoyé suivi du *ping response* en retour. Cela fait un peu moins de 900 paquets par seconde en tenant compte d'une légère dérive, donc approximativement 3'240'000 par heure. Il est clair que sur un réseau ordinaire, cette moyenne de paquet est rare, et donc le temps de cassage se verrait très nettement allongé.

Installation du portable sniffer

L'ordinateur qui exécute Aircsnort doit posséder certaines caractéristiques. Dans mon cas, il s'agit d'un portable DELL inspiron 2650. Il ne faut pas forcément beaucoup de ressources pour cette machine. Ce portable possède un processeur cadencé à 1.6Ghz, muni de 256Mb de RAM. La taille du disque dur n'est pas volumineuse non plus, car seuls les paquets intéressants sont stockés. Le fait de devoir utiliser Linux oblige l'emploi d'une carte wireless possédant un chip particulier et pouvant être activée en mode *monitor* (promiscuous mode). Seuls deux types de chips sont supportés par Linux actuellement, il s'agit des chips *orinoco* et *prism2*. Il est donc vivement conseillé de s'assurer de ce point lié à la compatibilité avant d'investir dans du matériel.

Pour que le packaging PCMCIA puisse fonctionner, afin d'exploiter la carte, la distribution de Linux doit impérativement posséder un noyau égal ou supérieur à 2.4.x. Toutes les distributions actuelles possèdent un tel noyau. Dans mon cas, RedHat 9 est installé avec un noyau 2.4.20-8 et la version 3.2.3 du packaging PCMCIA. Ce dernier composant est directement installé en même temps que le système complet. Il n'y a donc, en principe, pas d'installation supplémentaire à ce niveau. Si le packaging PCMCIA est correctement installé, vous devriez, en tapant la commande `cardctl ident`, voir ceci s'afficher dans la console (cas d'une carte AVAYA SilverCard compatible orinoco) :

```
Socket 0 :  
  product info: «Avaya Communication»,«Avaya Wireless PC Card »,«Version 01.01»  
  manfid: 0x0156, 0x0002  
  function: 6 (network)
```

Si ce n'est pas le cas, commencez par vérifier que le service pcmcia a bien démarré et que les modules *orinoco_cs*, *orinoco*, *hermes*, *yenta_socket* et *pcmcia_core* sont bien chargés. Cela se fait à l'aide de la commande `lsmod`. Si vous n'y arrivez toujours pas, patience. Vérifiez que tout est bien installé ou recommencer la procédure. Il existe un site [4] entièrement consacré à ces problèmes, où vous y trouverez toutes les informations nécessaires, ainsi que les sources dans le cas où vous seriez amené à installer une version du packaging différente que celle fournie par défaut avec la distribution de Linux.

Une fois PCMCIA fonctionnel, il faut se consacrer aux pilotes de la carte wireless. Comme déjà cité, seules les cartes munies d'un chip *orinoco* ou *prism2* sont compatibles. Ces pilotes sont disponibles sur le site de Aircsnort [3], ainsi que les différentes mises à jour (*patch*). Dans mon

cas, la carte AVAYA SilverCard possède un chip orinoco. J'ai donc installé les pilotes orinoco-0.13e munis du patch orinoco-0.13e-patch.diff. Si vous possédez un chip prism2, les pilotes adéquats se trouvent également sur ce site, mais les mises à jour sont différentes ! En effet, beaucoup plus de bugs ont été découverts pour les cartes orinoco et ont dû être corrigés à l'aide de patches. Le site est très complet et documente bien les bugs et la résolution des problèmes qu'ils impliquent.

Pour installer ces drivers, il faut donc d'abord les mettre à jour. Pour ce faire, il faut suivre la procédure suivante. Commencez par décompresser les pilotes que vous avez téléchargé depuis le site de Airsnort (orinoco-13e.tar.gz pour ma part). En général, les fichiers sources se placent dans `/usr/src/`. Ensuite, il faut appliquer le patch. Pour ce faire, il faut d'abord copier orinoco-0.13e-patch.diff dans le répertoire orinoco-0.13e qui vient d'être créé lors de la décompression des pilotes, et taper la commande suivante directement depuis le répertoire orinoco-0.13e :

```
patch -p1 < orinoco-0.13e-patch.diff
```

Vous devriez voir la liste des fichiers mis à jour s'afficher dans la console:

```
patching file hermes.c
patching file hermes.h
patching file orinoco.c
patching file orinoco.h
```

Une fois mis à jour, il faut les compiler. Cela se fait très facilement si vous avez les sources du noyau installées dans `/usr/src/`. Il suffit de se placer dans le répertoire des pilotes et taper `make`.

Si tout s'est déroulé normalement sans messages d'erreurs, les fichiers objet (*.o) ont dû être créés. C'est-à-dire `hermes.o`, `orinoco.o`, `orinoco_cs.o`, `orinoco_pci.o`, `orinoco_plx.o` et `orinoco_tmd.o`. En réalité, seuls les fichiers `hermes.o`, `orinoco.o` et `orinoco_cs.o` nous intéressent. Il suffit maintenant de copier ces 3 fichiers dans le répertoire où doivent se trouver les pilotes, afin d'écraser les vieux. Cela peut varier en fonction de la distribution. Pour RedHat 9 avec le noyau 2.4.20-8, c'est le répertoire :

```
/lib/module/2.4.20-8/kernel/drivers/net/wireless/
```

Voilà, il suffit de redémarrer la machine ou simplement le service `pcmcia`. Une fois redémarré, pour vérifier que les pilotes sont bien mis à jour, vous pouvez taper la commande suivante (l'interface réseau associée à la carte est `eth1` dans ce cas) et voir apparaître ce résultat à la console :

```
prompt# iwpriv eth1
```

```
eth1 Available private ioctl :
    force_reset    (8BE0) : set 0      & get 0
    card_reset     (8BE1) : set 0      & get 0
    set_port3      (8BE2) : set 1 int  & get 0
    get_port3      (8BE3) : set 0      & get 1 int
    set_preamble   (8BE4) : set 1 int  & get 0
    get_preamble   (8BE5) : set 0      & get 1 int
    set_ibssport   (8BE6) : set 1 int  & get 0
    get_ibssport   (8BE7) : set 0      & get 1 int
    monitor        (8BE8) : set 2 int  & get 0
    dump_recs      (8BFF) : set 0      & get 0
```

Il est possible d'avoir quelques différences au niveau des fonctions possibles de votre carte. Mais ce qui est important d'avoir, c'est la fonction `monitor`, en gras. C'est Aircnort qui va employer cette fonction pour utiliser la carte à bon escient.

Il est temps d'installer l'essentiel. Vous pouvez vous procurer la dernière version de Aircnort sur le fameux site [3] que vous connaissez bien maintenant. Pour ma part, j'ai installé la version 0.2.2b. Une fois le paquetage (`aircnort-0.2.2b.tar.gz`) téléchargé et décompressé comme à l'habitude dans `/usr/src/`, allez dans le répertoire `aircnort-0.2.2b` qui vient de se créer et tapez la commande suivante :

```
./autogen.sh
```

Ce petit script (équivalant à `./configure`) permet de contrôler la présence des bibliothèques nécessaires au fonctionnement de Aircnort et notamment des bibliothèques graphiques, car Aircnort est une application GUI. Il est donc important que la version 1.2.0 (ou plus) de GTK (Gimp Tool Kit) soit présente dans le système, et toutes ses dépendances. Il existe un site [5] spécialement consacré à cette bibliothèque, qui permet de contrôler et de télécharger les fichiers nécessaires. En général, GTK est présent si vous possédez le gestionnaire de bureau GNOME. Si tout se déroule comme il faut, le résultat du script vous invite à compiler le paquetage à l'aide de la commande `make`, puis de l'installer avec un `make install`.

Ajoutons encore que pour que l'adaptateur wireless soit reconnu comme interface réseau (`eth1`), il est nécessaire qu'une adresse IP lui soit attribuée. Il faut donc taper la commande suivante.

```
ifconfig eth1 1.1.1.1
```

De cette manière, Aircnort pourra afficher l'adaptateur dans sa liste des interfaces pouvant être utilisées. On remarque que l'adresse IP est totalement fictive, vous êtes libre de la choisir.

Voilà, c'est fait. Vous pouvez lancer Aircnort à l'aide de la commande `aircnort` et voir apparaître la fenêtre représentée par la figure 2.8.

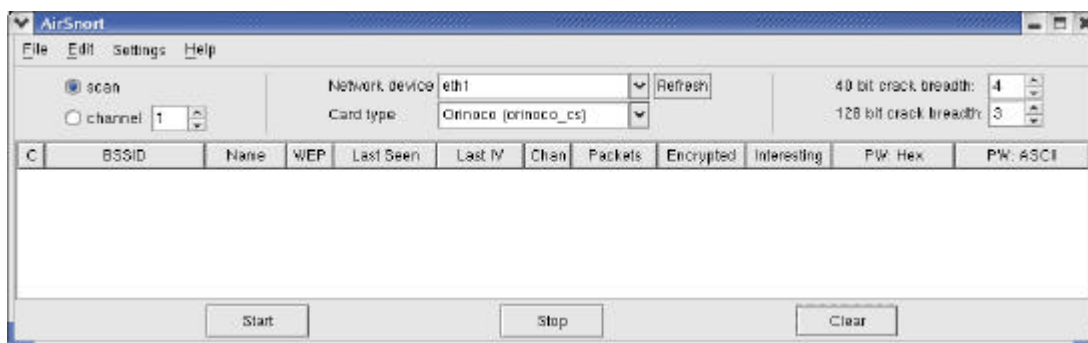


Fig. 2.8 GUI de Aircnort

Utilisation d'Aircnort

Ce logiciel est relativement simple d'utilisation. Une fois que vous avez sélectionné l'interface que vous voulez utiliser et le type de carte dont il s'agit, vous pouvez cliquer sur `start`. C'est là que Aircnort essaie de passer la carte en mode `monitor`. A ce stade, si le message représenté par la figure 2.9 apparaît, c'est que les pilotes ne sont pas correctement mis à jour, et qu'il faut revenir quelque peu en arrière.

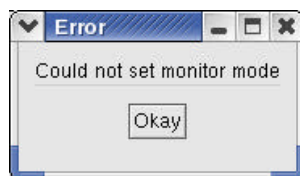


Fig. 2.9 Message d'erreur: "Could not set monitor mode"

Si aucune erreur ne se produit, Aircnort commence à scanner les différents canaux. Si vous connaissez le canal du BSSID que vous voulez craquer, vous pouvez directement le spécifier à l'aide des boutons radio à gauche. Sinon, attendez que Aircnort ait affiché tous les BSSID, ainsi que leur canal respectif et sélectionnez le bon après. La figure 2.10 donne une représentation de Aircnort lorsqu'il a découvert plusieurs BSSID.

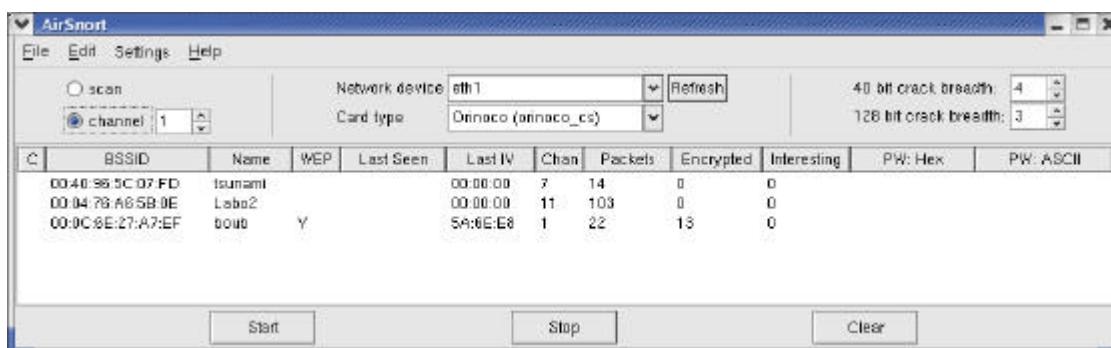


Fig. 2.10 Recherche de BSSID

On remarque ici que seul le BSSID *boub* est crypté, grâce au Y qui se trouve dans la colonne WEP. Aircnort affiche le dernier IV rencontré lorsqu'il s'agit d'un BSSID crypté. La colonne *Packets* indique le nombre de paquets capturés, alors que la colonne *Encrypted* indique le nombre de paquets cryptés capturés. Jusqu'ici tout va bien.

Maintenant, Aircnort est en mesure de récupérer les fameux paquets intéressants. Le nombre de ces trames ayant été trouvées s'affiche dans la colonne *Interesting*. Tous les 10 nouveaux paquets intéressants, Aircnort essaie de calculer la clé. Lorsqu'il l'a trouvée, elle est affichée sous le format hexadécimal et parfois sous le format ASCII. La figure 2.11 montre l'état de Aircnort lorsqu'il a réussi à déterminer une clé.

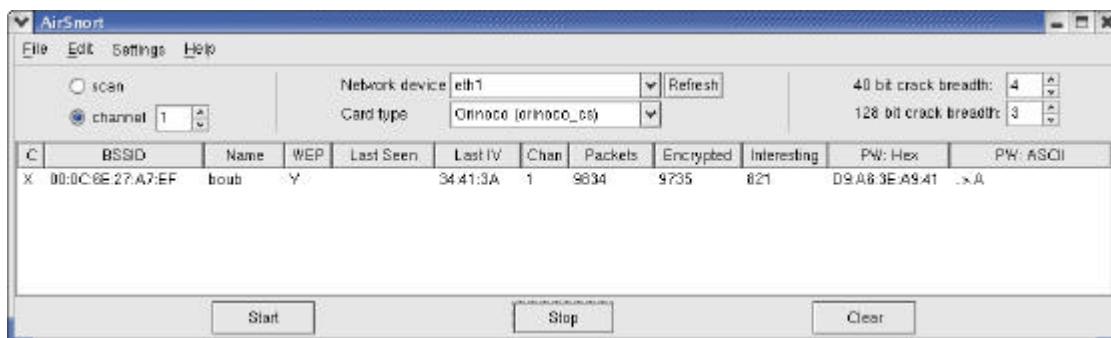


Fig. 2.11 Cassage réussi

On remarque la croix dans la première colonne C, qui indique que la clé a été trouvée. On peut bel et bien voir cette clé dans la colonne PW:Hex. On remarque que la colonne PW:ASCII ne sert pas à grand chose dans cette situation, mais peut s'avérer utile si la clé d'encryption a été enregistrée sous la forme de caractère ASCII.

Notons que dans ce cas seulement 821 paquets intéressants ont suffi pour déterminer la clé. Cela est relativement peu comme nous le verrons lors de la présentation des résultats. De plus, le nombre de paquets cryptés capturés n'est pas significatif ici, car il a été réinitialisé au milieu de la mesure. Il faudrait plutôt en considérer 2 ou 3 millions à la place. Mais nous verrons également cela dans le chapitre 2.3.4.

Airsnort utilise une attaque qui se base sur les probabilités. Avec un nombre de paquets limités et assez de puissance de processeur, il est possible d'effectuer des recherches plus exhaustives. La recherche d'une clé implique un parcours en profondeur d'un arbre naïf. La profondeur de l'arbre est de 5 pour une clé de 40 bits et de 13 pour 128 bits. La largeur de l'arbre est régulée par les deux champs en haut à droite de la fenêtre. Ces paramètres indiquent à Airsnort d'essayer les n meilleurs valeurs pour chaque position de la clé dans l'arbre, en utilisant une dérivation statique des IV qui ont pu être collectés. La valeur de ce paramètre peut influencer énormément sur le temps de passage. Par défaut, au début, il est conseillé de 4 pour 40 bits et 3 pour 128 bits. Si beaucoup d'IV ont été collectés, >1500 pour 40 bits et >3000 pour 128 bits, il est possible d'augmenter petit à petit cette largeur.

Nous y sommes, il ne reste plus qu'à attendre... Pendant ce temps, parlons un peu des problèmes rencontrés lors de la mise en œuvre de cette plateforme et surtout du sniffer.

2.3.3 Problèmes d'installation

Ayant passé un peu moins de deux semaines à installer la machine qui exécuterait Airsnort, je me devais d'écrire un chapitre sur tous ces problèmes et mon apprentissage de Linux. Le choix de cette plateforme est incontournable pour plusieurs raisons. Premièrement, Linux est un OS utile dans pratiquement toutes les situations. Deuxièmement il s'agit d'un OS open source et par conséquent gratuit. Ce deuxième fait implique que beaucoup de personnes et de groupes de développement indépendants conçoivent des utilitaires, des petits programmes ou même des logiciels performants qui s'avèrent être très pratiques que ce soit dans le domaine des télécommunications, de la programmation, de l'informatique ou du domaine scientifique. Le fait d'être open source entraîne également la création de plusieurs distributions. On peut citer RedHat, Debian, Slackware ou encore Suse. L'une ou l'autre de ces distributions peut s'avérer plus ou moins adaptée dans une situation particulière. Ce choix provient également de l'opinion et des expériences de chacun ainsi peut-être que d'un petit coup de cœur. Il est vrai que Linux entraîne quelques problèmes liés aux matériels et à sa compatibilité. Une bonne connaissance de cette plateforme est vivement recommandée pour espérer pouvoir résoudre ces problèmes. Dans mon cas, je ne disposais pas de ces connaissances avant les deux semaines qui m'ont fallu pour monter la machine qui avait pour rôle de sniffer les paquets et de casser la clé d'encryption. Comparé à la convivialité de windows, Linux peut paraître un peu froid..., du moins au début.

J'ai dû faire face à un ordinateur portable d'origine et de marque inconnue. Pour pouvoir exploiter la carte, il faut posséder un noyau égal à 2.4.x ou plus et le paquetage PCMCIA. Il a fallu tout d'abord trouver une distribution possédant un tel noyau et compatible avec le portable. Pour des raisons inconnues, ni RedHat, ni Debian n'étaient supportées et c'est finalement Suse qui a été choisie après déjà plusieurs jours d'acharnement. Une fois que la distribution a commencé à tourner correctement, avec le support PCMCIA, il a fallu mettre à jour les pilotes. Ceci a posé plus de problèmes. N'ayant pas les bonnes sources du noyau, ni les bonnes connaissances de ce sujet, il m'a fallu plusieurs jours pour comprendre comment les compiler.

Bref, pleins d'erreurs impossibles à décrire et de longues heures de recherche avant de demander un peu d'aide auprès des assistants.

Finalement, le gros problème se situait au niveau du mode monitor de la carte. Lors de la commande `iwpriv`, le mode était bien disponible, mais impossible de paramétrer la carte. Le message d'erreur de la figure 2.9 n'arrêtait pas d'apparaître. Après encore plusieurs jours de recherche et de bidouillage, il a officiellement été conclu que cette machine était incompatible avec Linux, et après un peu moins de deux semaines, un échange de portable m'a été proposé, le DELL inspiron 2650 ce qui a miraculeusement résolu tous les problèmes... ou presque. Avec ce nouveau portable, Aircnort arrive à mettre la carte en mode monitor et à capturer quelques paquets intéressants. Mais c'est tout, après ces quelques trames, plus rien. Il faut savoir qu'un petit utilitaire est fourni avec Aircnort. Il se nomme `orinoco_hopper`. Il permet de gérer les canaux à écouter et les modes de la carte au même titre que `iwpriv`. Il est dit dans la pratique d'exécuter ce script avant de lancer Aircnort. Malheureusement, dans mon cas, il ne fallait pas. Donc, pendant plusieurs jours, des recherches ont été entreprises pour trouver où pouvait bien se trouver le bug qui stoppait la capture des paquets intéressants après une dizaine. J'ai changé le firmware, la carte, la distribution de Linux, mais rien de tout cela n'a fonctionné. Ayant assez perdu de temps, j'ai passé à la suite du projet. Pendant la 9^{ème} semaine, un collègue qui avait déjà fait cette expérience, me signale que `orinoco_hopper` provoque un conflit avec la fonction `iwpriv` que Aircnort emploie pour paramétrer la carte. A partir de là, tout a bien fonctionné et j'ai pu commencer mes mesures.

Avec un peu de recul, ces deux semaines n'ont pas été totalement perdues. Ces quelques jours d'acharnement sous Linux m'auront permis de bien comprendre la philosophie de cet OS, ses particularités, son fonctionnement et ces subtilités. Plusieurs commandes qui m'étaient inconnues, me sont familières aujourd'hui.

2.3.4 Résultats

Donc c'est uniquement depuis la 9^{ème} semaine que les testes ont pu être réalisés. J'ai décidé d'effectuer plusieurs séries de mesures avec une clé de 64 bits, et uniquement deux pour une clé de 128 bits car cela prend plus de temps. Différents types de clés ont été testés. Des clés répétitives et des clés quelconques. Il faut dire que la clé croissante est considérée comme une clé quelconque du point de vue de l'algorithme de cassage. La figure 2.12 illustre le tableau des résultats obtenus pour des clés de 64 bits :

N° mesure	Clé Hex.	Particularité	Temps	Nbr de paquet crypté capturé	Nbr de paquet intéressant
1	AAAAAAAAAA	Répétitive	1h10	3'915'141	837
2	5555555555	Répétitive	1h05	2'542'369	1023
3	1234567890	Croissante (quelconque)	4h30	12'256'780	2480
4	D9A63EA941	Quelconque	1h17	4'437'342	1176
5	D9A63EA941	Quelconque	1h22	4'306'296	1163
6	D9A63EA941	Quelconque	0h52	3'090'525	821
7	BD1D309111	Quelconque	3h50	14'149'328	2989

Fig. 2.12 Résultats pour les clés de 64 bits

De manière générale, j'ai été surpris par la rapidité de cassage. Il faut préciser tout de même que les ping généraient énormément de paquets (900 par seconde), alors que dans un cas réel, un trafic plus faible circulerait (plutôt de l'ordre de 200 par seconde), donc plus de temps pour récolter les trames intéressantes. Tout d'abord, il est à signaler que ce n'est pas forcément le fait

d'avoir une clé simple (répétitive) qui réduit le temps de cassage. On remarque qu'en moyenne, dans le cadre de ces mesures, il faut envisager récolter entre 3 à 4 millions de paquets cryptés pour espérer récupérer entre 1000 et 3000 trames intéressantes. Le fait d'avoir casser trois fois la même clé (mesure 4, 5 et 6) n'améliore pas le temps de cassage. On remarque, au contraire, que le temps reste dans le même ordre de grandeur. On peut noter finalement que certaines clés peuvent entraîner la récupération de beaucoup plus de paquets intéressants, presque 3'000 pour la mesure 7, alors que seulement 821 ont suffi pour la mesure 6 par exemple. La conclusion à tirer de cela est que le nombre de paquets intéressants qu'il faut pour casser une clé ne dépend pas du type de clé.

Pour les clés de 128 bits, c'est un autre pair de manche. En effet, le temps n'est pas simplement doublé, mais le nombre de paquets intéressants est supérieur à 3'000. La figure 2.13 représente les résultats.

N° mesure	Clé Hex	Particularité	Temps	Nbr de paquet crypté capturé	Nbr de paquet intéressant
1	AAAAAAAAAA.....AAAAAA	Répétitive	40 min	1'905'762	3'235
2	0123456789ABCEDF0123456789	Quelconque	15 h	40'548'124	3'015

Fig. 2.13 Résultats pour les clés de 128 bits

Dans le cas de la clé répétitive, le temps est très court, plus court que pour la clé de 64 bits la plus rapidement trouvée, mais que beaucoup plus de paquets intéressants sont exigés. Dans la première mesure, sur 2 millions de paquets capturés, il y a déjà 3'235 trames intéressantes. On peut donc affirmer que la fréquence des paquets intéressants dépend de la clé. Par contre, pour la clé quelconque il a fallu énormément plus de temps pour arriver à déterminer la clé. Cette mesure a pris plusieurs jours par le fait que l'AP était victime d'un déni de service car il recevait trop de ping. Par contre, le même nombre de paquets intéressants était requis.

Le temps de cassage est déterminé par la quantité de trafic, et la fréquence des paquets intéressants, qui, comme on vient de le voir, dépend de la forme de la clé. Donc, le facteur chance est à considérer pour disposer d'une clé qui permet une bonne fréquence de diffusion de paquets intéressants.

