SAÉ24 — Attaques MITM sur un réseau local

Table des matières

1 Introduction

L'objectif de la SAÉ24 est de programmer et d'expérimenter des attaques de type MITM (Man In The Middle) sur un réseau local et d'étudier des mesures de protection pouvant être mise en œuvre pour contrer ces attaques. L'objectif est d'expérimenter ces attaques réseau à des fins pédagogiques (comprendre des attaques pour savoir comment s'en protéger). **Ne réalisez ces opérations que dans un cadre contrôlé (salle de TP, machines virtuelles confinées, . . .)**.

La SAÉ se décompose en trois parties. Dans la première partie, on programmera (en python) et expérimentera l'attaque ARP spoofing vue en TD. Dans la deuxième partie, on expérimentera une attaque sur SSH. Enfin, dans la troisième partie, on étudiera et expérimentera des techniques intégrées aux équipements CISCO qui permettent de bloquer ces attaques sur un réseau local.

2 Environnement de travail

Pour réaliser la SAÉ vous devrez travailler sur trois machines :
— une première jouera le rôle de l'attaquant ;
— une seconde jouera le rôle d'un serveur ;
— et une troisième jouera le rôle d'un client.

Le code python sera donc placé et exécuté sur l'attaquant. Le but de l'attaquant est, dans la partie 1, d'espionner les connexions TCP et les demandes d'écho échangées entre le client et le serveur ; puis, dans la partie 2, d'usurper l'identité du serveur auprès du client lors d'une connexion SSH. Les deux premières parties peuvent être réalisées dans un environnement virtuel (i.e., avec des machines virtuelles) ou physiques (i.e., sur les PC des salles de TP). Pour la troisième partie, vous devrez travailler sur des équipements physiques (PC et switch Cisco).

Environnement physique

Dans les salles de TP réseaux (P20X), l'image à utiliser est *Debian11-Butelle*. Scapy n'étant pas installé sur ce système, il faut suivre les indications du TP pour l'installer.

Environnement virtuel

Pour les parties 1 et 2, si vous travaillez avec des machines virtuelles, vous pourrez (mais ce n'est pas obligatoire) utiliser la VM suivante :

`https://lipn.univ-paris13.fr/~evangelista/cours/SAE24/sae24.ova`

Dans les salles de TP de l'IUT, le fichier se trouve également ici :

`/iutv/Mes_Montages/TP/TPRT/evangelista/sae24.ova`

Cette VM est une debian 11 contenant a priori tout ce qu'il faut pour réaliser la SAÉ (python3, scapy, wireshark, openssh, ssh-mitm). Un seul compte utilisateur existe : user. Son mot de passe est 123456, de même que le mot de passe de root. Vous pouvez utiliser cette VM pour les trois machines.

Pour que tout ceci fonctionne, il faut que les VM puissent communiquer entre elles. Il y a différentes choses à configurer pour cela. Voici la procédure pour VirtualBox v6 (la procédure est similaire pour la v7 mais les intitulés des menus peuvent changer) :

1. Dans le menu *Fichier* → *Gestionnaire de réseau hôte*, créez un nouveau Réseau. Dans les détails du réseau, onglet *Serveur DHCP*, attribuez l'IP 10.N.0.100 (N étant votre numéro de groupe attribué par l'enseignant) à votre serveur DHCP et indiquez 10.N.0.1 et 10.N.0.99 comme limites des adresses.
2. Téléchargez le fichier ova.
3. Créez une VM depuis le menu *Fichier* → *Importer un appareil virtuel* puis sélectionnez le fichier ova. Nommez vos VM vm-attaquant.
4. Faites de même avec une deuxième VM nommée vm-client.
5. Faites de même avec une troisième VM nommée vm-serveur.
6. Dans la configuration réseau des VM, changez le mode d'accès réseau : sélectionnez *Réseau privé hôte* et générez une adresse MAC aléatoire. (Sinon les VM auront la même adresse MAC et obtiendront la même IP auprès du serveur DHCP.)
7. Dans certaines salles de TP de l'IUT, il faut aussi désactiver le contrôleur USB (paramètres *USB* de la VM).
8. Démarrez les VM, ouvrez une session et vérifiez dans un terminal qu'elles ont bien une IP (`ip addr show`), que leurs IP sont différentes et bien dans votre réseau.

