Laboratorio de ARP Cache Poisoning

1 Descripción

El Address Resolution Protocol o Protocolo de Resolución de Direcciones (ARP) es un protocolo de comunicación usado para encontrar la dirección de hardware que corresponde a una determinada dirección IP. ARP es un protocolo bastante simple y no implementa ninguna medida de seguridad. El ARP poisoning cache es un ataque bastante común que se ejecuta en contra de este protocolo. Con un ataque de este tipo, los atacantes pueden engañar a la víctima para que acepte mapeos falsificados de IP-to-MAC. Esto puede hacer que los paquetes de la víctima sean redirigidos a la computadora con la dirección MAC falsificada, lo que lleva a posibles ataques man-in-the-middle.

El objetivo de este laboratorio es que los estudiantes ganen experiencia en el ataque de ARP Caché Poisoning y aprendan los daños causado por dicho ataque. En particular los estudiantes usarán este ataque para ejecutar ataques de man-in-the-middle, donde un atacante puede interceptar y modificar los paquetes entre una víctima A y otra víctima B. El otro objetivo del laboratorio es que los estudiantes practiquen el sniffing y el spoofing de paquetes, dado que son aptitudes esenciales en la seguridad de redes y sirven como base para construir diferentes tipos de ataques de red como así para defenderlas. Para dicho objetivo los estudiantes usarán Scapy en las tareas del laboratorio.

Este laboratorio cubre los siguientes tópicos:

- El Protocolo ARP
- Ataque de ARP Cache Poisoning
- Ataque de Man-in-the-middle
- Programación con Scapy

Videos. Para una cobertura más detallada sobre el protocolo ARP y sus ataques puede consultar:

- Sección 3 del curso de SEED en Udemy, *Internet Security: A Hands-on Approach*, by Wenliang Du. Para más detalles `https://www.handsonsecurity.net/video.html`.

Entorno de Laboratorio. Este laboratorio ha sido testeado en nuestra imagen pre-compilada de una VM con Ubuntu 20.04, que puede ser descargada del sitio oficial de SEED. Sin embargo, la mayoría de nuestros laboratorios pueden ser realizados en la nube para esto Ud. puede leer nuestra guía que explica como crear una VM de SEED en la nube.

2 Configuración del entorno usando Contenedores

Para este laboratorio necesitaremos tres máquinas. Usaremos contenedores para configurar el entorno del laboratorio. La ilustración 1 describe la configuración de nuestro laboratorio. En esta configuración tenemos una máquina de ataque (Host M), que es la que usaremos para lanzar los ataque en contra de las dos

máquinas, Host A y Host B. Estas tres máquinas deben de estar en la misma LAN, dado que el ataque de ARP cache poisoning se limita a una sola LAN.

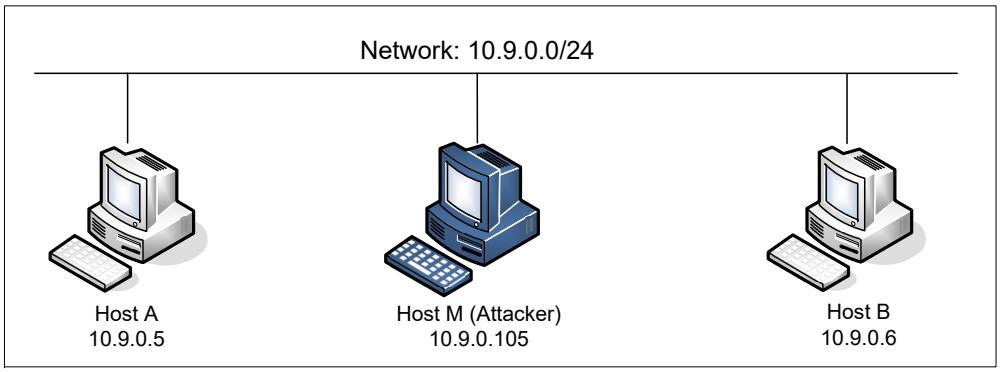

Figure 1: Configuración del entorno

2.1 Setup del Contenedor y sus Comandos

Para empezar a preparar el contenedor, deberá descargarse el archivo `Labsetup.zip` ubicado en el laboratorio correspondiente dentro del sitio web oficial y copiarlo dentro de la Máquina Virtual prevista por SEED. Una vez descargado deberá descomprimirlo y entrar dentro del directorio `Labsetup` donde encontrará el archivo `docker-compose.yml` que servirá para setear el entorno de laboratorio. Para una información más detallada sobre el archivo `Dockerfile` y otros archivos relacionados, puede encontrarla dentro del Manual de Usuario del laboratorio en uso, en el sitio web oficial de SEED.