Comme conclusion à cette première partie du diplôme, je dirais que le problème principal de WEP est la gestion des clés d'encryption. Le fait qu'elles soient trop peu changées, entraîne la répétition du même IV pour la même clé, ce qui est un grave problème et qui permet de nombreuses attaques. Essayons donc de trouver des solutions à ces problèmes.

3 Première approche

Il est temps de rentrer dans le vif du sujet. Les considérations par rapport aux pirates et aux attaques sont les mêmes partout. La norme 802.11 n'est rien d'autre qu'une adaptation des réseaux Ethernet dans les airs, et le même genre d'attaques que les réseaux câblés y sont réalisables. Seulement que dans le cas des WLAN, c'est beaucoup plus facile...

3.1 Déterminer ses besoins

Il faut commencer par bien définir les services, les applications et les données qui doivent être protégées, afin de déterminer quel schéma de sécurité doit être implémenté. Il faut se poser certaines questions. Est-ce qu'une authentification forte est de rigueur ou bien un simple mot de passe peut-il faire l'affaire ? Faut-il restreindre l'accès à certains services selon le type d'utilisateur ? Faut-il crypter une seule application particulière ou l'ensemble du trafic ? C'est toutes les réponses à ces questions qui déterminent la plateforme de sécurité à implémenter pour répondre à une meilleure efficacité. Evidemment d'autres facteurs entrent en jeu, comme les performances, le coût.

Dans le cadre du travail de diplôme, le but est de réaliser une plateforme sécurisée pouvant répondre aux problèmes qu'apportent les réseaux sans fil et ceux d'une application transportant de la voix ou de la vidéo.

3.2 Problèmes liés aux WLAN

Une difficulté qu'ajoute de tels réseaux est que tout le monde situé à proximité d'un point d'accès est susceptible de s'y connecter pour en tirer profit. Ceci oblige à réaliser une authentification forte de l'utilisateur et des machines (hosts). Tous les services que propose le réseau doivent être temporairement bloqués temps que l'authentification n'a pas été réalisée avec succès. Une police d'accès à diverses applications doit être mise en œuvre afin que certaines applications ne soient accessibles que pour un certain type d'utilisateurs par exemple. Ce schéma est indispensable dans le cas des réseaux sans fil.

Le fait de la libre circulation des données dans l'espace implique également un cryptage des transactions beaucoup plus strictes. De nombreux algorithmes existent et ont été développés dans l'optique des réseaux câblés (RSA, DES, RC4...). C'est les mêmes qui sont employés par les applications qui reposent sur un WLAN. Au niveau des protocoles, là aussi la plupart des solutions actuelles proviennent des réseaux filaires (IPsec, SSL,...). Le choix de telle ou telle solution sera déterminé suivant la volonté de crypter les paquets d'une application bien précise ou bien de tout le trafic. Dans la plateforme du diplôme, seules les applications de voix et de vidéo doivent être chiffrées.

Un dernier problème, mais qui n'est pas forcément lié aux réseaux sans fil, est la gestion des clés. Il a été vu avec WEP par exemple, que les attaques se basent essentiellement sur le fait que la clé secrète n'est que trop rarement changée. Le problème de ce protocole est qu'il n'y a pas du tout de gestion des clés. C'est l'administrateur qui doit mettre à jour la clé manuellement sur chaque machine. C'est peut-être un moyen sûr, mais qui devient vite impossible compte tenu du nombre d'utilisateurs qui ne cesse de croître. Donc, un protocole de gestion des clés est indispensable pour des questions de maintenance et surtout de sécurité.

3.3 Problèmes liés à la voie et à la vidéo

Cela semble clair, il s'agit du problème de la qualité de service. Il est évident que les performances d'un réseau, quel qu'il soit, sont réduites lors de l'implémentation d'un schéma de sécurité. Il est donc important de tenir compte de ce problème lors du choix de la solution qui va être mise en œuvre. Il paraît évident que l'on ne peut pas implémenter de la QoS sur un réseau dont les performances nuisent au bon déroulement d'une application, surtout pour la voix ou la vidéo. Donc le problème que pose la QoS est indirectement lié à la mise en œuvre d'une plateforme de sécurité.

Les chapitres suivants décrivent les différentes normes et protocoles qui existent actuellement. Chacune de ces solutions représente un élément indépendant qui possède un rôle bien précis dans l'architecture de la plateforme (Authentification, encryption ou gestion des clés). C'est en combinant plusieurs éléments que l'on obtient un schéma complet.

4 RADIUS

Pour débiter, il va être question d'authentification. Il vient d'être mentionné que n'importe qui peut se connecter à votre réseau. C'est la raison pour laquelle une vérification de l'utilisateur est indispensable, mais aussi que l'utilisateur puisse vérifier qu'il se connecte au bon réseau et non à un AP qu'un pirate aurait déposé pour détourner le trafic.

La première pièce de notre schéma est un serveur d'authentification. Son rôle est de stocker les données servant à l'authentification de chaque utilisateur (mots de passe, clés, certificats...). De plus, ce serveur enregistre un acompte pour chaque utilisateur. Cet acompte sauvegarde des paramètres sur l'état de la session en cours, comme la durée de la connexion par exemple. Une base de données est maintenue pour stocker toutes les informations relatives aux utilisateurs. On parle alors de serveur AAA, pour Authentification, Autorisation et Acompte.

RADIUS, qui signifie Remote Authentication Dial In User Service, est un protocole d'authentification de type client/serveur. Il a été créé par Lucent Remote Access et est défini par les RFC 2865 et 2866 [6]. Il a d'abord été conçu pour les connexions PPP. Il permet l'échange de messages entre le serveur AAA et un NAS (Network Access Service), pour réaliser le processus d'authentification. Dans notre cas, c'est l'AP qui joue le rôle du NAS (qui est nommé client dans ce texte). Le serveur d'authentification s'appellera serveur RADIUS. L'implémentation de ce serveur a été développée pour une plateforme UNIX, mais une version gratuite existe pour Linux, elle se nomme freeRADIUS.

RADIUS apporte certains avantages. Au lieu d'être dispersées un peu partout sur plusieurs machines de votre réseau, toutes les informations relatives aux utilisateurs sont stockées sur une seule machine, apportant ainsi de la clarté et minimisant déjà certains risques. Très flexible, ce protocole peut être utilisé avec plusieurs types de serveur de communication qui le supporte. Donc c'est RADIUS qui s'adapte à votre réseau, et non l'inverse. De plus, une base de données est relativement simple à tenir, il est facile d'y accéder et de mettre à jour les données. Une petite interface graphique rend la chose encore plus agréable. Finalement, un serveur RADIUS peut également jouer le rôle de proxy dans le cas où plusieurs serveurs sont disposés. Il transmettra alors les requêtes vers un autre serveur et redirigera également les réponses destinées au client.

4.1 Les messages

Notons tout d'abord que le protocole RADIUS est employé entre un NAS et un serveur RADIUS, et pas directement entre l'utilisateur et le serveur. Ce choix devient logique lorsque l'on prend en considération le rôle de l'autorisation. En effet, c'est au niveau de l'AP que l'on peut bloquer l'accès à certains services et applications. Nous verrons cela plus en détails dans le chapitre suivant. Dans le cas d'un réseau câblé, c'est un switch par exemple qui jouerait le rôle du NAS. Lorsqu'un client (AP) est configuré pour utiliser RADIUS, chaque utilisateur qui désire se connecter à celui-ci doit lui transmettre ses données d'authentification. Ensuite, le client envoie ces données vers un serveur RADIUS. Il s'agit du paquet *Access Request*, qui sert à réaliser une demande d'authentification. On trouve dans ce paquet certains attributs comme le nom d'utilisateur, son mot de passe ou encore l'ID du client par lequel il veut s'authentifier. À la réception de ce message, le serveur vérifie tout d'abord le secret partagé avec le client. Il s'agit d'une chaîne de caractère connue uniquement du client et du serveur et qui a été échangée de manière sur (indépendamment du réseau, comme une disquette par exemple). Si ce critère n'est pas satisfait, la requête ne sera pas traitée. Si le client est valide, le serveur peut contrôler l'existence de l'utilisateur dans sa base de données et vérifier son mot de passe. D'autres critères peuvent être ajoutés pour satisfaire l'authentification, comme l'IP du client ou encore le port qui

est alloué à l'utilisateur. On reviendra sur ces critères dans le chapitre suivant. Si un des critères n'est pas valide, le serveur renvoi au client un *Access Reject* pour indiquer le refus de l'utilisateur sur le réseau. Au contraire, si tous les critères sont confirmés, la réponse est retournée sous la forme d'un challenge auquel l'utilisateur doit participer. Il s'agit du paquet *Access Challenge*. Dans un mode d'authentification challenge/réponse, l'utilisateur reçoit un nombre pseudo-aléatoire qu'il doit encrypter grâce à une connaissance commune entre le serveur d'authentification et lui-même. Cette connaissance peut-être de plusieurs formes, comme une SmartCard ou un système de clé publique gérée par certificats. Le NAS transmettra alors ce challenge à l'utilisateur. Ce dernier pourra calculer la réponse et la renvoyer au client qui fera suivre la réponse au serveur RADIUS par l'intermédiaire du paquet *Access Response*. Le serveur contrôle alors la validité de la réponse et acceptera l'utilisateur dans le cas où la réponse est correcte. C'est le paquet *Access Accept* qui permet cette confirmation au client. Sinon, C'est encore un paquet *Access Reject* qui indiquera au client que l'utilisateur ne peut pas avoir accès au réseau. Il est possible d'avoir plusieurs challenges échangés entre l'utilisateur et le serveur lorsque la procédure d'authentification est un peu plus complexe que le simple échange de mots de passe. La figure 4.1 décrit le flux de messages échangés pendant une phase d'authentification réussie entre un client et un serveur RADIUS.

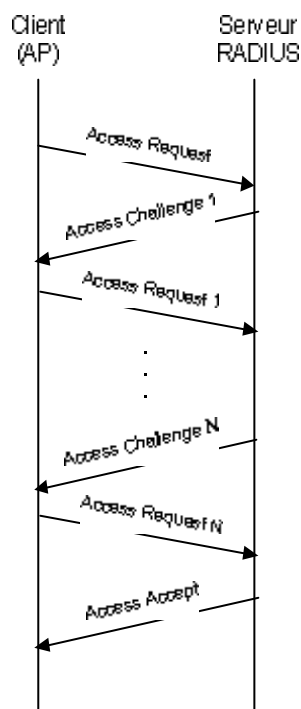


Fig. 4.1 Echange des messages RADIUS

Dès ce moment, l'utilisateur est en mesure d'accéder aux ressources et services que le réseau propose. Le protocole RADIUS est transporté par UDP et utilise le port 1812 en général. Il est possible que certaines versions précédentes de RADIUS emploient le port 1645. La figure 4.2 nous montre l'en-tête d'un paquet RADIUS.

bytes	1	1	2	16	0...4076
	Code	Identifier	Length	Authenticator	Attributes

4.2 Critères d'authentification

Les attributs RADIUS retiennent les détails de l'authentification, de l'autorisation et de l'information. C'est par l'intermédiaire de ces attributs que le client et le serveur peuvent s'échanger les données d'authentification.

Le format de transmission des différents attributs dans le champ prévu à cet effet est de la forme suivante, représenté par la figure 4.3.

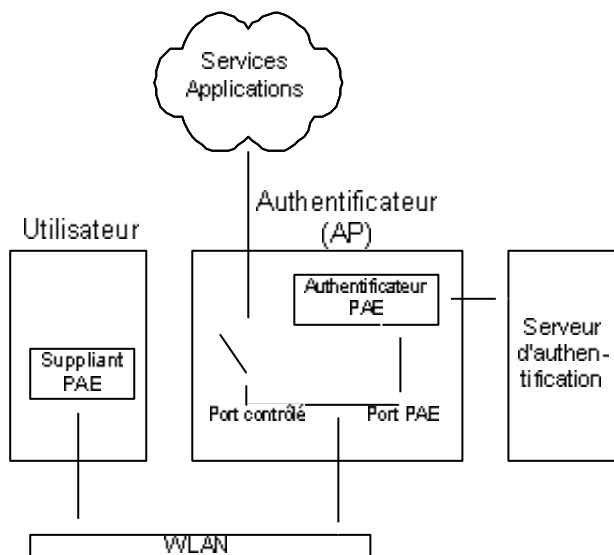
bytes	1	1	0..253
	Type	Length	Value

5 802.1x

802.1x est le dialogue d'authentification entre le système ayant besoin d'accéder au réseau (utilisateur) et le réseau lui-même (AP). Ce dialogue utilise EAP (Extensible Authentication Protocol). Initialement, lorsqu'un utilisateur veut se connecter à un réseau, tous les ports d'application lui sont fermés. Il doit d'abord procéder à l'authentification. Evidemment, on ne peut pas s'authentifier auprès d'un serveur si aucun port de l'AP n'est ouvert, car RADIUS fonctionne sur UDP. Donc, l'utilisateur et l'AP doivent employer un protocole qui fonctionne juste au-dessus de la couche liaison. De plus, ce protocole doit être capable de comprendre plusieurs moyens d'authentification (mots de passe, certificats, SmartCard...) afin de répondre à toutes les demandes. Le but est que le protocole d'authentification soit indépendant du type de serveur AAA et de l'AP intermédiaire. Donc, l'AP doit pouvoir gérer l'accès au réseau et permettre de relayer plusieurs modes d'authentification en direction du serveur AAA. C'est la norme 802.1x qui s'occupe de l'accès au réseau par la méthode Port-Based Network Access Control, et c'est le protocole EAP qui transporte les données d'authentification entre l'utilisateur et l'AP. Souvenons-nous que entre l'AP et le serveur RADIUS, c'est le protocole RADIUS qui est employé et ce sont les attributs RADIUS qui transportent les données d'authentification véhiculées par EAP. Il y a d'ailleurs des attributs bien spécifiques à certaines méthodes d'authentification. Donc, l'AP doit jouer le rôle de passerelle entre EAP et RADIUS et gérer l'accès au réseau.

5.1 Port-based Network Access Control

802.1x définit 2 ports (liaisons) logiques d'accès entre le suppliant (l'utilisateur) et l'authentificateur (AP), qu'il ne faut pas confondre avec les ports d'application. La figure 5.1 montre le schéma de ces deux ports logiques.



Un de ces deux ports, le Port Access Entity (PAE), permet l'échange des trames EAP entre l'utilisateur et l'authentificateur. L'autre port, le port contrôlé, permet l'échange des trames (trafic normal) entre l'utilisateur et le LAN une fois seulement l'authentification réussie. Il faut donc considérer que l'authentificateur bloque tout le trafic tant que l'authentification n'a pas été réussie, à l'exception des trames qui échangent les données d'authentification. Cela implique un point d'accès spécifiquement prévu à cet effet et qui puisse être «commandé » par le serveur d'authentification. Pour les détails de cette norme, se référer au standard IEEE 802.1x [7].

Maintenant que le principe de blocage des utilisateurs par l'AP a été décrit, parlons un peu du dialogue que partagent ces deux intervenants afin d'autoriser l'accès au réseau.

6 EAP

802.1x repose en partie sur EAP (Extensible Authentication Protocol). C'est le RFC 2284 [8] qui définit ce protocole qui a initialement été développé pour PPP. EAP fonctionne sur la couche liaison comme le montre la figure 5.1. L'on voit également que le moyen d'authentification est totalement indépendant du réseau sur lequel il est appliqué.

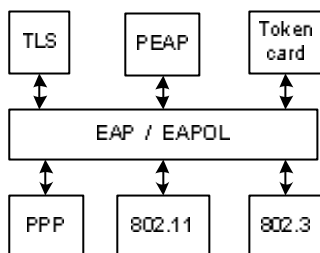


Fig. 5.1 EAP

Cela permet une très grande flexibilité et permet une grande hétérogénéité au niveau des choix relatifs aux principes d'authentification que vous voulez implémenter. Car il faut savoir que toutes ces méthodes d'authentification possèdent leurs avantages et inconvénients et sont plus ou moins sûres. Une méthode peut s'avérer plus adaptée dans certains cas qu'une autre. C'est un choix très important qu'il faut faire. Peut-être que pour une certaine gamme d'utilisateurs un simple échange de mots de passe peut suffire, alors que pour d'autres, une authentification par certificats doit impérativement être mise en place.

Notons encore qu'une version de EAP, appelée EAPOL (EAP over LAN) a été développée pour permettre le transport de EAP sur différents types de LAN, comme Token Ring par exemple.

Attardons-nous maintenant sur ces différentes méthodes d'authentification. Il y a deux familles : l'une basée sur la gestion de certificats et l'autre sur des mots de passe.

6.1 Authentification basée sur certificats

L'avantage des certificats est une meilleure sécurité, mais implique leur gestion, chose relativement complexe à installer et à gérer. En effet, il faut s'abonner auprès d'une autorité compétente de certification, faire la demande de plusieurs certificats et les installer dans toutes les machines. De plus, il faut sans cesse se référer à la liste de révocation pour s'assurer de la validité des certificats.

6.1.1 EAP-TLS

Transport Layer Security est le standard qui succède à SSL. Le RFC 2716 [8] décrit l'utilisation de TLS pour l'authentification. Cette méthode se base sur les certificats X.509. L'utilisateur doit posséder un certificat que le serveur peut valider et l'utilisateur doit également pouvoir valider celui du serveur d'authentification. On parle alors d'une authentification mutuelle. Ceci est important dans le cadre des réseaux sans fil, car un utilisateur doit être sûr qu'il se connecte bien au réseau auquel il croit et non à un AP qu'un pirate aurait déposé pour détourner le trafic. TLS apporte une solution pour protéger les données d'authentification entre l'utilisateur et l'AP, mais également pour s'échanger les clés de session de manière sûre, ce qui limite certains défauts de

WEP. Etant une solution relativement complexe, elle ne devient intéressante que pour de large entreprise, ou pour des réseaux et applications critiques.

6.1.2 PEAP

Ce principe d'authentification (Protected EAP) défini par IETF [10] diminue l'inconvénient de la gestion des certificats qu'implique EAP-TLS et simplifie donc quelque peu le schéma de sécurité. On parle toujours d'authentification mutuelle. La première étape consiste à authentifier le serveur auprès de l'utilisateur. Donc, le serveur doit tout de même posséder un certificat que l'utilisateur doit être en mesure de valider. Une fois cela réalisé, n'importe quelle autre méthode EAP peut-être employée pour authentifier l'utilisateur auprès du serveur.

La version Microsoft de PEAP ne permet que le protocole MS-CHAP pour authentifier l'utilisateur auprès du serveur. Ceci simplifie la base de données, qui ne contiendra que les utilisateurs supportant MS-CHAP version 2, comme les domaines Windows NT et Active Directory. Il existe également une version CISCO de PEAP, qui supporte l'authentification du client par OTP (One Time Password) et par login. Ceci permet le support pour une base de données OTP d'un vendeur comme RSA Security and Secure Computing Corporation, mais également des bases comme LDAP, Novell NDS ou Microsoft.

On voit donc que PEAP est une solution à cheval entre la méthode par certificats et celle par mots de passe.

6.1.3 EAP-TTLS

Tunneled TLS fonctionne sur le même principe que PEAP, mais plutôt pour une liaison PPP, car il supporte CHAP, MS-CHAP, MD5 comme moyen d'authentification du client vers le serveur. Il est également défini par IETF [11] et se base sur un premier tunnel TLS réalisé de la même manière que PEAP, c'est-à-dire avec certificats.

6.2 Authentification basée sur mots de passe

Ce principe d'authentification retire le problème lié aux certificats. Par contre, cette méthode se trouve être moins sécurisée et sujette à un plus grand nombre d'attaques. Notamment, une attaque basée sur des dictionnaires de mots de passe, que le pirate emploie pour tester de nombreuses possibilités. Ces dictionnaires contiennent des mots de tous les jours ou des surnoms. C'est pourquoi il est conseillé d'utiliser des mots de passe composés de caractères aléatoires comme e523nvksd8e par exemple et de les changer régulièrement. Evidemment, pour les mémoriser c'est plus difficile...

Dans le cas des authentifications par mots de passe, il est toujours question d'authentification mutuelle bien sûr. On utilise le concept de challenge. Le serveur voulant authentifier un utilisateur, lui envoie un challenge. Ce challenge peut-être un nombre aléatoire ou un string, que l'utilisateur doit employer, additionné d'un secret partagé, pour calculer un digest à l'aide d'une fonction de hachage. Une fois le digest retourné au serveur, ce dernier peut vérifier que l'utilisateur possède bien le même secret que celui sauvegardé sur le serveur d'authentification.

Il existe plusieurs manières et protocoles pour réaliser un challenge. Citons CHAP (Challenge Handshake Authentication Protocol), MSCHAP (Microsoft CHAP), MD5-Challenge (Message Digest 5), LEAP (Lightweight EAP) de CISCO, OTP (One Time Passwords) ou encore SPEKE (Strong Password Authentication Methods). On peut encore nommer un autre principe basé sur des TokenCard ou des SmartCard.

6.3 Messages EAP

Tout utilisateur qui désire s'authentifier doit commencer par s'associer avec l'authentificateur (l'AP dans notre cas) au niveau liaison. Ceci peut se faire de plusieurs manières suivant la configuration de l'AP. Si le mode d'authentification est paramétré open, n'importe qui peut s'associer avec l'AP en utilisant les paquets *probe request* et *probe response* définis par la norme 802.11. Par contre, si le mode d'association par clé partagée (shared secret) ou encore par filtrage MAC est paramétré, l'utilisateur doit d'abord respecter les critères d'association. C'est uniquement après cette étape que l'échange des trames EAP peut commencer.

Quelque soit la méthode d'authentification employée, c'est toujours les mêmes paquets qui sont utilisés. La figure 6.2 représente le flux des messages échangés entre l'utilisateur et l'authentificateur. L'utilisateur peut commencer par émettre une trame start, mais ceci n'est pas obligatoire.

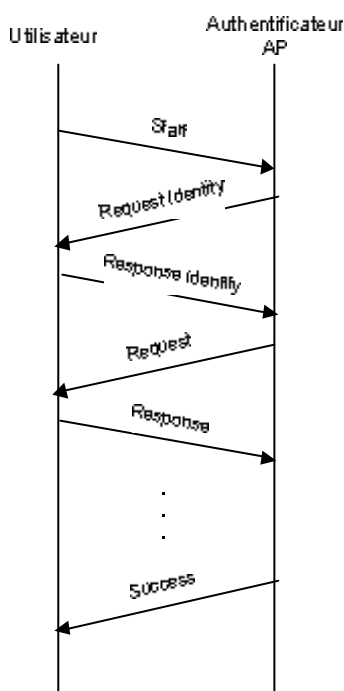
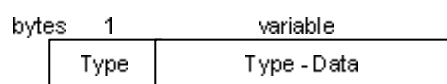
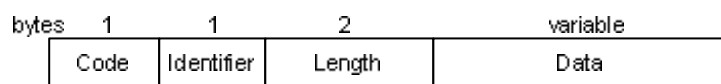


Fig. 6.2 Echange des messages EAP

Ensuite, l'authentificateur demande à l'utilisateur son identité à l'aide d'un paquet *Request identity*. L'utilisateur répond avec un paquet *Response identity*, qui contient notamment son nom d'utilisateur. Une fois l'identité vérifiée, le challenge peut commencer. L'authentificateur transmet à l'utilisateur au moyen du paquet *Request*, le challenge qui doit être effectué. L'utilisateur calcule alors la réponse et l'envoi grâce au paquet *Response*. Si le challenge est réussi, le paquet *Success* confirmera à l'utilisateur qu'il est bien authentifié. Sinon le paquet *Failure* se chargera d'annoncer le contraire. Comme pour le protocole RADIUS, il peut y avoir plusieurs échanges de challenge lors d'une seule session d'authentification.

Consacrons quelques lignes pour présenter l'en-tête d'une trame EAP. La figure 6.3 décrit cette en-tête.



7 Schéma d'authentification

Il est temps maintenant de mettre ensemble ces quelques concepts venant d'être vus pour réaliser un système d'authentification fort, robuste et fiable. La figure 7.1 nous montre cette solution. Un utilisateur voulant accéder au réseau doit d'abord s'associer avec l'AP au niveau liaison. Ensuite seulement, il peut s'authentifier au prêt du serveur RADIUS. Pour ce faire, étant donné que tous les ports d'application lui sont fermés au niveau de l'AP, l'utilisateur doit employer le protocole EAP pour communiquer à l'AP ses données d'authentification (trame EAP *Request Identity* et *Response Identity*). L'AP va ensuite transmettre ces données vers le serveur RADIUS par l'intermédiaire du protocole RADIUS et du paquet *Access Request*.