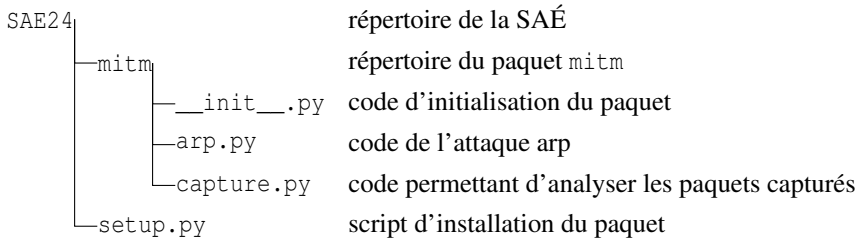
Le mode *Réseau privé hôte* de VirtualBox crée un réseau virtuel confiné dans lequel nos trois VM peuvent communiquer. Par contre elles ne peuvent pas communiquer avec l'extérieur, ce qui permet d'éviter que des paquets d'attaque soient envoyés sur le réseau physique. Si vous voulez accéder à Internet depuis une VM, p.ex., pour vous envoyer des fichiers, il faudra temporairement passer le mode réseau de la VM en mode NAT, puis réactiver l'interface réseau de la VM (sans redémarrer la VM).

3 Partie 1 — Attaque ARP spoofing avec le paquet mitm

Le paquet `mitm` contient le code de l'attaque d'empoisonnement ARP ainsi que du code permettant, une fois l'attaque lancée, d'observer les connexions TCP et les paquets ICMP échangés entre le client et le serveur.

3.1 Réalisation

Tout votre code sera placé dans un répertoire `SAE24` contenant le paquet ainsi que le script d'installation.



Le module `__init__.py` Il définit un numéro de version du paquet et affiche un message de bienvenue contenant le numéro de version, comme dans le paquet biblio écrit en R208.

Le module `arp.py` Il définit la fonction suivante :

```
def arp(ipa, ipb, delay=5):
```

Cette fonction réalise l'attaque d'ARP spoofing vue en TD afin que l'attaquant (l'hôte qui lance le script) capture tous les paquets échangés entre les deux hôtes dont les IP sont passées en paramètre (`ipa` et `ipb`). Le paramètre `delay` est, en secondes, le délai entre l'envoi de paquets d'attaque.

Utilisez la fonction `sleep` du module `time` pour introduire un délai dans le programme.

Il arrive que scapy n'envoie pas les paquets sur la bonne interface. Pour éviter ce problème, il faut préciser explicitement l'interface d'envoi en appelant `send`, `sendp`, `sr1` ou `srp1` avec le paramètre `iface` ayant comme valeur le nom de l'interface sur laquelle on veut envoyer le paquet. Par exemple :

```
paquet = Ether() / ARP()
sendp(paquet, iface="enp0s3")
```

Le module `capture.py` Il définit la fonction suivante :

```
def tcp_icmp(ip_client, ip_server, timeout=60, output="capture.json"):
```

Cette fonction capture pendant `timeout` secondes les paquets allant de `ip_client` vers `ip_server`. Elle affiche dans le terminal :

— une ligne pour chaque demande de connexion TCP capturée ;
— et une ligne pour chaque demande d'écho capturée.