Si esta es su primera experiencia haciendo el setup del laboratorio usando contenedores es recomendable que lea el manual anteriormente mencionado.

A continuación, se muestran los comandos más usados en Docker y Compose. Debido a que estos comandos serán usados con mucha frecuencia, hemos creados un conjunto de alias para los mismos, ubicados en del archivo `.bashrc` dentro de la Máquina Virtual provista por SEED (Ubuntu 20.04)

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

Dado que todos los contenedores estarán corriendo en un segundo plano. Necesitamos correr comandos para interactuar con los mismos, una de las operaciones fundamentales es obtener una shell en el contenedor. Para este propósito usaremos `"docker ps"` para encontrar el ID del contenedor deseado y ingresaremos `"docker exec"` para correr una shell en ese contenedor. Hemos creado un alias para ello dentro del archivo `.bashrc`

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash
```

```
// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

En caso de problemas configurando el entorno, por favor consulte la sección "Common Problems" en el manual ofrecido por SEED.

2.2 El contenedor de Ataque

Para este laboratorio podemos usar tanto una Máquina Virtual como un contenedor como máquina de ataque. Si observa el archivo Docker Compose, verá que el contenedor de ataque está configurado de forma diferente al resto de los contenedores. Las diferencias son las siguientes:

- *Directorio Compartido.* Cuando usemos el contenedor del atacante para realizar los ataques, necesitamos poner el código de ataque dentro del contenedor.

  La edición del código es más conveniente dentro de la Máquina Virtual que dentro del contenedor, ya que podemos usar nuestro editor de texto preferido. Para que la Máquina Virtual y el contenedor puedan compartir archivos, hemos creado un directorio compartido entre ambos para esto hemos usado `volumes` de Docker. Dentro del archivo de Docker Composer, encontrará que se ha agregado esta entrada en algunos de los contenedores. Esta entrada indica que se montará el diretorio `./volumes` en la Máquina Host (es decir nuestra Máquina Virtual) y se podrá usar dentro del contenedor. Escribiremos nuestro código dentro del directorio `./volumes` (en la Máquina Virtual) y este podrá ser usado en el contenedor.

  ```
  volumes:
         - ./volumes:/volumes
  ```

- *Modo Privilegiado.* Para poder modificar parámetros del kernel en tiempo de ejecución (usando `sysctl`), tal como IP forwarding, el contenedor debe de ser privilegiado. Esto se consigue incluyendo la siguiente entrada dentro del archivo Docker compose del contenedor.

  ```
  privileged: true
  ```

2.3 Sniffing de Paquetes

Poder hacer sniffing de paquetes es algo muy importante para este laboratorio, dado que si las cosas no resultan como se esperan, poder observar el trayecto de estos paquetes nos permitirá identificar determinados problemas. Existen varias formas para hacer sniffing de paquetes:

- Corriendo `tcpdump` en los contenedores. Hemos instalado la utilidad `tcpdump` en cada uno de los contenedores. Para poder sniffear los paquetes que viajan en una interfaz determinada, necesitamos

como primer paso determinar cual es esa interfaz y hacer lo siguiente (assumiendo que el nombre de la interfaz es `eth0`):

```
# tcpdump -i eth0 -n
```

Cabe aclarar que como estamos dentro de un contenedor, Docker implementa un mecanismo de isolación que sólo permite monitorear los paquetes que entran y salen del contenedor en donde se está ejecutando `tcpdump` por lo que no podremos hacer sniffing de paquetes entre contenedores. Sin embargo, si un contenedor usa el modo `host` en su configuración de red, puede sniffear los paquetes de otros contenedores.

- Corriendo `tcpdump` en la Máquina Virtual. Si corremos `tcpdump` dentro de la Máquina Virtual, no estamos sujetos a las mismas restricciones que impone Docker sobre sus contenedores, podemos hacer sniffing sobre todos los paquetes que entran y salen en los contenedores que se encuentran corriendo. La interfaz de red de la Máquina Virtual es diferente que la interfaz de los contenedores. Generalmente en los contenedores cada interfaz empieza con `eth`; en la Máquina Virtual, la interfaz de red creada por Docker empieza con `br-`, seguido por el ID de red. Puede usar el comando `ip address` para obtener el nombre de interfaz de la Máquina Virtual y de los contenedores.
- También podemos usar Wireshark en la Máquina Virtual para sniffear paquetes. Al igual que `tcpdump`, necesitamos seleccionar en que interfaz Wireshark debería de estar escuchando para hacer el sniffing.