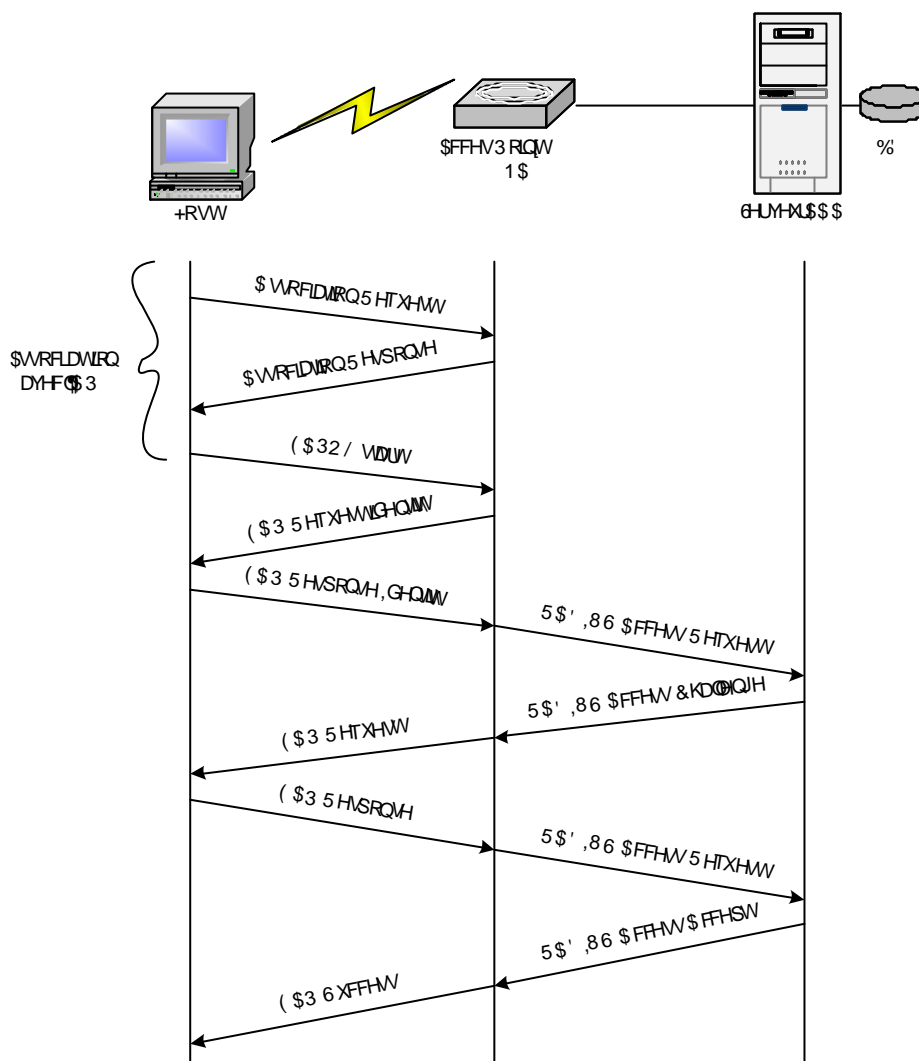


Fig. 7.1 Réalisation de l'authentification

Le serveur RADIUS va alors vérifier l'identité de l'utilisateur et transmettre le challenge vers l'AP. Ce dernier va faire suivre ce challenge vers l'utilisateur sous la forme d'un paquet EAP *Request*. Une fois le challenge rempli, l'utilisateur peut envoyer la réponse à l'AP qui va la rediriger vers le serveur RADIUS grâce au paquet *Access request*. Si le challenge est réussi, le

serveur RADIUS confirme l'authentification avec le paquet *Access Accept* qui va transiter par l'AP avant d'arriver à l'utilisateur sous la forme d'un paquet *Success* du protocole EAP.

Vous trouverez en annexe 2, la copie des trames échangées lors d'une authentification par challenge MD5 entre un AP et le serveur freeRADIUS, ainsi que les trames EAP échangées lors de la même session d'authentification, mais entre l'utilisateur et l'AP.

Nous avons donc maintenant un schéma d'authentification tout à fait sécurisé et très flexible du point de vu des nombreuses méthodes d'authentification qui peuvent exister. Il va être question maintenant de crypter les données qui vont transiter dans les airs.

Il faut noter une chose très importante : les paquets EAP qui circulent dans les airs entre l'utilisateur et l'AP ne sont pas cryptés si vous n'avez pas activé WEP ou tout autres systèmes de cryptage au niveau liaison ! Bien qu'aucuns mots de passe, secrets partagés ou encore certificats ne circulent directement en clair, mais plutôt sous la forme d'un digest ou de clés publiques, ne suffit pas forcément pour garantir une bonne sécurité. Cela reste encore une faille importante compte tenu des problèmes qui sont liés au protocole WEP et à l'algorithme RC4.

Il est impératif de trouver rapidement une solution à ce problème en particulier, mais également à celui que pose la transmission des données à proprement parler. En effet, il faut pouvoir disposer d'un algorithme performant pouvant répondre aux besoins des différents types de données qui composent le trafic. Notamment les données de la voix ou de la vidéo, dont il est surtout question dans ce travail de diplôme.

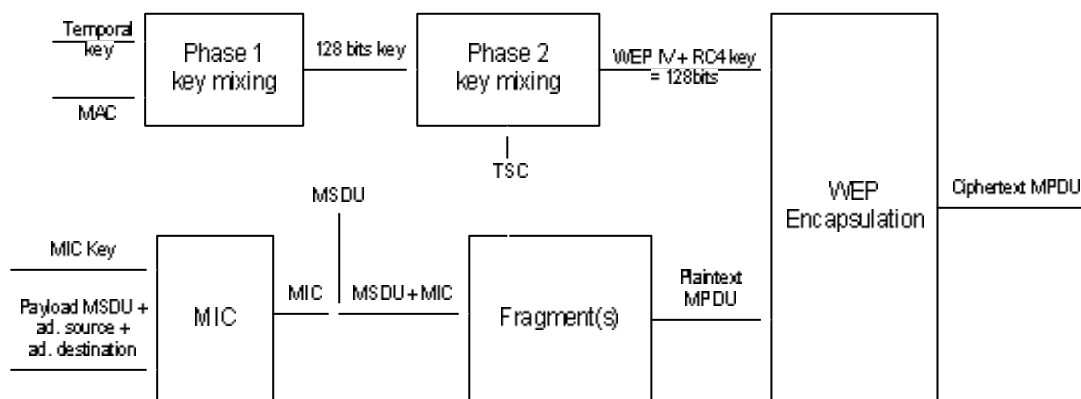
Nous entrons maintenant dans la deuxième étape de la sécurisation de la plateforme. Comme WEP n'est pas suffisant, il a fallu trouver des alternatives pouvant résoudre ces problèmes. Donc, une norme est en train d'être mise sur pied afin de répondre aux exigences du marché. Il s'agit de IEEE 802.11i. Se basant sur 802.1x, EAP et l'authentification forte, 802.11i définit surtout un nouvel algorithme d'encryption nommé AES. Mais ce genre de développement prend du temps, est une solution temporaire a été mise en place, il s'agit de WPA.

8 WPA

Wi-Fi Protected Access a été défini pour répondre rapidement aux faiblesses de WEP. WPA préconise l'utilisation d'un serveur d'authentification et de 802.1x, de la même manière que celle vu jusqu'à présent. Mais surtout, WPA instaure une gestion des clés dynamique. Ceci résout les problèmes de WEP, qui se situent essentiellement au niveau de la gestion des clés comme nous avons pu le voir dans le chapitre 2. Ce mécanisme de clés est assuré par le protocole TKIP (Temporal Key Integrity Protocol) et MIC (Message Integrity Check). WPA a été conçu pour pouvoir être implémenté dans le matériel existant par une simple mise à jour du firmware des cartes 802.11 et des points d'accès supportant WEP. Pour plus de détails se référer au standard Wi-Fi Alliance [12].

8.1 TKIP

Initialement, TKIP a été référencé en tant que WEP2. C'est une solution qui fixe les problèmes de réutilisation des clés de WEP. Les fondements TKIP sont les suivants. Ne jamais réutiliser le même IV pour plus d'une clé parmi plusieurs clés de session. Le récepteur doit éliminer tout les paquets qui posséderaient une valeur de l'IV qui serait plus petit ou égale au dernier paquet reçu avec succès encrypter avec la même clé. Générer régulièrement une clé de session aléatoire avant que le compteur d'IV dépasse sa capacité. Produire un mixage plus minutieux entre l'IV et la clé de session pour en dériver la clé RC4. Tout cela dans le but de fixer les défauts de WEP. Pourtant TKIP utilise toujours WEP 128 bits. Mais il emploie l'adresse MAC de la carte de l'utilisateur comme paramètre de calcul de la clé de session. De cette manière, chaque utilisateur possède une clé différente. De plus, TKIP repose partiellement sur MIC (Message Integrity Code), appelé Michael, un digest qui est appondu aux données avant que le tout soit crypté. Il faut savoir que le fait d'ajouter le MIC au payload entraîne la fragmentation des paquets. Finalement, un compteur de 16 bits appelé TSC (TKIP Sequence Counter) sert à conserver un ordre parmi les fragments. Il est réinitialisé à zéro lorsque la clé temporelle est réinitialisée ou rafraîchie. La figure 8.1 décrit la procédure d'encryption.



Le processus débute avec une clé temporelle de 128bits, partagée entre l'utilisateur et l'AP. La phase 1 de TKIP combine cette clé avec l'adresse MAC de l'utilisateur pour créer une clé de 128 bits, qui sera mixée avec le compteur TSC (phase 2) pour produire la clé de 128 bits servant à l'encryption. Cette clé doit s'interpréter de la même manière qu'une clé WEP. C'est-à-dire que les 24 premiers bits représentent l'IV, tandis que les 104 autres forment la clé secrète. Notons que le TSC est incrémenté à chaque fragmentation.

Parallèlement, l'algorithme MIC utilise une clé de 64 bits (MIC key), ainsi que le payload, l'adresse source et l'adresse de destination du paquet à crypter pour générer le Message Integrity Code, qui sera ajouté au MSDU avant d'être fragmenté. Chaque fragment se verra crypté avec une clé de flux différente.

L'avantage de cette solution temporaire est qu'elle est compatible avec le matériel que vous avez déjà acheté. Il suffit de se renseigner auprès du fabricant pour accéder aux différentes mises à jour des firmewares des cartes 802.11. Donc, WPA se présente plutôt comme une solution intéressante, compte tenu des problèmes et de l'insécurité que WEP propose. Malheureusement, TKIP utilise encore RC4. C'est le problème que 802.11i doit résoudre. Pour ce faire, un nouvel algorithme d'encryption au niveau liaison a été mis en place, il s'agit d'AES.

9 802.11i

C'est la solution que tout le monde attend ! 802.11i reprend WPA sur certains points. En effet, l'utilisation de 802.1x, EAP et un serveur d'authentification est de mise. Il y a l'équivalent de TKIP nommé CCMP, qui s'occupe de la création des clés de session, de leurs gestions et de l'encryption. Par contre, CCMP se base un nouvel algorithme d'encryption nommé AES, qui offre une meilleur sécurité que WEP. C'est le Task Group i de l'IEEE qui développe cette nouvelle norme qui devrait apparaître fin 2003 en théorie. Vous trouverez tous les détails concernant l'avancement du projet sur le site de IEEE [13]. Vu la situation au jour du 18 décembre 2003, la sortie du matériel certifié 802.11i semble être remise pour début 2004. En revanche, il est possible de trouver du matériel supportant déjà AES, et qui pourra être mis à jour lors de la sortie de 802.11i (le matériel Buffalo qui sera employé après par exemple). 802.11i défini ce que l'on appel un Robust Security Network (RSN), qui dénomme un réseau 802.11 qui utilise 802.1x et CCMP, alors qu'un réseau supportant 802.1x, TKIP et CCMP est appelé Transition Security Network (TSN).

9.1 AES

Advanced Encryption Standard offre une encryption plus forte. Le NIST (National Institutes of Standard and Technology) préconise AES [14] comme remplacement à DES. AES a fait l'objet d'une publication FIPS (Federal Information Processing Standard) [15]. Cet algorithme est donc maintenant reconnu comme un standard et est utilisé entre autre par le gouvernement des Etats-Unis pour protéger des informations sensibles. Le problème d'AES, est qu'il demande plus de ressource et de puissance de calcul que les cartes actuelles. Donc, l'achat de nouveaux adaptateurs est inévitable pour l'implémentation d'AES.

Ce standard est un algorithme symétrique à blocs de 128 bits, utilisant une clé d'encryption de 128, 192 ou 256 bits (AES_128, AES_192 et AES_256). Contrairement à WEP, AES utilise un tableau à 2 dimensions appelé tableau d'état (State Array), formé de 4 rangées r et de Nb colonnes c , Nb étant égale au quart de la longueur du bloque (4 dans le cas d'AES_128). Ce tableau est montré par la figure 9.1. Les éléments du tableau, des bytes, sont référencés par r et c , ce qui donne S_{rc} .

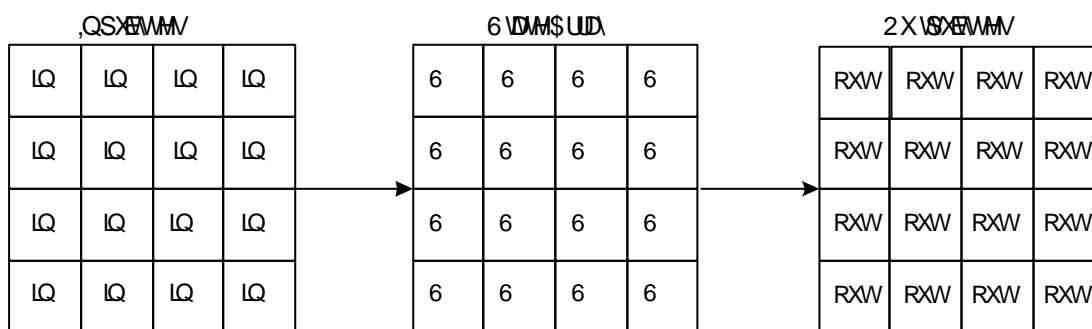


Fig. 9.1 Tableau interne de AES

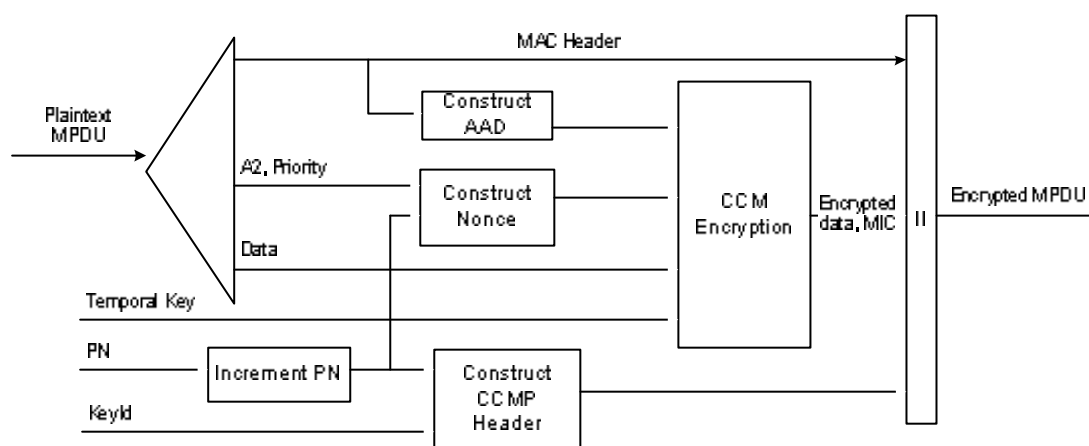
Chacune des colonnes comporte 4 bytes, ce qui fait 16 bytes en tout, donc 128 bits qui correspondent bien à la taille d'un bloque. Le tableau est d'abord initialisé avec un bloc de 16 bytes de données à crypter. Ensuite, un certain nombre de permutations vont être effectuées sur le tableau d'état suivant l'algorithme de Rijndael. Finalement, le tableau final sera copié vers la

sortie comme le montre la figure 9.1. Pour tous les détails de l'algorithme de Rijndael, se reporter au document [16].

En soit, AES en fait que crypter les données, selon une clé qui lui est passée en paramètre. C'est pourquoi, il a été rajouté autour d'AES, un mécanisme de création et de gestion des clés appelé CCMP.

9.2 CCMP

Ce protocole fonctionne selon la même idée que TKIP, mais emploie AES à la place de WEP. Counter CBC-MAC Protocol (CCMP) utilise AES en mode *Counter Mode with CBC-MAC* (CCM). Le mode CCM quant à lui, combine le Counter Mode (CTR) pour garantir la confidentialité (encryption) et le Cipher Block Chaining Message Authentication Code (CBC-MAC) pour garantir l'authentification et l'intégrité des données (MIC). La figure 9.2 explique le cheminement à suivre pour encrypter les données.



10 Les solutions des couches supérieures

Parlons tout de même un peu de l'encryption au niveau de la couche 3. Pour les réseaux câblés, l'encryption au niveau réseau est tout à fait suffisante dans la majeure partie des situations. Par contre, dans le cas des WLAN, cela pose deux problèmes qu'il ne faut pas négliger. Comme nous l'avons vu, sans l'encryption au niveau liaison, les trames EAP passeraient en clair. Ce qui est très dangereux, car les paquets EAP révèlent beaucoup de choses, comme le nom d'utilisateur, la méthode d'authentification, le challenge, le nom de l'AP ainsi que sa marque et son type (grâce à l'adresse MAC). Voir annexe 2, échange EAP entre l'utilisateur et l'AP CISCO. Deuxièmement, si WEP ou AES sont activés, le fait de rajouter une encryption au niveau réseau peut sembler « lourd ». Le coût en performance peut devenir inacceptable.

Il existe des protocoles efficaces adaptés pour certaines applications particulières. Nous allons brièvement parler d'IPsec pour les VPN et de SSL pour les applications Internet.

10.1 IPsec

Dans le domaine de la sécurité, on ne peut pas contourner IPsec. Il s'agit d'un moyen efficace de crypter les données au niveau réseau, mais ce n'est pas tout. Un mécanisme d'authentification forte est disponible, basé sur des certificats et des clés RSA. L'intégrité des données est assurée, ainsi que leur authenticité. Mais surtout, IPsec est capable de créer des VPN en encapsulant la trame IP dans une autre trame IP, on parle du mode tunneling.

C'est toute une gamme de protocoles qui composent IPsec, dont certains existent à part entière hors d'IPsec. Le protocole initial est IKE (Internet Key Exchange) défini par le RFC 2409 [17]. C'est le protocole administratif qui va s'occuper de créer et de gérer les différents tunnels. L'échange des données s'appuie sur deux autres protocoles. Authentication Header (AH), défini par le RFC 2402 [18], pour l'authentification des deux extrémités et Encapsulating Security Payload (ESP), défini par le RFC 2406 [19], pour chiffrer les données. Ce dernier permet également de garantir l'authenticité des données. Ce sont DES, MD5, SHA-1 ou encore Diffie-Hellman qui sont employés comme algorithmes d'encryption ou de hachages. Voici un site très complet sur le sujet de IPsec et ces derniers versions [20].

Ils est donc tout à fait envisageable d'implémenter IPsec sur un réseau 802.11. Mais il faut bien avoir en tête que ce n'est que depuis le niveau réseau que les données sont cryptées, et que si vous remédiez à ce problème en activant WEP ou AES, la débit de votre réseau risque de chuter.

10.2 SSL

Secure Socket Layer, standardisé par le RFC 2246 [21], «Transport Layer Security», peut s'avérer une bonne solution pour des communications Internet. Ce mécanisme rajoute une couche (SSL Record Protocol) entre le niveau transport et des protocoles comme http, ftp, Telnet et pourquoi pas ssh ! Mais n'exagérons pas tout de même. De plus, les protocoles cités sont encapsulés dans une trame, comme le montre la figure 10.1.

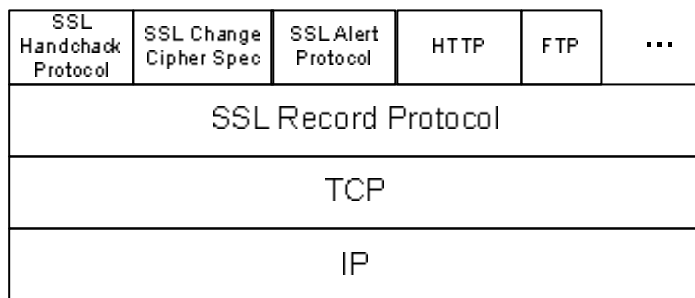


Fig. 10.1 Couche SSL

Ce protocole offre l'encryption des données, l'authentification du serveur, l'intégrité des messages et optionnellement il permet d'authentifier l'utilisateur pour une connexion TCP/IP. Il suffit juste d'installer un certificat. Ce protocole possède deux formats de clé de session, 40 bits et 128 bits, qui sont générées automatiquement pour chaque transaction.

Actuellement, SSL est implémenté dans la majeure partie des navigateurs et des serveurs WEB. Donc, lorsque vous allez surfer sur des sites de e-banking ou d'achats, une connexion SSL sera de toute manière entreprise, que vous aillez ou non un protocole de niveau deux installé. Comme pour IPsec, cela peut diminuer le débit de votre liaison sans fil. Pour plus de détails sur SSL, se référer au RFC 2246 [21]. La version open source de ce protocole pour Linux se nomme openssl. Vous trouverez tous détails sur le site [22].

En essayant de réfléchir aux problèmes de performance que posent les protocoles d'encryption de plus haut niveau, on s'aperçoit d'abord qu'ils sont indispensables lorsque que les paquets ont passés le routeur d'accès de votre société et qu'ils traversent Internet. En effet, un VPN ou une connexion SSL est vivement recommandée pour toutes transactions confidentielles qui traversent Internet. Par contre, entre le client 802.11 et l'AP où les performances sont importantes compte tenu du faible débit, l'encryption dès la couche deux pourrait suffire pour garantir une bonne sécurité. De ce fait, pourquoi rajouter un deuxième protocole d'encryption plus haut, alors que cela coûte en performances ? Pour résoudre ce problème, on pourrait imaginer une sorte de proxy qui servirait d'extrémité du tunnel IPsec ou SSL à la place du client 802.11. Le rôle de ce proxy serait alors de décrypter les paquets IPsec ou SSL puis de les transférer vers l'AP qui se chargerait de les encrypter avec le protocole de liaison configuré sur le WLAN. Ce proxy devrait être situé juste derrière l'AP qui fournit l'accès aux clients et connecté de manière sûre avec cet AP, car les trames y circuleraient en clair. Bref, tout ceci n'est que suppositions.

Voilà, nous avons fait le tour des solutions qui existent et qui possèdent un intérêt dans notre situation. Nous avons là la possibilité de construire une architecture complète couvrant tous les aspects de la sécurité. En résumant un peu, plusieurs éléments sont à notre disposition. Nous avons un mécanisme d'authentification composé d'un serveur AAA (RADIUS) et d'un point d'accès pouvant gérer l'accès au réseau et faire office de passerelle d'authentification entre le serveur et l'utilisateur (802.1x et EAP). Des algorithmes d'encryption de niveau deux (RC4 et AES). Des protocoles de gestion des clés (TKIP et CCMP). Et même des solutions de plus haut niveau, ciblées pour des applications bien précises (IPsec et SSL). Il est temps d'implémenter tout ça !

11 Mise en œuvre de la plateforme d'authentification

Passons maintenant à la pratique. Ce chapitre décrit l'ensemble de la plateforme mise en place. Une description détaillée de l'installation est expliquée sous la forme d'une marche à suivre. Ce chapitre inclus donc l'installation mais également la configuration de la plateforme. Il faut noter que la partie la plus importante est la mise en place du système d'authentification. Une fois celui-ci fonctionnel, le reste se résume, de manière caricatural, à quelques cases à cocher et boîtes de dialogue agréables.