Voici un exemple d'affichage de la fonction :

```
TCP 10.1.0.1:23443 -> 10.1.0.100:80
TCP 10.1.0.1:12522 -> 10.1.0.100:22
ICMP 10.1.0.1 -> 10.1.0.100
```

Le deux premières lignes ont été affichées suite à la capture de deux paquets TCP de demande de connexion des ports 23443 et 12522 vers les ports 80 et 22 respectivement. Les IP 10.1.0.1 et 10.1.0.100 sont respectivement les IP du client (`ip_client`) et du serveur (`ip_server`). Enfin la troisième ligne a été affichée suite à la capture d'une demande d'écho de 10.1.0.1 vers 10.1.0.100.

Les données affichées dans le terminal seront également sauvegardées au format JSON dans le fichier passé en paramètre (paramètre `output`). Les données JSON consisteront en une liste de dictionnaires. Chaque dictionnaire de la liste correspondra à un paquet capturé et aura trois clés : `prot` pour le protocole (`TCP` ou `ICMP`), `src` pour la source et `dst` pour la destination. Par exemple, pour l'affichage donné plus haut, le fichier aura le contenu ci-dessous :

```
[
  { "prot": "TCP",  "src": "10.1.0.1:23443", "dst": "10.1.0.100:80" },
  { "prot": "TCP",  "src": "10.1.0.1:12522", "dst": "10.1.0.100:22" },
  { "prot": "ICMP", "src": "10.1.0.1",       "dst": "10.1.0.100"    }
]
```

Le module `setup.py` Il contient le code d'installation du paquet `mitm`, comme celui écrit en R208. Attention, le paquet `mitm` ne contient pas de script exécutable. Il est donc inutile de passer l'argument `scripts` à la fonction `setup`.

Respectez *scrupuleusement* la structure du répertoire `SAE24` donnée plus haut ainsi que les noms de fichier, les noms de fonction et les noms des paramètres des fonctions. Une partie de votre code sera testée automatiquement. Il faut donc que les noms correspondent exactement sinon les tests ne fonctionneront pas.

3.2 Tests

Le paquet `mitm` doit être installé sur l'attaquant. Vous ouvrirez ensuite deux terminaux sur l'attaquant. Dans un premier terminal vous lancerez l'attaque sur le client (10.1.0.1 dans notre exemple) et le serveur (10.1.0.100 dans notre exemple) dans l'interpréteur python :

```
>>> from mitm import arp
>>> arp.atk("10.1.0.1", "10.1.0.100")
```

Dans le second terminal vous lancerez la capture dans l'interpréteur python :

```
>>> from mitm import capture
>>> capture.tcp_icmp("10.1.0.1", "10.1.0.100")
```

Puis, sur le client, vous enverrez des demandes d'écho ou de connexion TCP (p.ex., en ouvrant une connexion SSH) vers le serveur. Si l'attaque fonctionne, vous devriez voir des lignes affichées dans le deuxième terminal de l'attaquant.

Quelques commandes qui pourront être utiles pour vos tests :

```
sysctl net.ipv4.ip_forward=1  # active le routage
ip neigh show                 # affiche la table ARP
ip neigh flush all            # vide la table ARP
```

4 Partie 2 — Attaque man-in-the-middle sur SSH

Une attaque ARP spoofing est souvent le préalable à d'autres attaques permettant, p.ex., de manipuler les paquets capturés par l'attaquant. L'objectif de cette deuxième partie est de réaliser l'attaque sur SSH évoquée dans le cours de R203 (Bases des services réseaux). L'attaquant va d'usurper l'identité du serveur auprès du client, et inversement. On observera donc deux connexions SSH ouvertes sur l'attaquant : une dans laquelle il joue le rôle de serveur auprès du client et une autre dans laquelle il joue le rôle de client auprès du serveur. Il va ainsi déchiffrer (avec la clé de session négociée avec le client) toutes les données envoyées par le client (y compris le mot de passe) puis les rechiffrer (avec la clé de session négociée avec le serveur) pour les envoyer au serveur.

