Détecter et protéger la passerelle d'un réseau des attaques de type « ARP Cache Poisoning »

1. Introduction

Les attaques de type « ARP Cache Poisoning » sont des attaques relativement simplement simple à mettre en place sur un réseau Ethernet basé sur le protocole ARP[1] (Address resolution protocol). Celles-ci pouvent détourner une partie du traffic d'un réseau afin de récupérer des informations confidentielles. Il est intéressant d'augmenter la sécurité d'une passerelle réseau pour ce type d'attaques car par elle transite souvent de nombreuses informations sensibles.
Pour ce faire nous utiliserons le module ARP* (prononcé ArpStar) qui est un module pour les kernels Linux 2.4/2.6. Il permet de détecter et d'empecher la plupart des attaques de type « ARP Cache Poisoning » sur un réseau Ethernet basé sur un adressage en IP fixe.
Ils existent de nombreux logiciels[2] permettant de détecter ce type d'attaque mais aucun d'entre eux n'est capable de les éviter. Arp* a été développé en tant que module pour les kernels Linux, de ce fait, il est capable d'intercepter les packets ARP avant le kernel, de les traiter et éventuellement de les rejetés, sécurisant ainsi le système.

2. Fonctionnement

Dans ce paragraphe nous ne reviendrons pas sur le fonctionnement du protocole ARP mais plutot sur une de ses faiblesses qui permet d'exploiter des attaques de type « ARP Cache Poisoning ».

Dans un réseau Ethernet pour qu'une machine puisse établir une communication, elle doit connaître l'adresse MAC de la machine destinatrice. Elle émet donc une requete ARP (ARP Request) sur tout le réseau local (Broadcast[3]) afin de savoir a qui correspond l'adresse IP avec qui elle doit prendre contact. Puis la machine concernée envoye une réponse (ARP Reply) en indiquant l'association IP/MAC. Cette réponse est stocké dans le cache ARP de la machine emetrice pour les communications futures.
Une des faiblesses du protocole ARP est d'accepter les packets de type ARP Reply sans vérifié si celui-ci correspond à un ARP Request qu'elle aurait émis. Cela permet à un attaquant d'envoyer un ARP Reply avec une association IP/ARP usurpé à une machine afin de corrompre son cache ARP et ainsi détourné une partie du trafic. La plupart des logiciels capable d'effectuer des attaques de type « ARP Cache Poisoning » utilise cette faiblesse du protocole ARP.

Le fonctionnement d'Arp* est basé sur le fait que tout ARP Reply est sollicité pour un ARP Request. Ainsi, si une machine recoit un packet de type ARP Reply sans avoir émise d'ARP Request alors le packet sera rejeté. Si la machine ne l'a pas demandé c'est qu'elle n'en a pas besoin, ce qui paraît plutot logique.
D'autre part, lors de la réception d'un ARP Reply, ARP* compare l'association IP/MAC contenu dans le packet avec celle disponible dans son cache ARP afin de repérer un éventuel changement d'adresse MAC.
Mais malheureusement cette technique est relativement limitée comme nous allons le voir dans le prochain paragraphe.

3. Limitations

Une des limitations pour se défendre contre les attaques de type « ARP Cache Poisoning » intervient lorsqu'il n'y a pas d'entrée dans le cache ARP correspondant au contenu du packet ARP Reply recu. Ainsi on ne peut etre sur que l'ARP Reply correspond à une entrée valide.

Une autre limitation intervient lorsqu'une collision se produit c'est à dire lorsqu'une machine recoit plusieurs ARP Reply en meme temps, de contenu différent, correspondant à un meme ARP Request. La machine ne saura donc pas lequel choisir.
Dans ce cas, ARP* priviligie la connection à l'intégrité de la connection et autorise les packets.

D'autre part, Arp* permet via un systeme de signature présent dans les paquets ARP, d'identifier le logiciel auteur de l'attaque. Cependant ce systeme de détection est loin d'etre fiable.

4. Installation

La passerelle de test sur laquelle nous allons installer ARP* est une Ubuntu 6.06 (LTS).
Afin d'installer le module Arp* nous avons besoin des packets Ubuntu suivant:

- build-essential
- linux headers

Pour les installer :

- *apt-get install build-essential*
- *apt-get install linux-headers-`uname -r`*

Allez télécharger les sources du module Arp* sur http://sourceforge.net/projects/arpstar/

Puis décompresser l'archive et faite un *make*.

Copier le module arpstar.ko obtenu : cp arpstar.ko /lib/modules/`uname -r`

Indexez le module dans le fichier des dépendances avec la commande :

```
echo "/lib/modules/`uname -r`/arpstar.ko:" >>/lib/modules/`uname -r`/modules.dep
```

Et enfin chargez le module dans le kernel à l'aide de la commande : *modprobe arpstar*

Pour vérifier que le module est bien chargé, taper la commande *: modprobe – l | grep arpstar*

Ce qui devrait vous renvoyez quelquechose comme ca *:* /lib/modules/2.6.15-28-386/arpstar.ko

Arp* envoi ses alertes en cas d'attaque via syslog dans le fichier */var/log/syslog* :

kernel: [17387044.500000] arp*: CAIN & ABEL DETECTED at AA:14:6C:65:E9:3E trying to poison 192.168.0.1

5. Conclusion

Arp* permet d'augmenter le niveau de sécurité de la passerelle en interceptant et en contrant la plupart des attaques de types « ARP Cache Poisoning ». Mais comme toutes les sécuritées, elle n'est pas fiable à 100%.
Des alternatives à Arp* peuvent etre mise en place comme le protocole S-ARP[4]. Celui-ci s'appuie sur un système de *clé public/clé privé* afin d'assurer l'intégrité des données.

6. Bibliographie

(1) : http://fr.wikipedia.org/wiki/Address_Resolution_Protocol

(2) : Arpwatch, Arpcachewatch

(3) : http://fr.wikipedia.org/wiki/Broadcast

(4) : http://www.acsac.org/2003/papers/111.pdf