11.1 Matériel

Voici la liste complète du matériel mise à ma disposition durant le travail de diplôme :

- Portable DELL Inspiron 2650 avec une carte PCMCIA AVAYA SilverCard 802.11b possédant un chip orinoco et le firmware 8.10. OS Linux RedHat 9. Cet ordinateur est utilisé pour sniffer les paquets et craquer WEP. Il est nommé *inspiron* dans ce rapport.
- Station desktop DELL optiplex GX270 **sans** carte 802.11b. OS Linux Suse 9.2. Cet ordinateur est employé comme serveur freeRADIUS et comme générateur de trafic pour l'attaque de WEP. Il est nommé *serveur* dans ce rapport.
- Station desktop DELL dimension 4550 avec une carte PCI syslink WMP 11 v2.7 802.11b. Version 3.8.28.0 des pilotes. OS Windows 2000 professionnel, service pack 4. Cet ordinateur est employé comme utilisateur windows, mais également lors de l'attaque de WEP. Il est nommé *dimension* dans ce rapport. Le gestionnaire de cette carte (qui permet de configurer les profils) s'appelle *Linksys Wireless PCI Card WLAN Monitor ver 1.05* et se lance automatiquement lors du démarrage de windows. Il est donc accessible depuis la bar des tâches.
- Station desktop DELL optiplex GX1 avec une carte PCI D-Link DWL-G520 802.11g. Version 2.1.3.1 des pilotes. OS Windows 2000 professionnel, service pack 4. Cet ordinateur est employé comme utilisateur windows. Il est nommé *optiplex* dans ce rapport. Le gestionnaire de cette carte (qui permet de configurer les profils) s'appelle *D-Link AirPlus Xtreme G Configuration Utility* et se lance automatiquement lors du démarrage de windows. Il est donc accessible depuis la bar des tâches.
- Point d'accès ASUS SpaceLink WL-300 802.11b, firmware 1.4g.7. Cet AP est employé pour l'attaque de WEP et pour des tests ne nécessitant pas d'authentification, car ne supporte pas 802.1x. Il est nommé *AP ASUS* dans ce rapport. Pour configurer cette AP, un utilitaire est fournis sur le CD d'installation. Il se nomme ASUS AP Manager. Très intuitif et facile à prendre en main. Le nom du BSS géré par cet AP est *chouffleur*.
- Point d'accès CISCO Aironet série 350 802.11b, firmware 12.03T. Cet AP est employé pour l'authentification, car supporte 802.1x. Il est nommé *AP CISCO* dans ce rapport. Pour configurer cette AP, un site web est installé directement dans l'AP. Il suffit de rentrer l'adresse IP de l'AP pour y accéder. Le nom du BSS géré par cet AP est *tsunami*.
- Switch 8 ports Allied Telesyn AT-FS708-LE, pour relier le serveur freeRADIUS à l'AP. Il est nommé *switch* dans ce rapport.

- Point d'accès Buffalo AirStation WBR-G54 de la norme 802.11g. Cet AP est utilisé pour les mesures TKIP et AES. Il est nommé AP Buffalo dans ce rapport. Pour configurer l'AP, une interface WEB est disponible. Le nom du BSS géré par cette AP est *kiwi*.
- Adaptateur pcmcia Buffalo WLI-CB-G54 de la norme 802.11g avec les pilotes 3.30.15.1. Cette carte est utilisée pour les mesures TKIP et AES.

11.2 Schéma de la plateforme

La plateforme utilisée pour les tests et les mesures est montrée en figure 11.1. Elle est composée de deux AP, l'un avec authentification (AP CISCO) et l'autre sans authentification (AP ASUS), du serveur AAA et de deux utilisateurs.

Dans un premier cas de figure, deux utilisateurs s'authentifient auprès du serveur freeRADIUS par l'intermédiaire du client AP CISCO, qui joue le rôle de NAS. Dans un deuxième cas de figure, où l'authentification n'est pas requise, les utilisateurs emploient l'AP ASUS, qui ne requiert qu'une clé WEP.

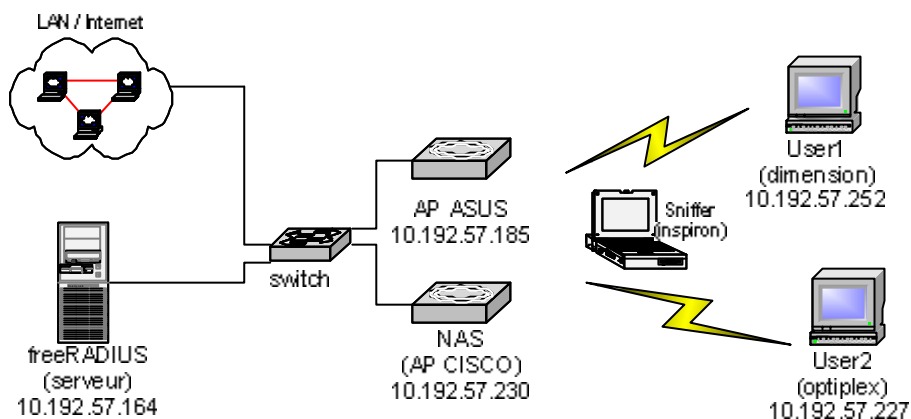


Fig. 11.1 Plateforme

Il n'y a qu'un seul sous réseau, 10.192.57.0. Il est à noter que les adresses IP peuvent éventuellement être changées, étant donné que DHCP est activé. Il est donc conseillé de vérifier les adresses avant tous les tests ou mesures, bien que la majeure partie du temps, les mêmes adresses IP sont attribuées aux mêmes adresses MAC.

11.3 Serveur freeRADIUS

La version de freeRADIUS installée est 0.9.2 et a été téléchargée depuis le site officiel de freeRADIUS [23]. Vous y trouverez également toute la documentation nécessaire. Il s'agit du paquetage freeradius-0.9.2.tar.gz. Il existe déjà la version 0.9.3 (sortie le 20 novembre 2003) qui est disponible sur le site. La machine sur laquelle est installé freeRADIUS est le DELL optiplex GX270. Il n'y a pas vraiment de pré-requis pour le serveur dans mon cas. En effet, pour un nombre faible d'utilisateurs, une machine ordinaire suffit amplement. Par contre, si plusieurs centaines d'utilisateurs s'y authentifient, il va certainement falloir investir dans une machine plus performante. En effet, une base de données relativement grande est requise pour contenir toutes les données d'authentification, d'autorisation et d'acompte relatives à chaque utilisateur. Il faut envisager un système de backup du serveur. De plus, si plusieurs dizaines d'utilisateur s'authentifient en même temps, il faut penser à la rapidité du CPU et la RAM afin de répondre aux requêtes dans un délai minimum. N'ayant pas réellement eu besoin de me soucier de ce problème

dans le cadre de ce projet de diplôme, il est normalement indispensable de s'assurer des performances requises pour le serveur avant d'investir de l'argent et du temps.

Dans la pratique, il faut placer ce serveur dans un local bien gardé, à l'abri de toute attaque par le réseau, derrière un firewall. Certaines conventions devraient être strictement respectées, comme les mots de passe qui doivent être régulièrement changés. Le compte *root* ne doit être employé que par l'administrateur système lors d'opération importante et éventuellement un utilisateur avec privilège peut être configuré. Finalement, il est vivement conseillé de ne pas implémenter d'autres services sur la même machine, comme http, FTP, email... Donc, dans la pratique, ces quelques points doivent être rigoureusement appliqués, mais dans le cadre de ce projet, ils ont été mis de côté.

11.3.1 Installation

Cette étape n'est pas très compliquée. Comme nous le verrons, c'est la configuration qui prend le plus de temps. L'installation se résume à la décompression d'un paquetage, à sa configuration, sa compilation et son l'installation. Commençons par citer qu'une connexion SSH est possible afin d'administrer le serveur à distance de manière sécurisée.

Pour débiter, une fois le paquetage téléchargé et décompressé dans le répertoire `/usr/src/` comme d'habitude, il faut créer le fichier Makefile servant à la compilation. Pour ce faire, il suffit de se placer dans le répertoire `freeradius-0.9.2` puis de taper la commande suivante : `./configure`. Cette étape a pour but de vérifier les dépendances et la présence de certaines bibliothèques. Un message d'erreur vous avertira si un paquetage requis vous manque. Il est possible maintenant de compiler les fichiers de freeRADIUS en tapant la commande `make` et de les installer avec la commande `make install`. C'est tout ! Avant de lancer le serveur, il faut penser à la configuration.

11.3.2 Configuration

Comme c'est majoritairement le cas avec Linux, la configuration s'effectue à l'aide de fichier texte. Il y a 3 fichiers qui nous intéressent et sont tous placés dans le répertoire `raddb`, dont la localisation dépend de la distribution et de la version de freeRADIUS. Pour Suse 9.2 et freeRADIUS 0.9.2, il s'agit de `/usr/local/etc/raddb`. Il est clair que les fichiers de configurations ne sont pas très conviviaux. Cela peut effrayer au début. Mais ils sont vraiment bien commentés et très vite l'ensemble devient clair et accessible.

Bien sûr, je n'ai de loin pas exploré toutes les possibilités de configuration. Je n'ai qu'effleuré les capacités de freeRADIUS. Ma démarche de configuration est triviale et correspond au minimum requis pour monter un serveur d'authentification.

radiusd.conf

Il s'agit du fichier principal de configuration. Il contient plus de 1000 lignes, mais rassurez-vous, il y a énormément de commentaires. Il est divisé en sections (Security, proxy, thread pool, modules, authorize et authenticate), qui regroupent les paramètres relatifs au même point de configuration. Il est impératif que ce fichier possède les droits administrateurs uniquement. Personne d'autre ne doit avoir accès à ce fichier. C'est très important. Tout est très bien expliqué et commenté. Pour chaque attribut et section, un commentaire adéquat se trouve juste avant. Comme quoi, on s'aperçoit que les commentaires dans un code sont très importants et simplifient grandement la vie de la personne qui relit ultérieurement le code...

Au début du fichier se trouvent les paramètres généraux, comme les variables d'environnement qui permettent entre autres de configurer la localisation des autres fichier de configurations, le numéro du port à employer (en général 1812), le temps maximum que peut prendre le traitement d'une requête, le nombre maximum de requêtes dont le serveur doit garder une trace. On peut y

spécifier encore la gestion des fichiers logs. Voici les variables d'environnement principales, qui définissent les localisations des différents fichiers importants. Bien entendu, il y en a d'autres qui ne sont pas montrées ici.

```
prefix = /usr/local
sysconfdir = ${prefix}/etc
logdir = ${localstatedir}/log/radius
raddbdir = ${sysconfdir}/raddb
radacctdir = ${logdir}/radacct
```

En principe il n'y a que le numéro du port à changer pour que sa valeur soit de 1812 et non de 1645 comme c'était le cas il y a quelques versions précédentes.

```
port = 1812
```

Sinon, dans beaucoup de cas triviaux, tous les autres paramètres sont déjà corrects pour une utilisation classique. Il est bien entendu conseillé d'affiner ces paramètres en fonction de vos besoins. On peut par exemple spécifier si les espaces sont accordés dans les noms des utilisateurs, ou encore à partir de quel utilisateur et quel groupe le service radiusd doit être lancé.

La section `security` permet de protéger le serveur contre quelques attaques DoS notamment. On peut y spécifier le nombre d'attributs maximums par requête, ou encore si le serveur doit répondre ou non au message Status-Request, qui serait source de DoS. Un autre paramètre permet de spécifier le temps d'attente avant de transmettre un Access-Reject afin de diminuer également les risques de DoS. Dans mon cas, je n'ai apporté aucune modification à ce niveau.

La section `proxy` permet de configurer les paramètres relatifs à la communication avec d'autres serveurs RADIUS. N'ayant pas besoin de cette fonctionnalité, il faut mettre la variable `proxy_request` à `no` et mettre en commentaire la ligne juste en dessous dans le fichier :

```
proxy_requests = no
#$INCLUDE ${confdir}/proxy.conf.
```

Une section `thread pool` permet de configurer le nombre de serveurs qui traitent les requêtes et le nombre de processus lancés. Il peut être utile, pour des raisons de performance et de montée en charge, de limiter le nombre de thread créé. Là non plus, les performances n'étant pas un problème, cette section n'a subi aucune modification.

Ensuite, vient la section `modules`. C'est l'une des plus importantes. Elle permet d'activer et de configurer les méthodes d'authentification vues précédemment, CHAP, LEAP, EAP-TLS. Cette section est la plus longue. On peut vraiment aller dans le détail. Dans mon cas, je n'ai touché qu'une petite partie d'un module particulier, celui pour MD5. Et comme vous allez pouvoir le constater, ce n'est vraiment pas compliqué d'activer ce moyen d'authentification, et encore moins de le configurer. J'ai choisi un challenge MD5 pour des raisons de simplicité de configuration. Il est vrai que ce n'est pas la méthode la plus sûre, et que c'est plutôt sur une solution basée sur des certificats qui est conseillée. Mais la mise en œuvre d'une architecture de certificat est compliquée et prend du temps. Voici la liste des méthodes d'authentification supportées par freeRADIUS :

CHAP, /etc/passwd, PAM, LDAP, MySQL DB, PostgreSQL DB, Oracle SQL DB, Kerberos, X9.9 authentication token, EAP (MD5, LEAP, TLS), MS-CHAP,...

Pour activer MD5, il s'agit d'ajouter dans la liste des mécanismes d'authentifications acceptés, dans le module eap, la méthode MD5, comme le montre l'exemple ci-dessous :

```
modules {  
    eap {  
        md5 {  
        }  
    }  
}
```

Il est à noter que toutes les autres méthodes eap ont été désactivées, c'est-à-dire mises en commentaire.

La section `authorize` spécifie l'ordre dans lequel le serveur doit vérifier les identités des utilisateurs désireux de s'authentifier. Par défaut, c'est seulement par le fichier `users` que l'on peut vérifier les utilisateur. Mais il est possible de rajouter les méthodes d'authentification paramétrées dans la section supérieure.

```
authorize {  
    files  
    eap  
}
```

Ici, les utilisateurs sont enregistrés dans le fichier `users` que nous allons voir plus loin, et non dans une base de données. On verra juste après que, au lieu de `file`, il faut avoir `sql`, pour activer la base de données. Donc, c'est une fois seulement que le serveur a pu vérifier l'existence de l'utilisateur que le module eap va être invoqué, comme il n'y a que la méthode MD5 qui est définie, c'est forcément un challenge MD5 qui va être entrepris.

La section `authenticate` précise quels modules sont disponibles pour l'authentification. Il ne faut pas confondre avec la section précédente. Celle-ci permet de paramétrer quelles sont les valeurs acceptées pour l'attribut `Auth-Type` de chaque utilisateur, qui permet d'invoquer une méthode d'authentification.

```
authenticate {  
    eap  
}
```

Ainsi, le paramètre `Auth-Type` des utilisateurs voulant s'authentifier grâce à un challenge MD5 doit avoir pour valeur `eap`.

Voilà, c'est la fin du fichier de configuration *radiusd.conf*! Il faut admettre que lorsqu'on l'ouvre pour la première fois, on prend peur. Mais très rapidement, sa structure s'éclaircit et cela devient tout à fait maîtrisable.

clients.conf

Voici le fichier permettant de configurer les clients, c'est-à-dire les NAS. Grâce à ce fichier il est possible de paramétrer certaines caractéristiques du client. L'adresse IP, le secret partagé ou encore le type de client. Comme premier exemple, observons le client local, qui sert à effectuer des tests sur le serveur après certaines configurations. Nous allons voir plus son utilité lors de la commande `radtest`.


```
client 127.0.0.1 {  
    secret = test123  
    shortname = localhost  
    nasstype = other  
}
```

Il faut noter qu'après le mot clé *client*, c'est le nom de la machine dans le domaine, ou l'adresse IP qui sont acceptés. De plus, le secret est en clair. Donc, toujours autant de précautions qu'avec *radius.conf* sont à prendre au sujet des droits d'accès sur ce fichier. Le secret partagé entre l'AP et le serveur d'authentification sert, lorsque le serveur reçoit un *Access Request*, à vérifier que le client est bien reconnu et n'est pas un AP pirate qui aurait été installé dans le but de détourner le trafic, mais le secret est utile pour encrypter les données entre l'AP et freeRADIUS.

Il est en principe très simple de définir un client. Voici le seul de notre schéma, il s'agit de l'AP CISCO :

```
client 10.192.57.230 {  
    secret = aironet  
    shortname = ap8021x  
    nas-ip-address = 10.192.57.230  
    nasstype = cisco  
}
```

Il s'agit de la même déclaration que le client *localhost*, sauf que l'on peut rajouter l'adresse IP de l'AP, ainsi que le type du NAS. Plusieurs types sont supportés : *cisco*, *computone*, *livingston*, *max40xx*, *multitech*, *netserver*, *pathras*, *patton*, *portslave*, *tc*, *usrhiper* et *other* pour tous les autres types non cités, comme le *localhost*.

users

Ce fichier regroupe les utilisateurs à authentifier et tous leurs attributs requis. Là aussi, les droits sur le fichier doivent être strictement limités à l'administrateur root. Commençons par l'utilisateur par défaut. Il s'agit de la déclaration dont tous les utilisateurs tiendront compte. Les critères définis dans l'utilisateur *default* sont valables pour tous les utilisateurs. Il est possible d'avoir plusieurs utilisateurs *default*, qui sont interprétés selon leur ordre dans le fichier.

```
DEFAULT Group == "disabled", Auth-Type := Reject  
    Reply-Message = "Your account has been disabled."
```

Cet exemple permet de désactiver tous les utilisateurs d'un certain groupe. Le champ *Auth-Type* permet de préciser la méthode d'authentification. Ici, il s'agit de rejeter tous les utilisateurs. La définition de *userone* ci-dessous est différente. C'est par un échange EAP que l'authentification va se faire. Donc voici les trois utilisateurs que j'ai créé pour ma plateforme :

```
usertest Auth-Type = local, User-Password == "test"  
    Service-Type = Login,  
    Reply-Message = "Le test est réussi"  
  
userone Auth-Type := eap, User-Password == "user1"  
    Service-Type = Login,  
    NAS-IP-Address = 10.192.57.230  
usertwo Auth-Type := eap, User-Password == "user2"  
    Service-Type = Login,  
    NAS-IP-Address = 10.192.57.230
```


Il faut remarquer que le mot de passe y est enregistré en clair ! Donc là aussi, il est impératif que les droits sur ce fichier soient strictement réservés à l'administrateur root. La première ligne correspond aux attributs à vérifier (séparé par des virgules, sauf le dernier), alors que les lignes suivantes indiquent les attributs à retourner (un par ligne avec une virgule à la fin de chaque ligne sauf la dernière). Donc, on voit ici que l'utilisateur veut utiliser la méthode eap pour s'authentifier et on peut savoir également sur quel AP il doit être connecté.

Il peut vite devenir difficile de gérer un grand nombre d'utilisateurs sous la forme d'un fichier texte. C'est pourquoi la gestion d'une base de données est comprise dans la configuration. C'est grâce au fichier sql.conf que nous allons voir en dessous que cela se paramètre.

Il est temps de tester notre serveur et les utilisateurs créés. Pour commencer il est possible de lancer le serveur en mode *debug* en tapant la commande suivant :

```
radiusd -X
```

Lors du démarrage, le serveur vérifie la syntaxe des fichiers de configurations et charge les modules requis. Vous allez donc pouvoir contrôler que tout est en ordre. Si tout se passe sans erreurs, vous devriez avoir ce message en fin de chargement :

```
Listening on IP address *, ports 1812/udp and 1813/udp.  
Ready to process requests.
```

La console reste figée. Ensuite, à chaque action du serveur, ces événements sont affichés dans la console. La sortie complète affichée dans la console lors du démarrage du serveur est donnée en annexe 3.

Ensuite, il faut tester la configuration des utilisateurs. Il existe une commande pour réaliser cela. Voici comment réaliser le teste de l'utilisateur usertest :

```
radtest usertest test localhost 1812 testing123
```

Et vous devriez voir apparaître ceci, si tout est bien configuré :

```
Sending Access-Request of id 130 to 127.0.0.1:1812  
  User-Name = "usertest"  
  User-Password = "test"  
  NAS-IP-Address = y013lx10  
  NAS-Port = 1812  
rad_recv: Access-Accept packet from host 127.0.0.1:1812, id=130,  
length=46  
  Service-Type = Login-User  
  Reply-Message = "Le test est reussi"
```

Dans la console où l'on trouve le serveur en mode *debug*, il va s'afficher ceci :

```
rad_recv: Access-Request packet from host 127.0.0.1:32768, id=134,  
length=60  
  User-Name = "usertest"  
  User-Password = "test"  
  NAS-IP-Address = 255.255.255.255  
  NAS-Port = 1812
```

```
modcall: entering group authorize for request 1
  users: Matched usertest at 90
  modcall[authorize]: module "files" returns ok for request 1
rlm_eap: EAP-Message not found
  modcall[authorize]: module "eap" returns noop for request 1
modcall: group authorize returns ok for request 1
  rad_check_password: Found Auth-Type local
auth: type Local
auth: user supplied User-Password matches local User-Password
radius_xlat: 'Le test est reussi'
Login OK: [usertest/test] (from client localhost port 1812)
Sending Access-Accept of id 134 to 127.0.0.1:32768
  Service-Type := Login-User
  Reply-Message = "Le test est reussi"
Finished request 1
Going to the next request
```

En résumé, il affiche la requête (*Access Request*) qu'il vient de recevoir, vérifie que c'est par fichier texte qu'il doit contrôler l'existante de l'utilisateur. Comme le module EAP est chargé, il vérifie que le mode d'authentification n'est pas EAP (ce qui est le cas pour usertest). Il contrôle le mot de passe, vérifie que les attributs soient bien respectés, puis envoie le paquet *Access Accept*.

Tout est en ordre, le serveur freeRADIUS est maintenant configuré. Il est temps d'installer la base de données MySQL afin d'avoir à disposition un moyen robuste de gérer plusieurs utilisateurs.

11.3.3 Installation MySQL serveur

Il est vivement conseillé de le faire si vous devez gérer plusieurs centaines d'utilisateurs, voire même plusieurs milliers. Cela permet une gestion bien plus structurée et plus sûre. Plusieurs bases de données sont supportées. Pour des raisons de simplicité et de coût, c'est le serveur MySQL implémenté pour linux qui a été choisi. Il est vrai que dans notre cas il n'est pas utile de monter une telle base. Pour trois utilisateurs... Mais je pense que cela est indispensable dans un vrai cas pratique, donc j'ai pris le temps de le faire afin de me familiariser avec MySQL. J'ai suivi la marche à suivre *FreeRadius and MySQL* [24] pour installer et configurer la base et freeRADIUS.

La première étape consiste à installer le serveur MySQL. Dans mon cas, le paquetage adéquat se trouvait directement sur les CD de Suse 9.2. Il s'agit de la version 3.23.55-14. Pour tout téléchargement, mise à jour, sources et documentation, il faut visiter le site de mysql [25]. Donc, l'assistant Suse 9.2 pour la gestion des paquetages fait tout le travail d'installation. Pas besoin de `./configure`, `make` ou `make install`. Si vous êtes forcés de passer pas là, tout est bien expliqué sur le site ou directement dans un fichier texte (README ou INSTALL) lors de la décompression du paquetage téléchargé. Il n'y a pas plus d'explications quant à cette étape dans ce rapport.