Pour réaliser l'attaque, nous allons utiliser le paquet python `ssh-mitm`. Il est déjà installé sur la VM fournie. Sinon vous devrez l'installer avec pip, le gestionnaire de paquets python.

Avant de réaliser cette attaque vous devrez également mettre en place une règle de translation d'adresses sur l'attaquant (voir l'outil iptables) pour qu'il modifie l'adresse IP des paquets SSH qu'il reçoit du client. En effet, lorsque l'attaquant reçoit les paquets SSH du client, il ne faut pas qu'il les route vers le serveur — ce qu'il ferait normalement puisqu'ils ne lui sont pas destinés. Il faut que ces paquets soient traités par le (faux) serveur SSH local de l'attaquant. La translation d'adresse qui sera mise en place permettra donc de modifier l'adresse de destination des paquets SSH reçus du client par l'adresse de l'attaquant.

Il faut ensuite lancer l'attaque ARP spoofing (comme dans la partie 1) puis lancer la commande `ssh-mitm` (avec les arugments adéquats) installée avec le paquet python pour réaliser l'attaque. Si l'attaque fonctionne vous devriez voir le mot de passe entré par l'utilisateur s'afficher dans le terminal.

5 Partie 3 — Sécurisation du réseau local

Les switch CISCO intègrent deux techniques permettant de protéger le réseau local contre les attaques de type ARP spoofing :

— *DHCP snooping* ;

— et *ARP inspection*.

La première permet au switch de mémoriser les IP attribuées par le serveur DHCP. Pour que cette technique fonctionne il faut donc que les interfaces du réseau soient configurées dynamiquement. À chaque fois qu'un bail est attribué, le switch mémorise le triplet (i, m, p) où :

— i est l'IP attribué par le serveur DHCP au client ;
— m est l'adresse MAC du client ;
— et p le numéro du port du switch auquel est connecté le client.

Par la suite, si la technique d'ARP inspection est activée, le switch vérifiera que chaque paquet ARP qu'il reçoit est cohérent avec les triplets mémorisés. Dans le cas contraire, le paquet ARP sera bloqué (i.e., ignoré) par le switch qui ne le retransmettra pas.

Dans notre scénario, ce sera le switch qui sera le serveur DHCP. Il faudra donc faire les configurations suivantes sur le switch :

1. lui attribuer une adresse IP sur le VLAN 1 ;
2. activer le service DHCP pour que le switch puisse attribuer des IP aux trois hôtes (le client, le serveur et l'attaquant) ;
3. activer le DHCP snooping ;
4. et enfin activer l'ARP inspection.

Une fois toutes ces configurations réalisées, vous pourrez tester en relançant l'attaque ARP spoofing. L'attaque devrait être bloquée par le switch. Vous devriez aussi voir dans le terminal minicom (celui dans lequel vous tapez vos commandes CISCO) des lignes affichées indiquant que les paquets ARP ont été bloqués.

6 Travail à rendre

À la fin de vos travaux vous rendrez :

— le paquet python `mitm` développé dans la partie 1 ;
— et un rapport (maximum 20 pages) détaillant tous les tests effectués (pour les 3 parties), sous forme de compte-rendu de TP avec copies d'écran à l'appui, permettant d'illustrer le fonctionnement et les conséquences des attaques et mesures de protection étudiées.

Remarques additionnelles :

— Le code python devra être envoyé sous forme d'archive et le rapport au format *PDF uniquement*.
— Le projet est à réaliser en binôme.
— La date limite pour l'envoi de votre projet (code du paquet + rapport) est fixée au *vendredi 14 juin*. C'est une date stricte. Tout compte-rendu reçu après cette date sera refusé.
— Il sera tenu compte dans la notation du respect des règles de codage vues en R208 :
  — docstrings dans les fonctions et modules ;
  — commentaires ;
  — lignes de 80 caractères maximum ;
  — et fonctions de 30 lignes maximum.

  Vous pouvez donc utiliser le même fichier `pylintrc` que celui fourni en R208.