3 Tarea 1: ARP Cache Poisoning

El objetivo de esta tarea es poder hacer spoofing de paquetes para lanzar un ataque de ARP cache poisoning sobre una máquina, de tal forma que cuando dos máquinas víctimas A y B se traten de comunicar entre ellas, sus paquetes serán interceptados por el atacante, quién podra realizar cambios sobre los mismos y ser un man in the middle entre A y B. Este tipo de ataque es llamado Man-In-The-Middle (MITM). En este laboratorio usaremos ARP cache poisoning para conducir un ataque MITM.

El siguiente código base muestra como construir un paquete ARP usando Scapy. Por favor reemplace el nombre de la interfaz `br-05f0c56e8085` con la obtenida en su propia máquina (vea la sección anterior).

```
#!/usr/bin/env python3
from scapy.all import *

E = Ether()
A = ARP()

pkt = E/A
sendp(pkt, iface='br-05f0c56e8085')
```

El programa anterior construye y envía un paquete ARP. Por favor complete con los valores necesarios para definir los atributos de su propio paquete ARP. Podemos usar `ls(ARP)` para ver los nombres de los atributos que posee la clase ARP. Si un campo no tiene valor, un valor por defecto será usado (vea la tercera columna del output):

```
$ python3
>>> from scapy.all import *
>>> ls(ARP)
hwtype     : XShortField                          = (1)
ptype      : XShortEnumField                      = (2048)
```

```
hwlen      : ByteField                              = (6)
plen       : ByteField                              = (4)
op         : ShortEnumField                         = (1)
hwsrc      : ARPSourceMACField                      = (None)
psrc       : SourceIPField                          = (None)
hwdst      : MACField                               = ('00:00:00:00:00:00')
pdst       : IPField                                = ('0.0.0.0')
```

En esta tarea, tenemos tres máquinas (contenedores), A, B y M. Queremos atacar la caché ARP de A, de tal manera que la IP de B sea mapeada en la dirección MAC de M. Podemos chequear la cache ARP de una máquina usando el siguiente comandop. Si quiere chequear la cache ARP asociada a una interfaz específica, puede usar el parámetro `-i`.

```
$ arp -n
Address     HWtype  HWaddress           Flags Mask  Iface
10.0.2.1    ether   52:54:00:12:35:00   C           enp0s3
10.0.2.3    ether   08:00:27:48:f4:0b   C           enp0s3
```

Existen muchas formas de conducir un ataque de ARP cache poisoning. Los estudiantes deberán de intentar los siguientes métodos y reportar cuales funcionan y cuales no.

- **Tarea 1.A (Usando ARP Request).** En el Host M, construya un paquete ARP Request y envíelo al Host A. Vea la caché ARP del Host A y si la dirección MAC de M es mapeada en la dirección IP de B.
- **Tarea 1.B (Usando ARP Reply).** En el Host M, construya un paquete ARP Reply y envíelo al Host A. Vea la caché ARP del Host A y si la dirección MAC de M es mapeada en la dirección IP de B. Intente este ataque para los siguientes dos escenarios:
  - Escenario 1: La dirección IP de B existe en la caché de A.
  - Escenario 2: La dirección IP de B no existe en la caché de A.
- **Tarea 1C (Usando ARP gratuitous message).** En el Host M, construya un paquete ARP gratuitous message y envíelo al Host A. Vea la caché ARP del Host A y si la dirección MAC de M es mapeada en la dirección IP de B. Por favor ejecute el ataque usando los dos escenarios que son decriptos en la Tarea 1.B.

  ARP gratuitous es un tipo de paquete especial ARP Request. Este se usa cuando una máquina host necesita actualizar información que quedo desactualizada dentro de la caché ARP del resto de las máquinas. El paquete ARP gratuitous tiene las siguientes características:
  - La dirección IP de origen y destino son las mismas, y pertenecen a la máquina que está emitindo el paquete ARP gratuitous.
  - Tanto en la cabecera ARP como en la cabecera Ethernet, la dirección MAC de destino es la dirección MAC de broadcast (`ff:ff:ff:ff:ff:ff`).
  - No se espera respuesta.