Quand le serveur est bien installé, et que le service a démarré sans erreur, vous pouvez taper la commande `mysql` et voir s'afficher ceci :

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6 to server version: 3.23.55-Max-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Le prompt (`mysql>`) vous invite à taper des requêtes, des commandes divers pour gérer le serveur et les bases de données. Si le message d'erreur suivant apparaît c'est que vous devez rentrer le mot de passe root de votre serveur mysql pour vous y connecter :

```
ERROR1045: Access denied for user: 'root@localhost' (Using password: NO)
```

Ajoutez alors l'option `-p` à la commande `mysql` et une invite vous demandera d'entrer le mot de passe avant de vous autoriser à ouvrir une console mysql. L'étape suivante consiste à créer la base de données appelée `radius`. Tapez alors la commande suivante depuis la console sql:

```
create database radius;
```

Et le message suivant s'affiche pour confirmer que la commande a réussi:

```
Query OK, 1 row affected (0.00sec)
```

Pour créer la structure de la base `radius`, un script est fourni par `freeRADIUS` qui génère automatiquement les tables et relations de la base. Ce script se trouve dans `/usr/src/freeradius-0.9.2/src/modules/rlm_sql/drivers/rlm_sql_mysql/`. Le script se nomme `db_mysql.sql`. Pour le lancer, tapez la commande ci-dessus depuis le shell traditionnel:

```
mysql -uroot -prootpass radius < db_mysql.sql
```

Où `root` et `rootpass` correspondent à votre nom et mot de passe du serveur mysql. Rien ne s'affiche pour confirmer l'exécution correcte du script. Pour le vérifier, retournez dans la console mysql, connectez vous à la base `radius`, et affichez toutes les tables de la base. Suivez les deux commandes suivantes :

```
mysql> connect radius;  
mysql> show tables;
```

Vous devriez voir la liste des tables la base `radius` :

```
+-----+  
| Tables_in_radius |  
+-----+  
| radacct          |  
| radcheck         |  
| radgroupcheck    |  
| radgroupreply    |  
| radreply         |  
| usergroup        |  
+-----+  
6 rows in set (0.00 sec)
```

Vous possédez maintenant le schéma adéquat pour accueillir les données d'authentification des utilisateurs. Prenons quelques lignes pour expliquer le rôle des tables.

`usergroup`: fait le lien entre un utilisateur et l'appartenance à un groupe.
`radcheck`: contient les attributs à vérifier pour l'authentification (checks items)
`radgroupcheck`: contient les attributs de chaque groupe à vérifier (checks items)
`radreply`: contient les attributs à retourner à l'utilisateur (reply items)
`radgroupreply`: contient les attributs à retourner à tout un groupe (reply items)
`radacct`: contient les acomptes de tous les utilisateurs.

Vous trouverez le contenu complet des tables en annexe 4.

Il faut maintenant configurer freeRADIUS afin qu'il puisse se référer à cette base plutôt qu'au fichier de configuration `users`. Nous allons nous intéresser au fichier `sql.conf`.

sql.conf

Ce n'est vraiment pas compliqué. Il faut commencer par ajouter, dans le fichier `sql.conf`, le nom de serveur (IP ou nom de domaine), le login et le mot de passe de la base. Comme indiqué ci-dessous :

```
# Connect info
server = "localhost"
login = "root"
password = "rootpass"
```

En principe, le reste des paramètres doivent être correctes si vous utilisez le schéma proposé de la base. Afin d'afficher les propriétés `mysql` lors du lancement de freeRADIUS en mode debug, il faut modifier l'attribut `sqltrace` à `yes`. Cela peut vous aider si vous ne comprenez pas pourquoi la référence à la base de fonctionne pas. C'est tout pour `sql.conf`.

Il faut retourner dans le fichier `radiusd.conf` afin de spécifier l'utilisation de la base comme moyen d'authentification. Dans la section `authorize`, il faut mettre en commentaire `file` et rajouter à la place `sql`. Si vous souhaitez tout de même garder le fichier `users` valide, il ne faut pas mettre la ligne `file` en commentaire. Attention à l'ordre entre `file` et `sql`, c'est important. Finalement, si vous souhaitez également que les `acomptes` soient gérés par la base de données, il faut aussi rajouter une ligne `sql` dans la section `accounting`.

La configuration du serveur freeRADIUS avec une base MySQL est maintenant terminée. La prochaine étape consiste à ajouter les utilisateurs et leurs attributs dans la base. En annexe 4, vous trouverez le schéma de la base ainsi que la copie d'écran de certaines tables intéressantes. Le principe est clair, c'est avec des requêtes SQL que l'on ajoute les éléments dans les différentes tables. Exemple avec l'utilisateur `usertest` :

```
INSERT INTO usergroup (UserName, GroupName)
VALUES ('usertest', 'user') ;

INSERT INTO radcheck (UserName, Attribute, op, Value)
VALUES ('usertest', 'User-Password', ':=', 'test') ;

INSERT INTO radcheck (UserName, Attribute, op, Value)
VALUES ('usertest', 'Auth-Type', ':=', 'local') ;

INSERT INTO radcheck (UserName, Attribute, op, Value)
VALUES ('usertest', 'Service-Type', ':=', 'Login') ;

INSERT INTO radreply (UserName, Attribute, op, Value)
VALUES ('usertest', 'Reply-Message', ':=', 'Bienvenue usertest') ;
```

En cas de problème, voici la requête pour effacer une ligne dont l'id est connu:

```
DELETE FROM nomtable WHERE id='id_ligne'
```

Pour tester si tout est en ordre, relancez le serveur en mode debug et contrôlez le bon chargement des modules sql et de la liaison avec la base. Ensuite, réalisez le même test que tout à l'heure avec la commande `radtest`.

11.3.4 Interface graphique

Heureusement, une interface graphique existe pour passer outre ces nombreuses requêtes qui prennent vite beaucoup de temps. On pourrait imaginer créer des scripts pour automatiser l'ajout d'un utilisateur par exemple ou pour d'autres opérations. Mais une interface graphique est encore meilleure. Relativement basique, cette interface permet de gérer les utilisateurs et les groupes. Elle ne permet pas de configurer le serveur.

Malgré cela, cette interface est facile d'utilisation et bien présentée. Elle fonctionne sous la forme de page HTML et de scripts PHP4. Ce qui implique que sur la machine où tourne freeRADIUS, il faut installer un serveur http avec le module PHP4. Ceci est quelque peu contradictoire avec les critères d'un bon serveur RADIUS qui indiquaient de ne pas activer d'autres services sur le serveur pour des raisons de sécurité. Pour résoudre ce problème, on pourrait imaginer activer le service http uniquement lorsque le besoin s'en fait sentir. Personnellement, je n'ai pas utilisé cette interface, mais elle est quand même activée. La figure 11.2 représente une copie d'écran de la page d'ajout d'utilisateur.

Fig. 11.2 Interface graphique Dialup Admin

Les fichiers sources du site se trouvent dans le répertoire `dialup_admin` qui se trouve dans le répertoire source de freeRADIUS. Pour mon cas :

`/usr/src/freeradius-0.9.2/dialup_admin/htdocs.`

Une fois que votre serveur http fonctionne ainsi que le support php4, il suffit de placer toute l'arborescence du site à l'endroit prévu par le type de serveur web installé et de remplacer la page d'index. Pour un serveur apache 1.3.27-38 sous Suse 9.2, l'endroit où doivent se trouver les sources du site est `/srv/www/htdocs/`.

Il faut savoir que le fait d'installer l'interface rajoute 4 tables à votre schéma de base de données qui se transforme comme suit :

```
+-----+
| Tables_in_radius |
+-----+
| badusers          |
| mtotacct          |
| radacct           |
| radcheck          |
| radgroupcheck     |
| radgroupreply     |
| radreply          |
| totacct           |
| usergroup         |
| userinfo          |
+-----+
```

La table `userinfo` permet de stocker le nom, l'email et le numéro de téléphone des utilisateurs, la table `badusers` sert à enregistrer l'heure, le nom de l'utilisateur ainsi que le type d'erreur qui a empêché un utilisateur de s'authentifier. Les deux autres tables, `totacct` et `mtotacct` font partie de la gestion des acomptes.

Le serveur d'authentification `freeRADIUS` est maintenant opérationnel. Il s'agit certes d'un serveur par défaut relativement trivial, mais cela suffit pour l'instant. Passons maintenant à la configuration de l'AP 802.1x.

11.4 802.1x et EAP


Il faut configurer maintenant l'AP qui supporte 802.1x et qui va se charger de transférer les données d'authentification du serveur à l'utilisateur et vice versa. Pour tout les détails concernant la configuration de l'AP, il faut consulter la documentation CISCO Aironet350 [26]. Nous allons quand même nous attarder un peu sur les points qui nous intéressent, c'est-à-dire l'activation de l'authentification EAP et du serveur `freeRADIUS`. Pour accéder à la page de configuration de l'AP, il suffit de taper son adresse IP dans un navigateur. La figure 11.3 montre la homepage.

AP350-5c07fd Summary Status

Cisco 350 Series AP 12.04

Home Map Network Associations Setup Logs Help

Uptime: 06:47:45



Current Associations				
Clients: 9 of 10	Repeaters: 0 of 0	Bridges: 0 of 0	APs: 1	

Recent Events		
Time	Severity	Description
06:44:09	Info	Station [169.254.14.16]000d888bce50 Associated
06:44:09	Info	Station [169.254.14.16]000d888bce50 Authenticated
06:40:24	Info	Station [169.254.14.16]000d888bce50 Associated
06:40:24	Info	Station [169.254.14.16]000d888bce50 Authenticated
06:40:22	Info	Deauthentication from [169.254.14.16]000d888bce50, reason "Sender is Leaving (has left) ESS"

Network Ports				Diagnostics
Device	Status	Mb/s	IP Addr.	MAC Addr.
Ethernet	Up	100.0	169.254.14.1	0040965c07fd
AP Radio	Up	11.0	169.254.14.1	0040965c07fd

Fig. 11.3 Configuration CISCO, home page


On peut y découvrir entre autre la liste des événements dernièrement générés ainsi que l'état des ports. Pour accéder à la page de configuration, cliquez sur le lien *Setup* dans le menu gris en haut de la page. Vous allez arriver sur la page de Setup suivante, figure 11.4.

AP350-5c07fd Setup

Cisco 350 Series AP 12.04

Home Map Network Associations Setup Logs Help

Uptime: 06:48:25



Express Setup			
Associations			
Display Defaults	Port Assignments	Advanced	
Address Filters	Protocol Filters	VLAN	Service Sets

Event Log		
Display Defaults	Event Handling	Notifications

Services			
Console/Telnet	Boot Server	Routing	Name Server
Time Server	FTP	Web Server	SNMP
Cisco Services	Security	Accounting	Proxy Mobile IP

Network Ports					Diagnostics
Ethernet	Identification	Hardware	Filters	Advanced	
AP Radio	Identification	Hardware	Filters	Advanced	

Fig. 11.4 Configuration CISCO, Setup page

Cette page permet d'accéder à toutes les options de configuration d'un point particulier de l'AP. La page de configuration qui nous intéresse se nomme *Security*, dans la rubrique *Service*. Une fois cliqué sur le lien, vous allez arriver sur la page qui regroupe les différentes stratégies de sécurité comme l'authentification, les VLAN ou encore l'encryption et le moyen d'association. Il est

également possible d'accéder à la page de gestion des administrateurs de l'AP pour créer différents comptes. La figure 11.5 illustre cette page.



Fig. 11.5 Configuration CISCO, Security Setup page

A partir de là, deux liens nous intéressent. *Authenticator Serveur*, pour la gestion du serveur d'authentification, et *Radio Data Encryption(WEP)* pour l'activation de WEP et de EAP. Une capture d'écran de cette page est montrée par la figure 11.6. Cette configuration se fait en deux étapes. Tout d'abord il faut inscrire la clé, sa longueur et la méthode d'association à l'AP grâce aux cases à cocher Open et Require EAP comme le montre la figure 11.6. Puis appuyez sur OK.

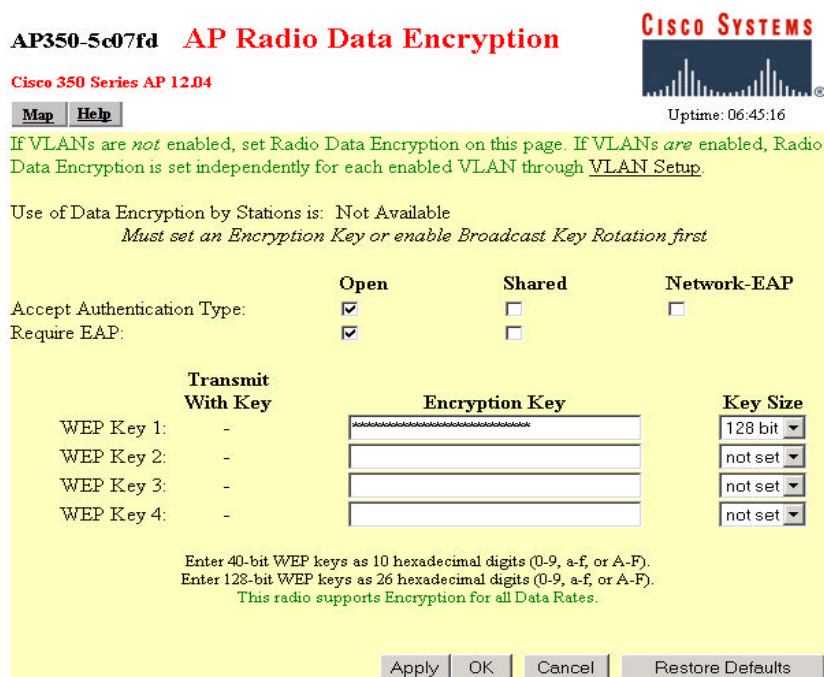
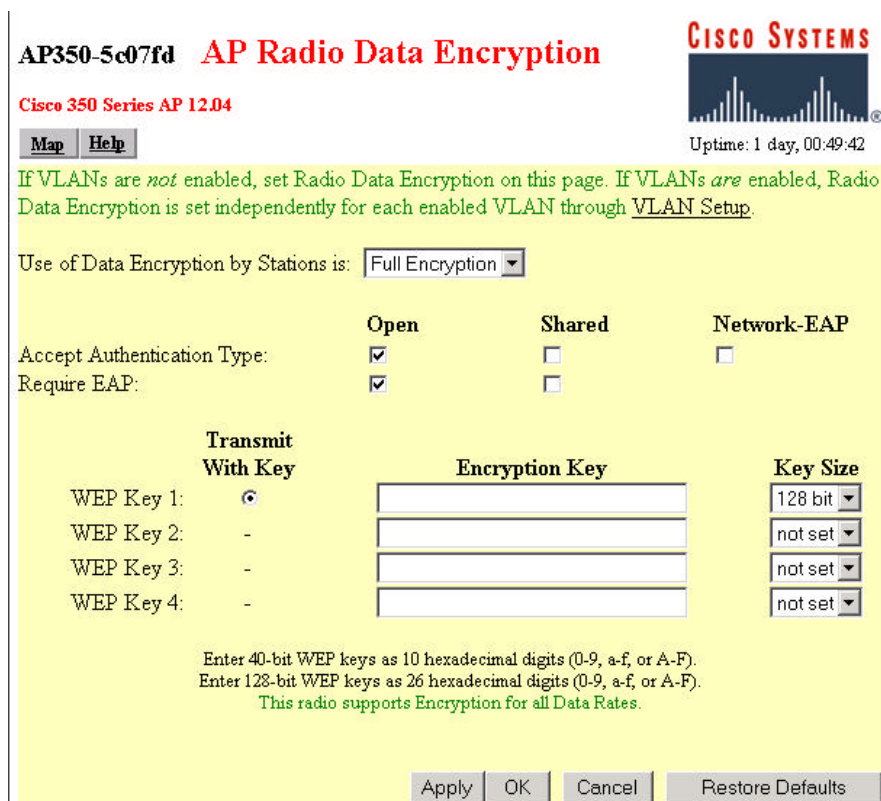


Fig. 11.6 Configuration CISCO, Radio Data Encryption page

Lorsque vous allez retourner sur la même page, elle va ressembler à la figure 11.7. Il est possible depuis là de préciser si l'association préalable avec la clé WEP est optionnelle ou obligatoire (Full Encryption). Car il faut garder en tête, qu'avant que l'échange EAP puisse s'effectuer, il faut d'abord que le poste de l'utilisateur s'associe avec l'AP. Et cette association peut être ouverte, par secret partagé ou encore par filtrage MAC. Et c'est une fois seulement l'association réalisée, que l'on requiert ou non EAP suivant l'état de la case à cocher.



AP350-5c07fd AP Radio Data Encryption

Cisco 350 Series AP 12.04

Uptime: 1 day, 00:49:42

If VLANs are *not* enabled, set Radio Data Encryption on this page. If VLANs *are* enabled, Radio Data Encryption is set independently for each enabled VLAN through [VLAN Setup](#).

Use of Data Encryption by Stations is: **Full Encryption**

Accept Authentication Type: ☒ Open ☐ Shared ☐ Network-EAP

Require EAP: ☒ Open ☐ Shared ☐ Network-EAP

	Transmit With Key	Encryption Key	Key Size
WEP Key 1:	<input checked="" type="radio"/>		128 bit
WEP Key 2:	<input type="radio"/>		not set
WEP Key 3:	<input type="radio"/>		not set
WEP Key 4:	<input type="radio"/>		not set

Enter 40-bit WEP keys as 10 hexadecimal digits (0-9, a-f, or A-F).
Enter 128-bit WEP keys as 26 hexadecimal digits (0-9, a-f, or A-F).
This radio supports Encryption for all Data Rates.

Apply OK Cancel Restore Defaults

Fig. 11.7 Configuration CISCO, Radio Data Encryption page 2

Il faut encore s'assurer que la clé que vous avez entrée précédemment soit considérée comme la *Transmit Key* en contrôlant l'état du bouton radio. Appuyez sur OK.

Finalement, il ne reste plus qu'à configurer le serveur d'authentification depuis la page représentée par la figure 11.8. Il est possible d'activer jusqu'à 4 serveurs à la fois. Avant de commencer, il faut s'assurer de la version du draft utilisée. Il s'agit de la version du protocole EAP. Elle est définie en fonction de votre matériel. Pour déclarer un serveur, il suffit d'entrer l'adresse IP, le type de serveur (les serveurs TACAS sont également compatibles), le numéro du port (le même que celui spécifié dans le fichier radiusd.conf) et le secret partagé (celui inscrit dans le fichier clients.conf). Cochez la case EAP Authentication en dessous de la configuration du serveur. Les autres options n'ont en principe pas besoin d'être modifiées, mais vous pouvez ajuster certaines caractéristiques de retransmission des requêtes. Appuyez sur OK.

AP350-5c07fd Authenticator Configuration

Cisco 350 Series AP 12.04

802.1X Protocol Version (for EAP Authentication): 802.1x-2001

Primary Server Reattempt Period (Min): 0

Server Name/IP	Server Type	Port	Shared Secret	Retran Int (sec)	Max Retran
10.192.57.164	RADIUS	1812	*****	5	3
	RADIUS	1812	*****	5	3
	RADIUS	1812	*****	5	3
	RADIUS	1812	*****	5	3

Use server for: ☒ EAP Authentication ☐ MAC Address Authentication ☐ User Authentication ☐ MIP Authentication

Note: For each authentication function, the most recently used server is shown in green text.

Apply OK Cancel Restore Defaults

Fig. 11.8 Configuration CISCO (Authenticator Configuration)

A ce stade, l'AP est prêt à collaborer avec les utilisateurs et le serveur freeRADIUS. Il est conseillé de s'intéresser à la documentation CISCO [26] pour les détails complémentaires. La dernière étape de l'installation de notre plateforme d'authentification consiste à configurer les postes des utilisateurs afin qu'ils se connectent à l'AP par le protocole EAP.

11.5 Côté utilisateur

C'est la dernière étape. Pour qu'un utilisateur puisse accéder au réseau, il doit s'associer avec l'AP, puis lui transmettre ces données d'authentification par l'intermédiaire d'EAP. Evidemment, le dialogue EAP ne se fait pas tout seul. L'utilisateur doit posséder la configuration et un logiciel client qui gère ce dialogue. Plusieurs solutions sont envisageables. Soit par le service offert par windows, soit par un logiciel client installé sur votre poste fourni par une entreprise de développement ou un constructeur.

Logiciel client

Commençons par la solution retenue dans ce travail, c'est-à-dire AEGIS client développé par MeetingHouse [27]. J'ai téléchargé la version 2.1.0 du client AEGIS (AEGIS_Client_v2.1.0.exe). Il s'agit d'une version d'évaluation qui dure 15 jours. L'installation est très simple sous windows grâce au fichier exécutable. Il existe une version sous linux qui est un peu plus délicate et subtile. N'ayant pas de PC muni d'une carte wireless PCI compatible linux, je n'ai pas testé le client Linux. Contrairement à l'AP qui doit être compatible 802.1x, dans le cas de la carte du client, aucun matériel particulier n'est requis. Votre adaptateur 802.11b ou g acheté il y a quelques mois, fera l'affaire. Passons à la marche à suivre de l'installation.

Inutile d'expliquer comment installer le client à partir de l'exécutable windows. Tout est par défaut, pas de particularités, il suffit de suivre l'assistant d'installation. Une fois l'ordinateur redémarré, le client devrait se lancer automatiquement et un petit logo AEGIS est présent dans la

barre des tâches en bas à droite. Ce n'est pas lors du login de l'utilisateur sur la machine que l'échange EAP se réalise, mais lorsque l'accès au réseau est requis, juste après l'association avec l'AP. Dès que vous êtes associé, si le client est lancé et la carte configurée pour utiliser ce service, l'AP va vous envoyer la trame *Request Identity*. Pour configurer la carte avec le client AEGIS, allez dans les propriétés réseau de l'adaptateur 802.11. En principe, vous devriez voir dans la liste des composants un nouveau service disponible, *AEGIS Protocol*, comme le montre la figure 11.9. Il faut bien évidemment cocher cette case.

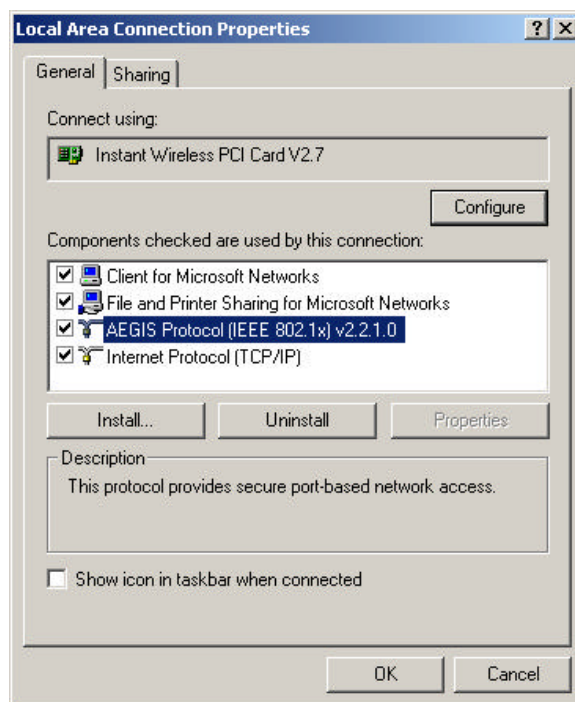


Fig. 11.9 Configuration service AEGIS

Le fait d'avoir cette case activée permet de voir l'adaptateur dans la liste des cartes utilisables par le client AEGIS. Pour ouvrir la console du client (qui est déjà lancée), il faut simplement double-cliquer sur l'icône AEGIS dans la barre des tâches. La figure 11.10 montre la fenêtre de base du client avec l'adaptateur sélectionné.

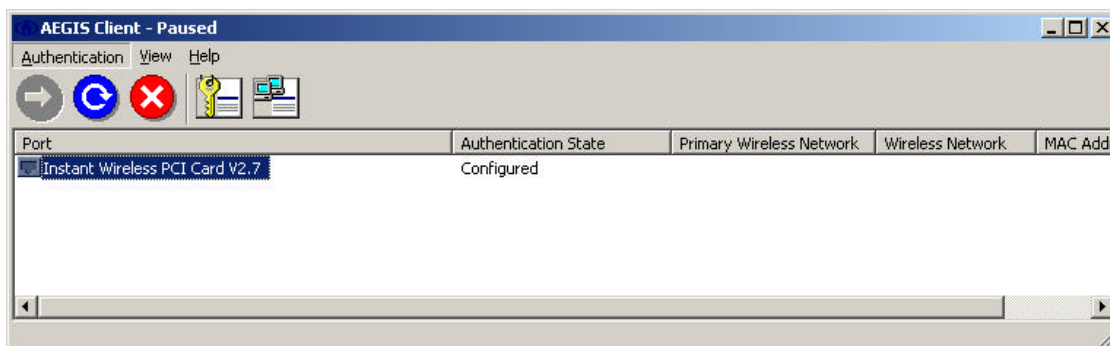


Fig. 11.10 Client AEGIS

A ce stade, il ne reste plus qu'à configurer le profil d'authentification et le profil réseau. Pour le premier, utilisez le bouton avec la petite clé dans la barre d'outils, ou le menu *Authentication* → *Configure*. La fenêtre qui apparaît est représentée par la figure 11.11. Nous allons en créer un seul, celui pour l'accès au BSS *tsunami* (AP CISCO). Le nom du profil est *tsunami_prof*. Comme nous l'avons dit pendant la configuration du serveur freeRADIUS, c'est un challenge MD5 qui réalise l'authentification. Donc, le type d'authentification sélectionné pour le profil est MD5-Challenge comme le montre la figure ci-dessous. Pour l'identité, il suffit de mettre son nom de login utilisateur et le mot de passe enregistré dans la base de données radius.

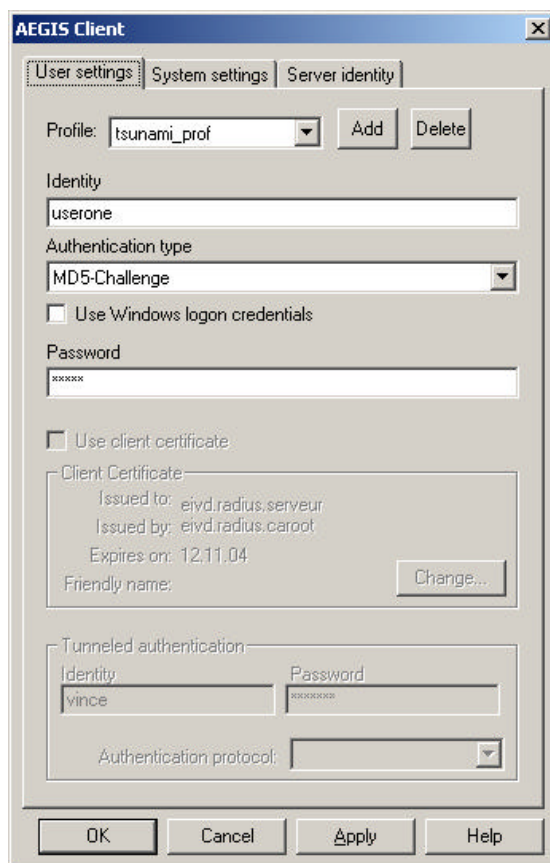


Fig. 11.11 Configuration du profil d'authentification AEGIS

Les autres onglets n'ont pas besoin d'être modifiés, car il s'agit de paramètres pour d'autres méthodes d'authentification. Le nombre et le type de méthodes supportées dépend du client. Avec celui-ci, il est également possible d'utiliser TLS, LEAP, TTLS et PEAP. Cliquez sur OK. Passons maintenant à la configuration du profil réseau. Il faut d'abord sélectionner l'adaptateur à l'aide du curseur. Ceci a pour effet d'activer le bouton avec les deux petits ordinateurs dans la barre d'outils. Quand vous cliquez dessus, la fenêtre représentée par la figure 11.12 apparaît. La liste des BSSID disponibles s'affiche dans la zone *Available Network*.

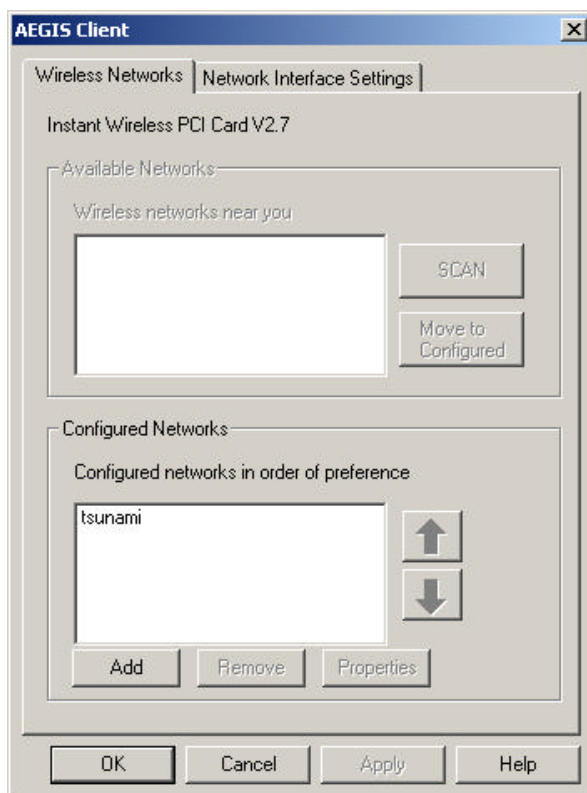


Fig. 11.12 Configuration profil réseau AEGIS

Ajoutez une configuration de réseau. Cela va ouvrir la nouvelle fenêtre ci-dessous, illustrée par la figure 11.13. Inscrivez le nom du profil réseau (*tsunami*), ainsi que le nom du profil d'authentification, (*tsunami_prof*). N'oubliez pas de décocher *Association with any available network*.

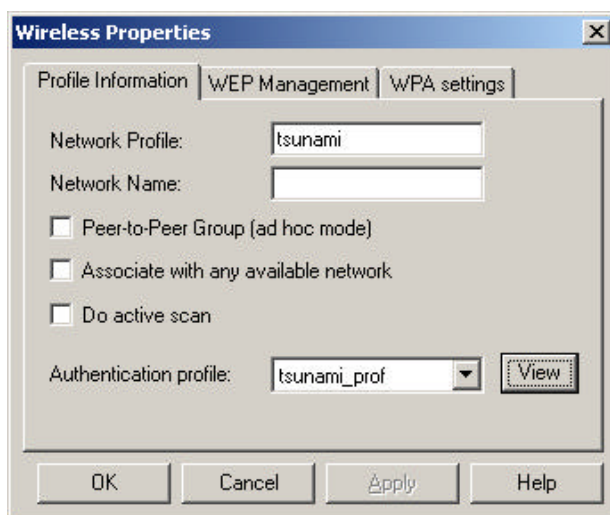


Fig. 11.13 Configuration profil réseau AEGIS 2

Dans l'onglet *WEP Management*, vous devez rentrer la clé d'encryption WEP inscrite dans l'AP CISCO à la page *AP Radio Data Encryption*, et cocher les mêmes cases que montre la figure 11.14.

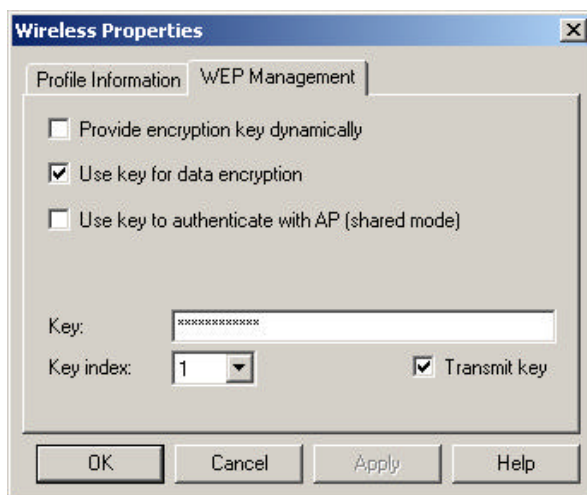


Fig. 11.14 Configuration profil réseau AEGIS 3

Après avoir redémarré le client AEGIS comme il vous l'est demandé (simple clique sur la croix rouge dans la bar d'outils), vous êtes en principe prêt à vous connecter au serveur d'authentification. Cliquez sur la flèche verte et attendez. L'échange EAP doit démarrer, et vous serez averti lorsque l'authentification a réussi. Quand vous êtes connecté, le petit icône AEGIS dans la barre des tâches devient vert. Voilà, vous venez de réaliser votre première authentification avec EAP.

Il existe plein d'autres clients 802.1x développés par d'autres entreprises. Citons *Odyssey* de Funk Software [28], que nous allons employer plus bas lors des mesures. L'installation est aussi simple que AEGIS. En plus de permettre l'accès à un réseau par différentes méthodes d'authentification, ce client peut paramétrer une carte sans fil afin d'activer les mécanismes de sécurité proposés par l'adaptateur (WEP, TKIP, AES). Bien qu'il existe AEGIS pour linux, un autre client open source cette fois-ci, nommé XSupplicant, est disponible sur le site d'open1x project [29] et permet autant de fonctionnalités que les autres clients venant d'être cités.

Service offert par Windows

L'autre solution pour se connecter à un réseau 802.1x, est d'employer le service fourni directement avec Windows. Pour avoir accès à cette option, vous devez soit avoir windows 2000 avec le service pack 4, ou windows XP. Si le service est actif, vous devez avoir, dans la fenêtre des propriétés de la carte réseau, un onglet *Authentication*, comme le montre la figure 11.15.

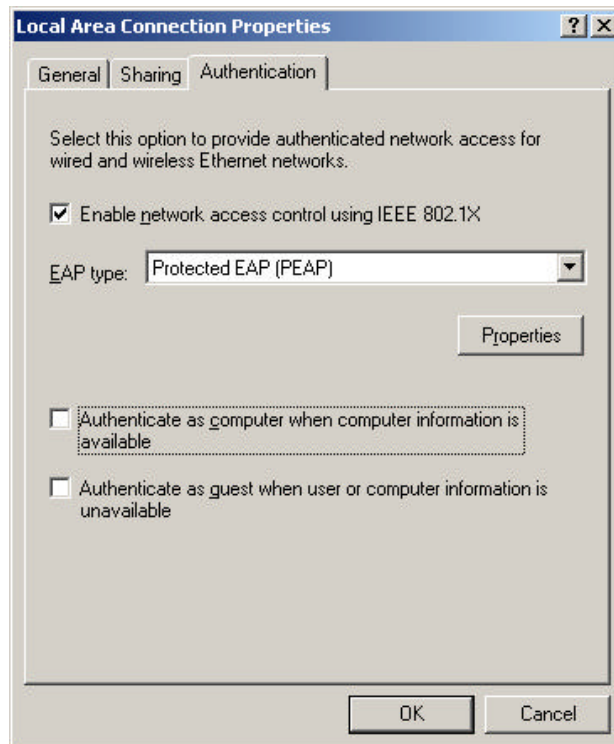


Fig. 11.15 Configuration 802.1x windows

Si ce n'est pas le cas, c'est que le service n'a pas démarré. Allez alors dans le gestionnaire de service (panneau de configuration) et activez *Wireless Configuration* comme le montre la figure 11.16. Assurez-vous qu'il va dorénavant démarrer automatiquement. Là aussi plusieurs moyens d'authentification sont disponibles. Cela dépend de la version de windows. Pour 2000 professionnelle, PEAP et SmartCard sont possibles. Pour XP c'est les mêmes, avec en plus le challenge MD5.

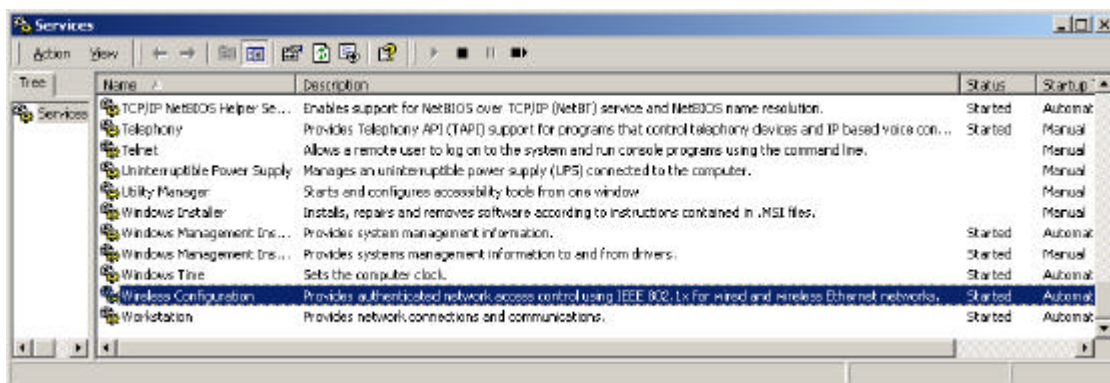


Fig. 11.16 Configuration 802.1x windows 2

Je n'ai pas exploré les possibilités du client offert par windows. Pour les détails supplémentaires veuillez vous référer à la documentation de Microsoft [30].

Nous arrivons au terme de l'installation de la plateforme d'authentification. Il s'agit maintenant d'activer un mécanisme d'encryption capable de protéger l'échange EAP qui circule en clair pour l'instant ! Nous allons activer WEP, TKIP, AES et IPsec.

11.6 Mécanismes d'encryption

Une fois la plateforme d'authentification opérationnelle, l'activation et la configuration des mécanismes d'encryption est très facile et prend beaucoup moins de temps. Généralement, ces configurations se résument, la plus part du temps, à quelques options à remplir par l'intermédiaire d'une interface WEB pour les AP et d'un logiciel client pour les cartes. Dans le cas de l'AP ASUS, il n'y a pas d'interface WEB, mais un manager fourni avec le matériel (ASUS AP Manager 1.0.0.10), qui s'installe sur un poste servant de centre de configuration. Par contre, pour l'AP CISCO et Buffalo, un petit site WEB implanté directement dans l'AP permet d'effectuer ces configurations. Pour le manager des cartes, chaque constructeur fournit un petit utilitaire permettant d'effectuer les paramétrages de base. Mais ce n'est pas toujours suffisant. Dans le cas de la carte Buffalo par exemple, pour pouvoir exploiter TKIP et AES il faut disposer d'un autre manager [29].

Activation de WEP

Avant de paramétrer WEP, il faut déterminer la longueur de la clé que vous voulez utiliser. Il y a en général quatre clés à inscrire. Cela permet d'effectuer des changements de temps à autre sans devoir les inscrire à chaque fois. Une simple sélection du numéro de la clé suffit. Ensuite, il faut choisir vos clés. Deux formats existent. Sous la forme ASCII, de 8 à 64 caractères, ou sous la forme hexadécimal, de 10 ou 26 caractères. Ces clés doivent être choisies au hasard et sans suites logiques. Si c'est la forme ASCII qui est employée, évitez des mots classiques, les surnoms, etc. Sinon, il existe un générateur de dés automatique se basant sur une paraphrase qui permet de créer un jeu de quatre clés. Là aussi, attention au choix de la paraphrase.

Nous avons déjà indirectement activé WEP lors de la configuration de la plateforme d'authentification. Cela c'est fait en deux étapes. Tout d'abord la configuration du point d'accès par l'intermédiaire de l'interface WEB de l'AP CISCO (figure 11.6 et 11.7), puis grâce au client Meetinghouse (figure 11.14). De manière générale, c'est d'abord l'AP qui est configuré, puis les clients qui s'y connectent. Je ne vais pas détailler et faire une copie d'écran de tous les clients et toutes les interfaces WEB utilisés dans ce projet, car il y en a plusieurs, un pour chaque type de matériel. Celui de Syslink, de D-Link, d'AVAYA, d'ASUS et celui de Buffalo. Pour avoir accès aux détails de configuration, il faut se référer à la documentation fournie avec le matériel.

Activation de TKIP et d'AES

Dans mon cas c'est la méthode TKIP-PSK (Pre-Shared Key) qui est employée. Cette clé partagée est déterminée à l'aide d'une paraphrase de 8 à 63 caractères ASCII ou de 64 caractères hexadécimaux. La même paraphrase est utilisée pour créer la clé AES. La figure 11.17 représente la page de configuration des paramètres de sécurité de l'AP Buffalo. On voit qu'il suffit de sélectionner un bouton radio et d'inscrire les clés ou la paraphrase. Il est même possible de paramétrer le nombre de secondes entre chaque changement de clé temporel. Il faut noter au passage que Buffalo est l'un des premiers fabricants à proposer du matériel de ce type, capable d'intégrer TKIP et AES associé à CCMP. Si l'on fait référence à la norme 802.11i, il s'agit d'un TSN, Transition Security Network (Voir chapitre 9).

[Return to TOP](#)

LAN settings

- Wireless
- Wireless security**
- LAN port
- DHCP server
- Wireless LAN Computer Limitation
- Wireless bridge(WDS)

Wireless security settings

Broadcast SSID ☒ Enabled ☐ Disabled

Data encryption ☒ Disabled

☐ WEP

WEP key 1: HEX
 2: HEX
 3: HEX
 4: HEX

☐ TKIP ☐ WPA-PSK (Pre-Shared Key)

☐ AES

WPA Group Rekey Interval 120 Sec

IEEE802.1x/EAP authentication (WPA) ☒ Do not authorize ☐ Authorize

RADIUS Authentication

RADIUS Server
RADIUS Port 1812
RADIUS Key

Fig. 11.17 Page des paramètres de sécurité de l'AP Buffalo

Evidemment, cet AP est compatible 802.1x et peut être paramétré pour dialoguer avec un serveur RADIUS et pour implémenter EAP. Donc, à partir de cette page, il est possible de gérer tous les paramètres de sécurité de l'AP nécessaires aux mesures.

Du côté du client, pour que la carte de l'utilisateur puisse se connecter à l'AP configuré avec TKIP ou AES, il faut pouvoir paramétrer les propriétés de l'adaptateur. Comme déjà dit, pour exploiter les capacités de la carte Buffalo, il a fallu télécharger le manager Odyssey [29], ainsi que la version 3.30.15.1 des pilotes, disponibles sur le site de Buffalo [31]. Une fois Odyssey installé, un composant réseau s'ajoute dans les propriétés de votre adaptateur, de la même manière que pour le client AEGIS. La figure 11.18 montre ce nouveau composant.

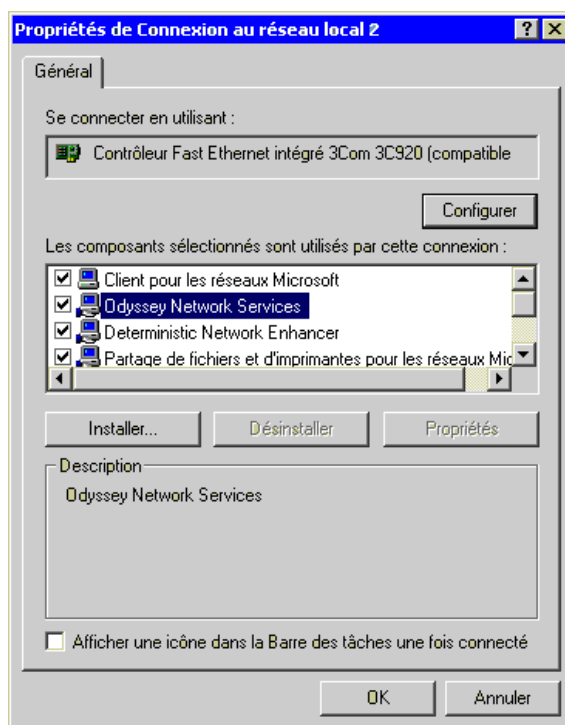


Fig. 11.18 Propriété de la carte après l'installation du client Odyssey

Dorénavant, le client va démarrer automatiquement dès que vous activez l'adaptateur. Un petit icône reste visible dans la barre des tâches. La figure 11.19 représente la fenêtre de configuration des paramètres de sécurité de la carte .

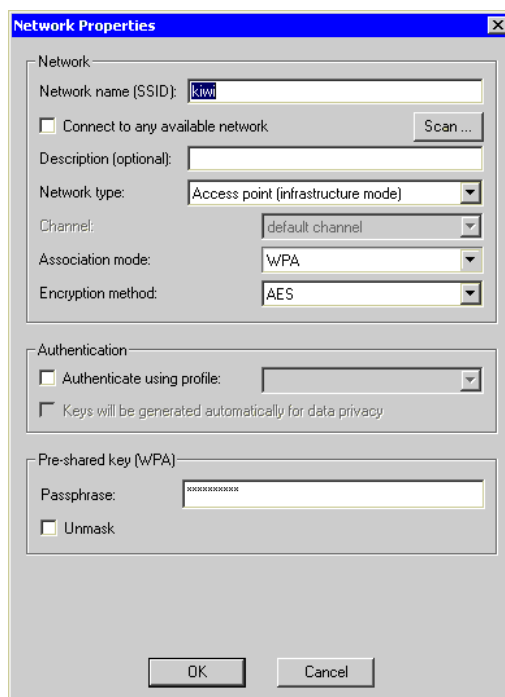


Fig. 11.19 Configuration du mode de sécurité de la carte Buffalo depuis le client Odyssey

On y voit le nom du SSID, le type de réseau, le mode d'association (open, shared ou WPA) et la méthode d'encryption (WEP, TKIP ou AES). Dans la zone de texte située en bas de la fenêtre, il y a la possibilité d'entrer la paraphrase nécessaire à TKIP et AES. Finalement, pour se connecter à l'AP Buffalo, il faut ouvrir la fenêtre de connexion comme le montre la figure 11.20.

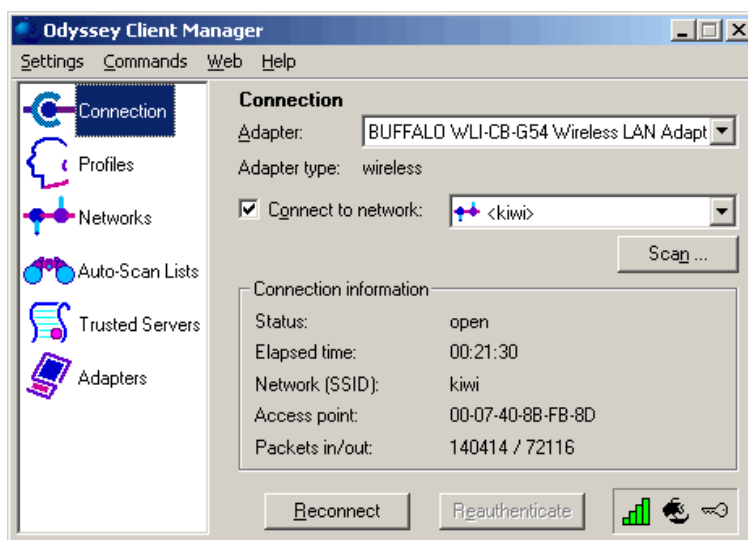


Fig. 11.20 Connexion au réseau depuis le client Odyssey

Il suffit de sélectionner l'adaptateur, le nom du SSID que vous venez de taper dans la fenêtre précédente et c'est tout. La petite clé située en bas à droite permet d'indiquer le mode d'encryption. Voilà pour les méthodes d'encryption du niveau liaison. Passons à une méthode de niveau supérieur. Comme discuté plus haut, la possibilité de mettre en œuvre IPsec sur un réseau 802.11 est tout à fait envisageable.

11.7 IPsec

La configuration est exactement la même que pour un réseau câblé. Sous windows, IPsec est directement implémenté. Il suffit d'activer le service IPsec depuis le panneau de configuration, comme pour l'authentification (figure 11.16). De cette manière, vous aurez la possibilité de créer des polices d'accès. Pour lancer l'utilitaire qui permet de gérer ces polices, il suffit d'aller dans les outils d'administration du panneau de configuration et d'ouvrir l'utilitaire *Local Security Policy*. La figure 11.21 illustre la fenêtre des polices d'accès.

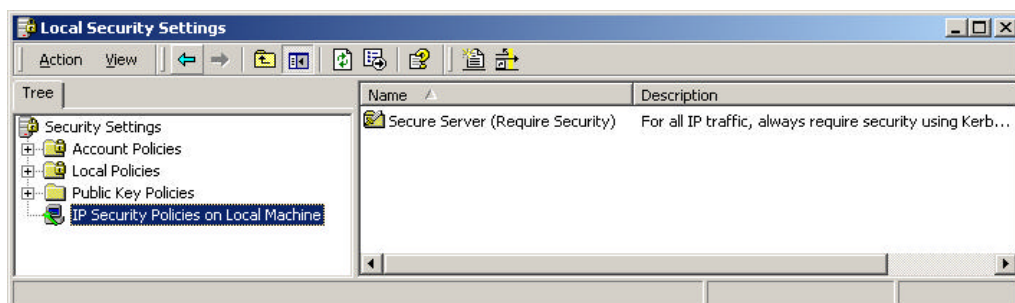


Fig. 11.21 Polices d'accès

Dans ce cas, la police se nomme *Secure Server*. Ces polices d'accès permettent de définir des règles qui caractérisent un tunnel particulier. On y définit les deux extrémités du tunnel, quels types de trafic doivent être cryptés, la méthode d'authentification et d'autres paramètres. La figure 11.22 montre la fenêtre des règles de la police *Secure Server*. Seule la règle *All IP traffic* est active. Elle est configurée pour crypter tout le trafic entre deux machines définies.

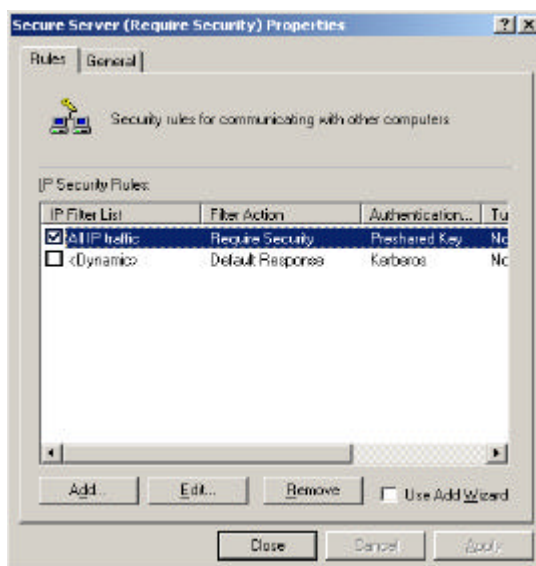


Fig. 11.22 Règles IPsec

Créer une règle n'est pas très compliqué. Si vous cochez la case *Use Add Wizard*, un assistant est là pour vous aider dans le processus. Sinon vous pouvez le faire manuellement. Cette procédure n'est pas décrite ici, mais tous les détails et la marche à suivre se trouvent dans le document [32].

Une fois les règles bien définies, il faut associer la police à une interface réseau. Pour cela, il faut aller dans les propriétés TCP/IP de l'adaptateur, puis dans les paramètres avancés. Dans l'onglet option, il y a les propriétés d'IPsec, comme le montre la figure 11.23.

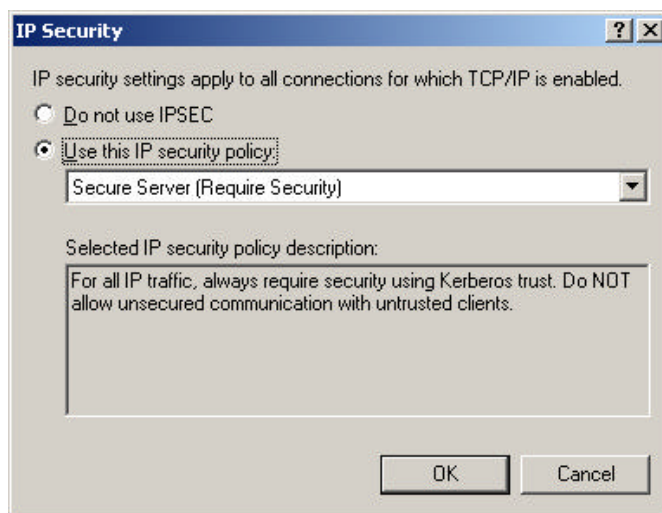


Fig. 11.23 Propriétés IPsec

C'est tout. Pour vérifier que cela fonctionne, faites un ping entre deux ordinateurs où IPsec est activé. Le résultat se présente comme ci-dessous. La négociation IPsec, phase IKE et AH, a lieu lors des quatre premiers ping. Ensuite, la connexion est réalisée et les requêtes obtiennent réponses.

```
C:\>ping 169.254.14.16
Envoi d'une requête 'ping' sur 169.254.14.16 avec 32 octets de données :
```

```
Négociation de la sécurité IP.
Négociation de la sécurité IP.
Négociation de la sécurité IP.
Négociation de la sécurité IP.
```

```
Statistiques Ping pour 169.254.14.16:
    Paquets : envoyés = 4, reçus = 0, perdus = 4 (perte 100%),
Durée approximative des boucles en millisecondes :
    minimum = 0ms, maximum = 0ms, moyenne = 0ms
```

```
C:\>ping 169.254.14.16
Envoi d'une requête 'ping' sur 169.254.14.16 avec 32 octets de données :
```

```
Réponse de 169.254.14.16 : octets=32 temps=10 ms TTL=128
Réponse de 169.254.14.16 : octets=32 temps=10 ms TTL=128
Réponse de 169.254.14.16 : octets=32 temps<10 ms TTL=128
Réponse de 169.254.14.16 : octets=32 temps<10 ms TTL=128
```

```
Statistiques Ping pour 169.254.14.16:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
```

Durée approximative des boucles en millisecondes :
minimum = 0ms, maximum = 10ms, moyenne = 5ms

Pour constater que l'encryption fonctionne, faites une capture avec Ethereal et vous devriez obtenir des trames ESP.

Voilà pour windows. Maintenant, il serait bien d'implémenter IPsec sous Linux afin de couvrir une palette plus large de clients. Il y a bien évidemment la possibilité de le faire. La version open source de Linux se nomme *Freeswan*. Toutes les informations relatives à ce paquetage sont disponibles sur le site officiel de ce projet [33]. Pour mettre en œuvre ce protocole, il suffit donc d'installer un paquetage et d'attaquer quelques fichiers de configuration. Le plus important de ces fichiers se nomme *ipsec.conf* et se trouve en général dans le répertoire */etc*. Il permet de configurer les règles des polices d'accès. J'ai pu mettre en œuvre ce protocole entre deux machines Linux. Un site [34] regroupe l'ensemble des documents, marches à suivre, etc. L'implémentation d'IPsec fonctionne sous la forme d'un service qui doit démarrer automatiquement, ou manuellement à l'aide de la commande *ipsec setup start*. Ce service s'appuie sur un module (daemon) qui doit obligatoirement tourner en arrière plan et qui permet de gérer la création et la configuration des tunnels IPsec. Ce daemon s'appelle *pluto*.

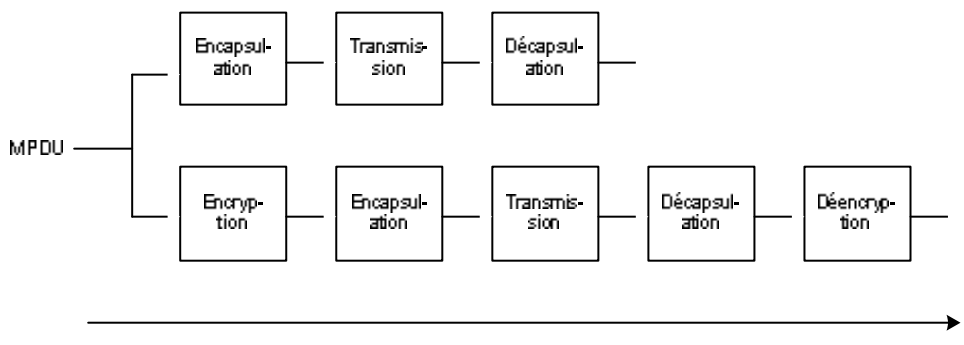
Pour faire fonctionner IPsec entre une machine Linux et Windows, c'est une autre paire de manches. Il faut installer sous Windows un petit utilitaire (*ipsec tools*) capable de dialoguer avec *freeswan*, grâce au même fichier de configuration *ipsec.conf*. Une marche à suivre [35] décrit la procédure de manière détaillée. Le problème majeur est qu'il faut implémenter les certificats, chose relativement complexe. Pour avoir un peu d'aide, il faut se référer au site [36] qui est consacré aux certificats X.509 sous Linux et plus particulièrement leurs implémentations avec *freeswan*.

Voilà, l'installation de la plateforme est terminée. Tout ce qui a été vu et mis en œuvre lors de ce travail a été décrit dans ce chapitre. Nous disposons donc d'une plateforme opérationnel, capable de réaliser l'authentification et l'encryption des données à plusieurs niveaux.

12 Mesures des performances

Un des buts de ce travail de diplôme est de déterminer les coûts au niveau des performances, que produit l'implémentation d'une plateforme de sécurité. La phase d'authentification, qui comporte l'échange des paquets EAP et RADIUS, n'entre pas en compte dans les baisses de performances. En effet, seulement quelques paquets sont transmis lors de cette phase. Ces trames ne sont qu'une goutte d'eau dans la masse de trafic. De plus, une fois connecté, l'authentification est terminée et ne se reproduira plus, à moins qu'un mécanisme d'authentification périodique soit mis en place. Mais même dans ce cas, cela reste insignifiant.

Par contre, l'encryption des trames contribue largement aux baisses de performances. Le fait de passer les données à la moulinette, de calculer et de générer des clés de session, prend du temps non négligeable. Le paramètre intéressant est le débit en bits par seconde. Il est calculé en mesurant le délai. On considère comme délai, le temps que met le paquet dès qu'il est transmis par l'application aux couches inférieures, jusqu'au moment où, chez le destinataire, ce même paquet est transmis à la couche applicative. De là, on peut déterminer le débit en divisant la quantité effective de données transmises en bit, par le délai mesuré en seconde. De manière très simple et qualitative, on peut se rendre compte du temps supplémentaire que prend l'encryption pour traiter l'information, comme le montre la figure 12.1.



les applications en générant exactement le même type de trafic. Il est dès lors facile grâce au générateur de choisir exactement la quantité de données, le type et la quantité des paquets à transmettre. De l'autre côté, à la réception, on connaît exactement ces paramètres d'émission et il est alors facile de comparer le trafic reçu avec celui généré pour tirer les conclusions.

Pour le problème de la synchronisation des machines, on peut utiliser le protocole NTP (Network Time Protocol), défini par le RFC 1305 [37]. NTP consiste en une application client/serveur, où l'horloge de référence est stockée sur le serveur, que les clients interrogent de manière régulière. Ensuite, on mesure l'heure à laquelle est parti le paquet de la source, on enregistre celle à laquelle il arrive au destinataire, puis par une simple soustraction, on obtient le délai. Un simple analyseur de réseau comme Ethereal peut faire l'affaire. C'est fiable, mais il faut être rigoureux. J'ai essayé d'implémenter cette solution, mais je me suis vite aperçu qu'il était très difficile de synchroniser les machines à la milliseconde près, précision obligatoire pour ce genre de mesures. La documentation [38] décrit l'installation et la configuration de NTP sous Linux, et le lien [39] permet de télécharger le client pour windows, *automachron*.

Une deuxième manière consiste à éviter la synchronisation en mesurant le délai sur la même machine. Si l'horloge qui mesure l'heure d'émission est la même que celle servant à la mesure du délai à la réception, aucun problème pour déterminer le temps écoulé. C'est précis et juste. Cela entraîne une machine possédant deux interfaces, l'une pour émettre et l'autre pour recevoir.

On peut encore imaginer une autre manière pour éviter la synchronisation en passant par le calcul du décalage de temps entre les horloges des machines. Si l'on sait que l'horloge de la machine qui réceptionne les données possède un offset bien précis par rapport à l'horloge d'émission, il est alors facile de tenir compte de ce décalage lors des calculs du délai. La mesure de cet offset n'est pas forcément facile, comme avec NTP, il faut être rigoureux, faire plusieurs mesures. Sans parler qu'il faut tenir compte de la dérive des deux horloges.

12.2 Plateforme

La solution retenue est réalisée à l'aide d'un logiciel nommé *Chariot* [41], de l'entreprise *NetIQ*. C'est une méthode active, qui se base, pour le calcul du débit, sur le temps de réponse. C'est-à-dire le temps que met le paquet pour traverser le réseau et pour revenir. C'est donc la même horloge qui mesure le délai, mais une seule interface est requise. Ce logiciel coûtant relativement cher, une version d'évaluation a été téléchargée depuis le site officiel de l'entreprise [40]. La figure 12.2 décrit le schéma de mesure. Pour les détails sur le rôle des machines, voir le chapitre 12.3.

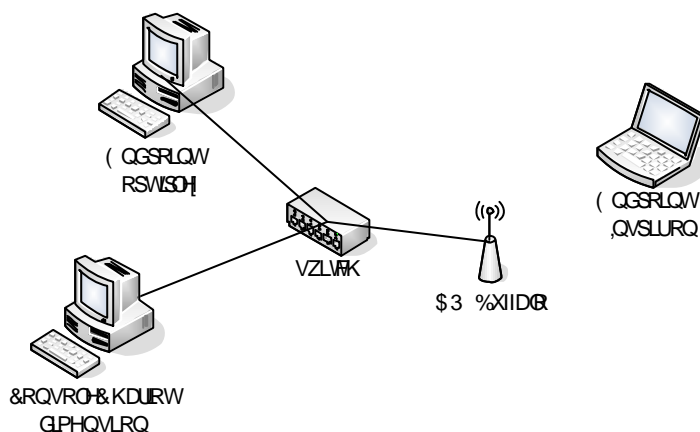


Fig. 12.2 Plateforme de mesure avec Chariot 4.3

Il faut noter encore que les mesures ont été effectuées sur un petit sous réseau privé (169.254.0.0) conçu à cet effet. De cette façon, aucun trafic extérieur ne vient perturber la mesure. Afin de pouvoir mesurer les performances d'AES et de TKIP, c'est l'AP Buffalo qui est employé et la carte Buffalo. Il faut savoir que les pilotes et le client fourni avec le matériel Buffalo ne permet pas d'activer TKIP ni AES. Donc il a fallu mettre à jour les pilotes de la carte et employer un autre client. Il s'agit de Odyssey Client Manager 2.22 de Funk Software [29]. Là aussi, c'est une version d'évaluation. Notons au passage que ce client permet également l'authentification 802.1x au même titre que le client MeetingHouse vu précédemment. Pour l'installation et la configuration d'Odyssey Client Manager et des pilotes, se référer au chapitre 11.5. Pour configurer l'AP, une interface WEB est disponible. Voir également le chapitre 11.5.

12.3 Logiciel de mesure

Chariot fonctionne sur la base de plusieurs éléments, les *endpoints* (client), les scripts et la console. Les *endpoints* sont des processus (thread) qui tournent sur chaque machine impliquée dans le test et qui génèrent le trafic. Les scripts sont exécutés par ces *endpoints*, et décrivent le type de trafic à générer, le débit à respecter, la taille des paquets, etc. La console sert à distribuer les scripts et à récupérer les informations des mesures que génèrent les *endpoints*. Dans mon cas, seulement deux *endpoints* sont activés et une console pour les contrôler. Le premier génère le trafic en direction du deuxième, qui confirme alors la bonne réception en renvoyant un acquittement au premier qui s'occupe alors de collecter les informations. Notons que le générateur de trafic n'est pas une machine sans fil relié à l'AP par les airs. De cette manière, le débit n'est pas divisé par deux. Car, comme la configuration de la plateforme est une infrastructure gérée par un AP, tous les paquets échangés doivent passer par le coordinateur et occupent donc deux fois le média avant d'arriver chez le destinataire. La figure 12.4 décrit les scripts exécutés par chacune des extrémités de la mesure. Il faut interpréter les lignes en minuscule comme paramètres de la fonction en majuscule juste au-dessus. Par exemple, la fonction SLEEP ne prend qu'un seul paramètre, time, alors que la fonction SEND comprend quatre paramètres.

Line	Endpoint 1	Endpoint 2
1	SLEEP	
2	time = initial_delay (0)	
3	CONNECT_INITIATE	CONNECT_ACCEPT
4	port = source_port (AUTO)	port = destination_port (AUTO)
5	LOOP	LOOP
6	count = number_of_timing_records (1000)	count = number_of_timing_records (1000)
7	START_TIMER	
8	LOOP	LOOP
9	count = transactions_per_record (1)	count = transactions_per_record (1)
10	SEND	RECEIVE
11	size = file_size (100000)	size = file_size (100000)
12	buffer = send_buffer_size (DEFAULT)	buffer = receive_buffer_size (DEFAULT)
13	type = send_datatype (NOCOMPRESS)	
14	rate = send_data_rate (UNLIMITED)	
15	CONFIRM_REQUEST	CONFIRM_ACKNOWLEDGE
16	INCREMENT_TRANSACTION	
17	END_LOOP	END_LOOP
18	END_TIMER	
19	SLEEP	
20	time = transaction_delay (0)	
21	END_LOOP	END_LOOP

Fig. 12.4 Scripts de mesure

On remarque tout d'abord, qu'il y a un script pour le générateur et un pour le récepteur. On peut parler de programmation par rendez-vous. En effet, tant que le *endpoint2* n'a pas accepté la connexion (CONNECT_ACCEPT), le *endpoint1* ne va pas entrer dans la première boucle LOOP. De même que le *endpoint2* va stopper l'exécution de son script sur RECEIVE, tant qu'il n'a pas reçu les paquets émis (SEND) par le *endpoint1*. C'est la même chose pour CONFIRME_REQUEST. D'où la disposition des scripts l'un à côté de l'autre pour une meilleure visualisation. Il faut savoir qu'une connexion TCP est de toute façon établie entre les deux points de la mesure, afin de s'échanger la confirmation de la connexion ou la bonne réception des paquets. Le langage du script est très simple, seulement un dizaine de fonctions très vite comprises et bien documentées dans l'aide du logiciel. D'ailleurs, la lecture de ce petit script est relativement intuitive :

Une première connexion TCP est établie (sur un port bien précis choisi automatiquement ici) entre les deux intervenants afin de démarrer la mesure de manière claire et de pouvoir s'échanger les informations de gestion de la mesure (CONNECT_INITIATE et CONNECT_ACCEPT). Une première boucle (LOOP) compte le nombre d'acquisitions. Dans ce cas, 1000 mesures de temps vont être effectuées. Pour chaque mesure, on démarre le chronomètre (START_TIMER) et on entre dans une deuxième boucle interne (LOOP) qui compte le nombre de transactions (SEND) à effectuer pour une mesure. Dans ce cas, une seule transaction va être effectuée afin de transmettre 100000 bytes de données. Il est évident que lors de cette transaction, plusieurs paquets vont être envoyés. Dès que toutes les transactions ont été réalisées, le chronomètre est stoppé (END_TIMER) et une nouvelle mesure commence. Une fois toutes les mesures terminées, le rapport est envoyé vers la console qui affiche les résultats.

12.4 Types de trafic

Deux types de trafic vont être émuloés. Comme c'est le cas pour des applications de voix et de vidéo, du trafic UDP va être généré. De plus une connexion TCP va également être testée afin d'être un peu plus large et de considérer d'autres applications se basant sur ce protocole. N'ayant pu commencer les mesures seulement l'avant dernière semaine (depuis le 8 décembre), elles sont brèves et pas extrêmement approfondies. Il aurait fallu orienter ces mesures par rapport à des applications VoIP, en tenant compte des caractéristiques de différents codecs. Voir combien de communications simultanément, une fois cryptées, peut gérer l'AP. Malheureusement, le manque de temps a tranché et seulement deux types de trafic ont été testés. Néanmoins, ces quelques mesures expriment très bien la baisse des performances de manière générale et mettent en évidence les différences entre plusieurs mécanismes d'encryption.

12.5 Présentation des résultats

Je me suis contenté d'interpréter les graphiques directement générés par le logiciel de mesure. Il aurait mieux valu récupérer les données sous un format texte et de les traiter manuellement. Comme vous allez le voir, les graphes sont quelques peu bruités par le fait de la fluctuation du débit des réseaux 802.11. Le fait de traiter ces données manuellement aurait pu filtrer certaines valeurs de crêtes et rendre plus lisible le graphique. Chaque graphique illustre la mesure pour un type de paquet bien précis. Chacune des traces représente un protocole d'encryption particulier. Cinq mesures de protocole ont été réalisées. Sans encryption, avec WEP 128 bits, TKIP, AES et IPsec.

Il faut savoir que les graphiques présentés sont issus de la comparaison de chaque mesure indépendante. En effet, il a fallu effectuer une mesure après l'autre pour les deux types de paquets

et pour chaque protocole d'encryption. C'est-à-dire 2 fois 5 mesures, 10 mesures en tout. Dans ce rapport, les 4 mesures des algorithmes d'encryption ont été regroupées sur le même graphique afin de mieux mettre en évidence la baisse de débits.

Pour les paquets TCP

Tout d'abord, montrons le débit mesuré pour le transfert de données non cryptée. La figure 12.5 représente le débit en Mbits/s en fonction du temps.

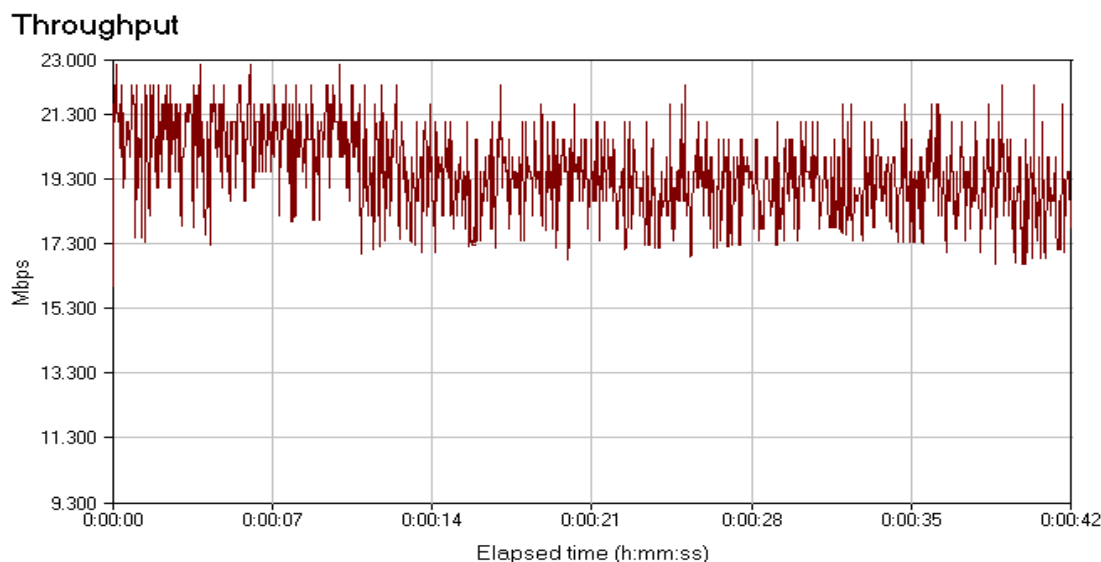


Fig. 12.5 Débit des données non cryptées pour le protocole TCP

Premièrement, on remarque que le débit varie très rapidement entre 17.3 et 21.3 Mbits/s environ. Il n'est pas du tout régulier comme on pourrait le remarquer dans le cadre des réseaux câblés. Ceci est dû au média de transmission qui est sujet à de nombreuses perturbations de tout genres. On relève que la valeur moyenne de ce débit (calculée par le logiciel Chariot) est de 19.4 Mbits/s, ce qui est relativement honnête pour une liaison 802.11g dont seulement une extrémité de la communication passe par les airs. Si les deux intervenants de la communication sont associés au même AP, le débit est divisé par deux.

Si l'on compare maintenant ce débit avec ceux mesurés pour WEP 128, AES, TKIP et IPsec, on s'aperçoit bien des baisses de débit. La figure 12.6 représente ces comparaisons.

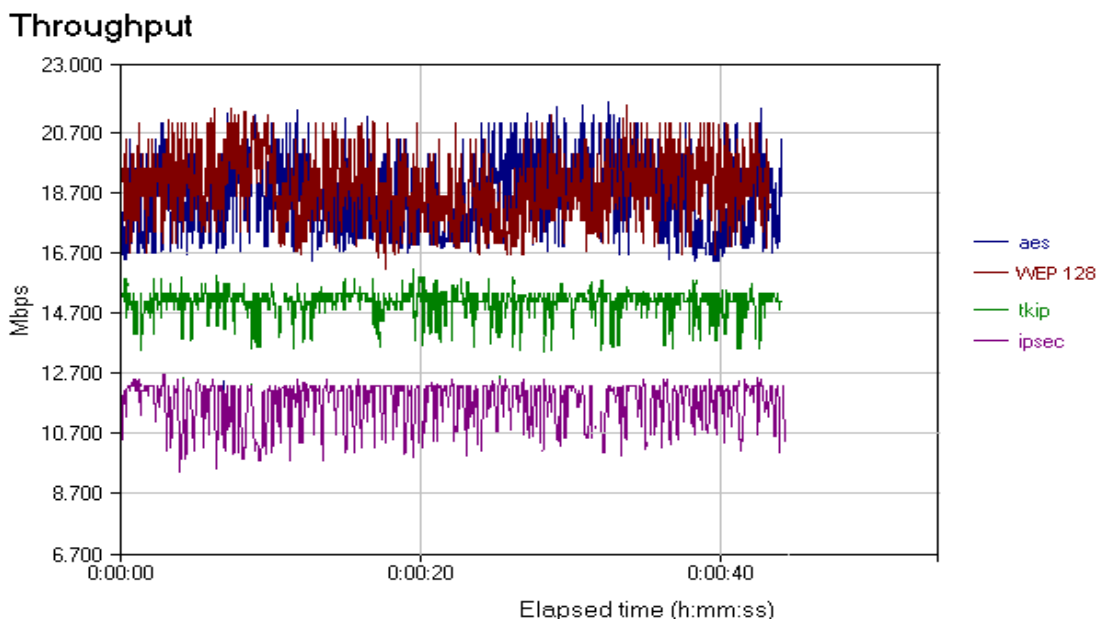


Fig. 12.6 Débit des données cryptées pour le protocole TCP

Dans ce deuxième graphique, on voit nettement la baisse des performances. On remarque tout d'abord que WEP 128 bits et AES ne diminuent pas forcément beaucoup le débit, moins d'un Mbits/s dans notre cas. Ces solutions sont directement implémentées dans le matériel et ne coûtent donc pas très chers en performances. Là aussi, beaucoup de fluctuations du débit interviennent dans la mesure. Les valeurs du débit de AES et de WEP 128 se chevauchent même, tant le délai n'est pas constant. Par contre, pour TKIP, le débit est très nettement affecté. Pourtant TKIP utilise WEP 128. Le problème vient de la création et de la gestion des clés, qui doit être mis à jour au niveau du firmware, donc les opérations prennent plus de temps que si elles étaient directement implémentées dans le matériel comme pour AES avec CCMP. On relève les valeurs moyennes en Mbits/s sont de 18.3 pour AES, 18.8 pour WEP128, 14.9 pour TKIP et 11.6 pour IPsec. On peut donc quantifier, bien que ces mesures soient un peu légères, la baisse de débit en pourcent. Par rapport au débit mesuré sans encryption, le fait d'activer WEP 128 et AES ne change pas grand chose. On peut dire que le débit est diminué de 3% pour WEP, 5.6% pour AES, 23.2% pour TKIP et 40.2% pour IPsec. La si grande baisse avec IPsec se justifie par le fait qu'il s'agit d'un protocole de niveau réseau, mais surtout à cause de son architecture de protocole lourde et ces nombreux algorithmes d'encryption et de hachage. Encore une fois, étant donné que ces mesures ne sont pas forcément représentatives de toutes les situations, ces chiffres sont à prendre avec précaution. Mais cela permet tout de même de donner un ordre d'idée quant aux coûts en performances de ces algorithmes.

Pour les paquets UDP

Pour ce type de paquet, le graphique représenté par la figure 12.7 montre tout d'abord le débit des données non cryptées. Ce dernier est plus bas qu'avec TCP. Ceci est dû au fait que TCP peut gérer de manière dynamique la taille de la fenêtre et l'ajuster en fonction des performances du réseau. C'est bien connu, TCP est un protocole orienté bytes, qui fonctionne selon le principe de flux, ce qui est plus performant que les rafales de paquets UDP toujours de la même taille.

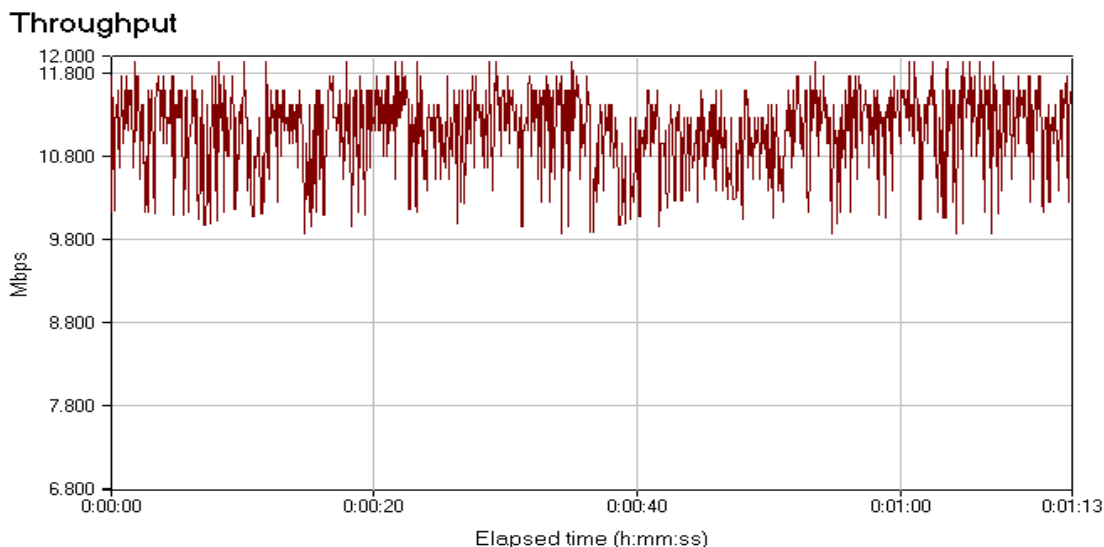


Fig. 12.7 Débit des données non cryptées pour le protocole UDP

On relève donc que la valeur moyenne du débit pour ce type de paquets vaut 11.1 Mb/s. Cette mesure n'apporte pas grand chose de plus que la précédente. Elle met simplement en évidence le désavantage d'UDP par rapport à TCP au niveau des performances du protocole. Ce n'est pas vraiment ce qui nous intéresse dans le cadre de ce projet. Lors de la comparaison des différents algorithmes d'encryption, représentée par la figure 12.8, on s'aperçoit que le comportement est le même que précédemment.

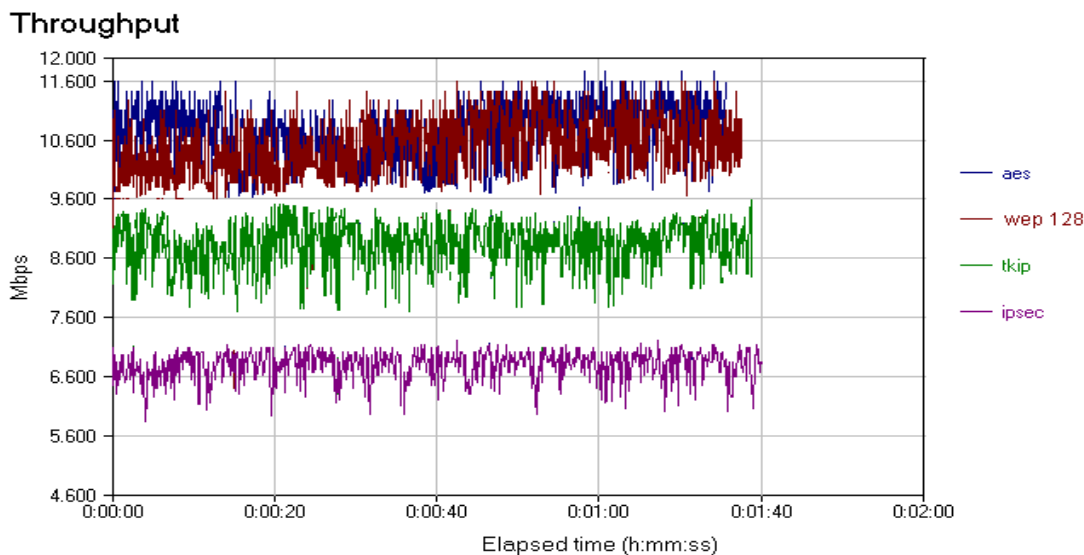


Fig. 12.8 Débit des données cryptées pour le protocole UDP

Les valeurs moyennes des différents débits se trouvent donc encore plus basse que pour la connexion TCP (10.7 pour AES, 10.3 pour WEP 128, 8.7 pour TKIP et 6.7 pour IPsec). Par contre, la baisse en pourcentage reste équivalente. Elle se situe au alentours de 3.6% pour AES, 7.2% pour WEP 128, 21.6% pour TKIP et 39.6% pour IPsec.

Comme synthèse de ces brèves mesures, il faut garder en tête une chose. Les valeurs à retenir sont plus qualitatives que quantitatives. C'est-à-dire que c'est un ordre de grandeur plutôt que des valeurs précises et fiables qui est donné. Je ne veux pas dire que ces mesures sont fausses, mais qu'il ne faut pas les prendre comme références absolues valable dans tous les cas de figure. Le fait d'avoir la même proportion pour la diminution du débit avec TCP et UDP, permet de démontrer que cette baisse est indépendant du type de trafic qui circule sur le réseau. Par contre, pour des applications bien précises, comme la voix ou la vidéo, il aurait pu être intéressant de mesurer les délais supplémentaires que l'encryption ajoute avec différent codecs, afin de déterminer l'impact sur la QoS. Malheureusement cela n'a pas pu être testé par manque de temps.

13 Travail à poursuivre

Maintenant que la plateforme est installée et configurée, comment la gérer et qu'en faire ? Elle n'est bien évidemment pas totalement au point. Quelques améliorations peuvent être apportées. Ce chapitre a pour but de décrire le travail à suivre pour assurer une certaine continuité à cette plateforme.

Premièrement, on peut parler du serveur d'acompte. En effet, cette option de freeRADIUS a été totalement ignorée lors de ce projet. Il pourrait être intéressant de mettre en œuvre ce mécanisme afin de garantir une meilleure gestion des utilisateurs et de leurs données de session.

Au niveau de EAP, une amélioration certaine serait de changer la méthode d'authentification. En effet, un challenge MD5 n'est pas forcément des plus sûr. Tout repose sur un simple mot de passe. Donc, l'idée serait de passer à PEAP ou EAP-TLS. Le serveur freeRADIUS et tous les clients 802.1x cités dans ce rapport permettent ces modes d'authentification. Cela implique la configuration du serveur (fichier radiusd.conf) afin d'accepter un nouveau module d'authentification et la mise en œuvre d'une structure de certificats. Cette structure se compose d'un certificat ROOT capable de valider des requêtes afin de délivrer des certificats aux utilisateurs. Il est possible de le faire à l'aide de deux commandes du packaging openssl [22] sous Linux, `newca` et `newreq`. J'ai essayé de monter cette hiérarchie de certificats, mais par manque de temps j'ai dû renoncer. Pour les détails d'implémentation se référer à [36].

Je pense important de configurer un client Linux pour l'authentification et l'encryption. Pour cela il faut d'abord posséder un client 802.1x pour le dialogue EAP. Il en existe plusieurs cités dans ce rapport. Ce client doit aussi pouvoir paramétrer les options de sécurité de la carte afin d'activer le bon mécanisme d'encryption. Le problème est de trouver une carte compatible. Le fait d'implémenter PEAP ou TLS impliquent également la gestion de s certificats.

Il serait intéressant de créer un tunnel IPsec entre Windows et Linux. Actuellement, il est possible de créer une connexion IPsec entre deux PC windows ou deux PC Linux. Un tunnel hétérogène n'est pas implémenté. Là toujours, les certificats sont recommandés pour une meilleur sécurité.

Donc, un mécanisme de certificats DOIT être rapidement mis en œuvre pour élever le niveau de la sécurité.

De plus, une fois que le matériel normalisé 802.11i sortira sur la marcher, il faudrait l'installer et évaluer ces performances. Des mesures plus approfondies que dans ce projet sont encore à faire.

Finalement, le dénouement de ce projet serait de créer un Hot Spot opérationnel, fiable et sécurisé, qui puisse être présenté à des entreprises. Mais il reste quelques petites étapes avant d'y arriver...

14 Conclusion

Il est primordial de sécuriser son réseau sans fil de manière draconienne, si vous sentez que vos ressources et vos données sont cruciales pour votre entreprise. Les mots d'ordres sont **authentification, interdiction d'accès, encryption et gestion des clés**. A la lueur de ce rapport, un seul schéma est actuellement envisageable autant pour ses qualités de sécurité que pour ses performances, il s'agit de celui proposé par **IEEE 802.11i**. D'un côté, le protocole d'authentification **RADIUS** accompagné de **IEEE 802.1x** garantissent un accès réservé uniquement aux utilisateurs certifiés. De l'autre, **CCMP** basé sur **AES** fournit une encryption efficace et performante, comme nous avons pu l'entrevoir lors des mesures effectuées.

Bien que la norme IEEE 802.11i ne soit pas encore standardisée au niveau du matériel, tous les composants faisant partie de l'architecture sont déjà disponibles. Il est donc déjà possible d'implémenter un schéma équivalent, mis à part le fait que les points d'accès et les interfaces devront être mis à jour, au niveau du firmware, lors de la sortie sur le marché du matériel certifié. Pour ceux qui veulent attendre avant d'investir, il existe une alternative qui consiste à combler certaines failles de WEP, en rajoutant le protocole de gestion des clés **TKIP**. Mais les baisses de performances qu'engendre l'implémentation de ce dernier ne sont pas négligeables et diminuent considérablement le débit des données.

Personnellement, j'ai eu du plaisir à effectuer ce projet de diplôme et je suis satisfait du travail accompli. Ce fut une excellente expérience qui m'aura permis de découvrir en profondeur un domaine des télécommunications qui m'intéresse, qui est en constante recherche et qui risque de durer encore longtemps. Mon seul regret est de ne pas avoir pu considérer les problèmes de performances liés aux applications multimédia comme il était demandé dans le cahier des charges.

15 Remerciements

Je pense important de citer les personnes qui sont venues à mon secours. Tout d'abord deux assistants du département logiciel, Yann Crausaz et Philippe Waelti, qui m'ont redonné espoir quand ils ont réussi à installer Linux sur mon portable après plusieurs jours perdus. Ensuite, Christian Tettamanti, un assistant du département télécommunication, qui m'a également aidé pour installer le sniffer sur le même portable. Finalement mon collègue de bureau Mesoud Hasanovic, qui m'a prêté son matériel alors que j'avais mal géré le miens. Finalement, je remercie Stephan Robert, pour m'avoir orienté dans ce projet.

16 Références

- [1] S. Fluhrer, I. Mantin et A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4", Eighth Annual Workshop on Selected Areas in Cryptography, 2001.
- [2] A. Stubblefield, J. Ioannidis et A.D. Rubin, "Using the Fluher, Mantin and Shamir Attack to Break WEP", AT&T Labs-Research, Florham Park, NJ, 2001.
- [3] Site officiel du groupe de développeurs The Shmoo Group : <http://www.shmoo.com> et site officiel de Airtsnort : <http://airsnort.shmoo.com>
- [4] Linux PCMCIA Information Page de source forge : <http://pcmcia-cs.sourceforge.net>
- [5] The Gimp Tool Kit, bibliothèque GTK : <http://www.gtk.org>
- [6] C. Rigney, S. Willens, Livingstone, A. Rubens, Merit, W. Simpson Dyadreamerm, "Remote Authentication Dial In User", RFC 2865, juin 2000.
C. Rigney, Livingstone, "Radius Accounting", RFC 2866, Juin 2000.
- [7] IEEE standard 802.1X-2001, "Port-based Network Protocol", Juin 2001.
- [8] L. Blunk, J. Vollbrecht, Merit Network Inc., "PPP Extensible Authentication Protocol", RFC 2284, Mars 1998.
- [9] B. Aboba, D. Simon, Microsoft, "PPP EAP TLS Authentification Protocol", RFC 2716, Octobre 1999.
- [10] Anderson, Hakan, S. Josefsson, G. Zorn, D. Simon et A. Palekat, "Protected EAP Protocol", IETF draft-josefsson-pppext-eap-tls-eap-05.txt, Septembre 2002.
- [11] Funk, P. et S. Blake-Wilson, "EAP Tunneled TLS Authentication Protocole", IETF draft-ietf-pppext-eap-tls-02.txt, Novembre 2002.
- [12] Wi-Fi Alliance, "Wi-Fi Protected Access: Strong standards-based, interoperable security for today's Wi-Fi networks", Avril 2003.
- [13] Status du projet 802.11i: http://grouper.ieee.org/groups/802/11/Reports/tgi_update.htm

-
- [14] National Institute of Standard and Technology (NIST) sur le standard AES :
<http://csrc.nist.gov/CryptoToolkit/aes/>
 - [15] Federal Information Processing Standard, “Advanced Encryption Standard”, publication 197, Novembre 2001.
 - [16] D. Whiting, Hifn, R. Housley, Vigil Security, N. Ferguson, MacFergus, “Counter with CBC-MAC”, RFC 3610, Septembre 2003.
 - [17] D. Harkins, D. Carrel, CISCO systems, “The Internet Key Exchange”, RFC 2409, Novembre 1998.
 - [18] S. Kent, BBN Corp., R. Atkinson, @Home Network, “IP Authentication Header”, RFC 2402, Novembre 1998.
 - [19] S. Kent, BBN Corp., R. Atkinson, @Home Network, “IP Encapsulating Security Payload”, RFC 2406, Novembre 1998.
 - [20] Site officiel de l’IETF sur le groupe de développement IPsec.
<http://www.ietf.org/html.charters/ipsec-charter.html>
 - [21] T. Dierks, Certicom, C. Allen, “The TLS Protocol Verison 1.0”, RFC 2246, Janvier 1999.
 - [22] Site officiel de la version open source de SSL: <http://www.openssl.org>
 - [23] Site officiel de freeRADIUS : <http://www.freeradius.org>
 - [24] Scott Bartlett , “FreeRadius and MySQL notes”, février 2003, scott@frontios.com,
<http://www.frontios.com/freeradius.html>
 - [25] Site officiel de la plus populaire base de données open source: <http://www.mysql.com>
 - [26] CISCO system, “Aironet 350 séries Bridge Software configuration Guide“, 2003.
 - [27] Site officiel de MeetingHouse: <http://www.mtghouse.com>, client 802.1x AEGIS.
 - [28] Site officiel de Funk Software: <http://www.funk.com>, client 802.1x Odyssey.
 - [29] Site officiel de 802.1x sous Linux: <http://www.open1x.org>, client Xsupplicant.
 - [30] Documentation microsoft de son client 802.1x pour windows 2000:
<http://www.microsoft.com/windows2000/server/evaluation/news/bulletins/8021xclient.asp>
 - [31] Site officiel de Buffalo Technology Inc., <http://www.buffalotech.com>.
Pilotes WLI-CB-G54a : <http://www.buffalotech.com/wireless/support/downloads.php>
 - [32] Documentation Microsoft, “How to configure IPsec tunnelling in windows 2000”, Microsoft Knowledge Base Article 252735, Novembre 2003.

- [33] Site officiel de freeswan : <http://www.freeswan.org>
- [34] Site officiel consacré à la mise en oeuvre d'IPsec sous Linux : <http://www.ipsec-howto.org/>
- [35] Nate Carlson, "Configuring an IPsec tunnel between freeswan and windows 2000," <http://www.natecarlson.com/linux/ipsec-x509.php>, Novembre 2003
- [36] Lien consacré à la mise en œuvre des certificats sous Linux et plus particulièrement freeswan:
<http://www.strongsec.com/freeswan/>
- [37] David L. Mills, University of Delaware, "Network Time Protocol, Specification, Implementation and Analysis", RFC 1305, Mars 1992.
- [38] Site officiel du projet Network Time Protocol installaiton <http://www.ntp.org>
- [39] Marche à suivre de l'installation de NTP sous Linux.
http://www.eecis.udel.edu/~ntp/ntp_spool/html/build.html
- [40] Logiciel client pour NTP: http://www.cru.fr/NTP/clients_ntp.html
- [41] Site officiel de NetIQ, logiciel Chariot: <http://www.netiq.com/products/chr/default.asp>

17 Table des figures

FIG. 2.1 ALGORITHME KSA ET ALGORITHME PRGA	4
FIG. 2.2 TABLEAU RC4 INITIAL	4
FIG. 2.3 EXEMPLE D'UN TABLEAU RC4 PERMUTÉ APRÈS KSA	5
FIG. 2.4 SCHÉMA DE RC4	5
FIG. 2.5 TRAME WEP ET EN-TÊTE LLC	5
FIG. 2.6 TABLEAU RC4 APRÈS KSA	6
FIG. 2.7 PLATEFORME DE TEST WEP	7
FIG. 2.8 GUI DE AIRSNORT	10
FIG. 2.9 MESSAGE D'ERREUR: "COULD NOT SET MONITOR MODE"	11
FIG. 2.10 RECHERCHE DE BSSID	11
FIG. 2.11 CASSAGE RÉUSSI	11
FIG. 2.12 RÉSULTATS POUR LES CLÉS DE 64 BITS	13
FIG. 2.13 RÉSULTATS POUR LES CLÉS DE 128 BITS	14
FIG. 4.1 ECHANGE DES MESSAGES RADIUS	18
FIG. 4.2 EN-TÊTE GÉNÉRALE DES PAQUETS RADIUS	19
FIG. 4.3 FORMAT DU CHAMP ATTRIBUTES DE LA TRAME RADIUS	20
FIG. 5.1 SCHÉMA PORT-BASED NETWORK ACCESS CONTROL	21
FIG. 5.1 EAP	23
FIG. 6.2 ECHANGE DES MESSAGES EAP	25
FIG. 6.3 TRAME EAP	26
FIG. 6.4 PAQUET EAP REQUEST ET RESPONSE	26
FIG. 7.1 RÉALISATION DE L'AUTHENTIFICATION	27
FIG. 8.1 SCHÉMA TKIP	29
FIG. 9.1 TABLEAU INTERNE DE AES	31
FIG. 9.2 SCHÉMA CCMP	32
FIG. 10.1 COUCHE SSL	34
FIG. 11.1 PLATEFORME	36

FIG. 11.2 INTERFACE GRAPHIQUE DIALUP ADMIN.....	45
FIG. 11.3 CONFIGURATION CISCO, HOMEPAGE.....	47
FIG. 11.4 CONFIGURATION CISCO, SETUP PAGE.....	47
FIG. 11.5 CONFIGURATION CISCO, SECURITY SETUP PAGE.....	48
FIG. 11.6 CONFIGURATION CISCO, RADIO DATA ENCRYPTION PAGE.....	48
FIG. 11.7 CONFIGURATION CISCO, RADIO DATA ENCRYPTION PAGE 2.....	49
FIG. 11.8 CONFIGURATION CISCO (AUTHENTICATOR CONFIGURATION).....	50
FIG. 11.9 CONFIGURATION SERVICE AEGIS.....	51
FIG. 11.10 CLIENT AEGIS.....	51
FIG. 11.11 CONFIGURATION DU PROFIL D'AUTHENTIFICATION AEGIS.....	52
FIG. 11.12 CONFIGURATION PROFIL RÉSEAU AEGIS.....	53
FIG. 11.13 CONFIGURATION PROFIL RÉSEAU AEGIS 2.....	53
FIG. 11.14 CONFIGURATION PROFIL RÉSEAU AEGIS 3.....	54
FIG. 11.15 CONFIGURATION 802.1X WINDOWS.....	55
FIG. 11.16 CONFIGURATION 802.1X WINDOWS 2.....	55
FIG. 11.17 PAGE DES PARAMÈTRES DE SÉCURITÉ DE L'AP BUFFALO.....	57
FIG. 11.18 PROPRIÉTÉ DE LA CARTE APRÈS L'INSTALLATION DU CLIENT ODYSSEY.....	57
FIG. 11.19 CONFIGURATION DU MODE DE SÉCURITÉ DE LA CARTE BUFFALO DEPUIS LE CLIENT ODYSSEY.....	58
FIG. 11.20 CONNEXION AU RÉSEAU DEPUIS LE CLIENT ODYSSEY.....	58
FIG. 11.21 POLICES D'ACCÈS.....	59
FIG. 11.22 RÈGLES IPSEC.....	59
FIG. 11.23 PROPRIÉTÉS IPSEC.....	60
FIG. 12.1 EFFET DE L'ENCRYPTION.....	62
FIG. 12.2 PLATEFORME DE MESURE AVEC CHARIOT 4.3.....	63
FIG. 12.4 SCRIPTS DE MESURE.....	64
FIG. 12.5 DÉBIT DES DONNÉES NON CRYPTÉES POUR LE PROTOCOLE TCP.....	66
FIG. 12.6 DÉBIT DES DONNÉES CRYPTÉES POUR LE PROTOCOLE TCP.....	67
FIG. 12.7 DÉBIT DES DONNÉES NON CRYPTÉES POUR LE PROTOCOLE UDP.....	68
FIG. 12.8 DÉBIT DES DONNÉES CRYPTÉES POUR LE PROTOCOLE UDP.....	68