4 Tarea 2: Ataque MITM en Telnet Usando ARP Cache Poisoning

Los Hosts A y B se comunican usando Telnet, el Host M quiere interceptar sus comunicaciones, para poder realizar cambios en los datos enviados entre A y B. Figura 2 Hemos creado una cuenta dentro del contenedor

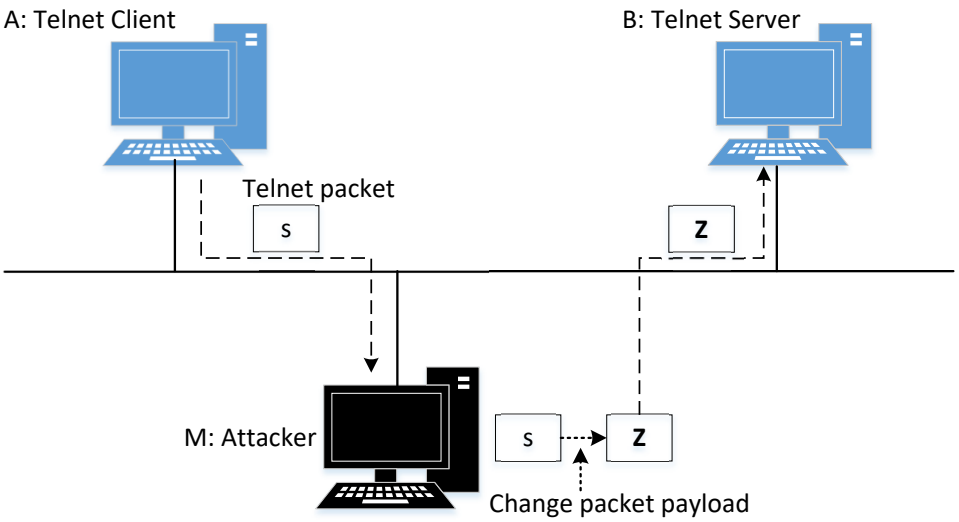

Figure 2: Ataque Man-In-The-Middle Attack contra telnet

llamada "seed", su password es "dees". Puede usar esta cuenta para telnet.

Paso 1 (Lanzar el Ataque ARP cache poisoning). Primero será el Host M quien conducirá el ataque entre las máquinas A y B, de tal forma que en la caché ARP de la máquina A, este mapeada la dirección IP de B que tendrá la dirección MAC de M y en la caché ARP de B, la dirección IP de A también tendrá mapeada la dirección MAC de M. Después de esto, los paquetes traficados entre A y B serán enviados a M. Usaremos el ataque de ARP cache poisoning de la Tarea 1 para lograrlo. Es mejor que envié paquetes spoofeados constantemente (cada 5 segundos); de otra forma las entradas falsas pueden ser reemplazadas por las originales.

Paso 2 (Probando). Después de que el ataque sea exitoso, haga un ping entre los Hosts A y B, reporte sus observacioness. Por favor use los resultados de Wireshark en su reporte. Antes de realizar este pase, asegúrese que el IP Forwarding en el Host M esté desactivado. Para hacerlo debe ejecutar el siguiente comando:

```
# sysctl net.ipv4.ip_forward=0
```

Paso 3 (Active IP Forwarding). Después del Paso 2, activaremos nuevamente el IP Forwarding en el Host, lo que permitirá que se forwardeen los paquetes entre A y B nuevamente. Para hacerlo debe ejecutar el siguiente comando y repita el Paso 2. Por favor describa su observación.

```
# sysctl net.ipv4.ip_forward=1
```

Paso 4 (Lanzar el Ataque MITM). Estamos listos para hacer cambios en los datos de Telnet que se trafican entre A y B. Asuma que A es el cliente telnet y B el servidor telnet. Después de que A se conecta al servidor telnet en B, por cada tecla tipeada en la ventana de telnet de A, se genera un paquete TCP y es enviado a B. Vamos a interceptar este paquete TCP y reemplazar cada caracter tipeado con un caracter fijo (digamos Z). De esta forma, no importa lo que el usuario escriba en A, telnet siempre mostrará Z.

En los pasos anteriores, nos encargamos de redireccionar los paquetes TCP hacia el Host M, pero en vez de redireccionarlos, deberíamos de reemplazarlos con paquetes spoofeados. Escribiremos un programa de sniff-and-spoof para lograr este objetivo. Los pasos a hacer serían los siguientes:

- Primero dejaremos la opción de IP Forwarding activada, por lo que podremos crear una conexión telnet entre A y B. Una vez establecida esta conexión, desactivaremos IP Forward usando el siguiente coando. Por favor escriba algo en la ventana telnet de A y reporte su observación.

```
# sysctl net.ipv4.ip_forward=0
```

- Ejecutamos nuestro programa de sniff-and-spoof en el Host M, para capturar los paquetes que van desde A hacia B, spoofeamos un paquete pero los datos en este paquete TCP serán diferentes. Para los paquetes que van desde B hacia A (la respuesta telnet), no haremos nada, así se conservará el paquete original que será igual al paquete spoofeado.

Para ayudar a los estudiantes, hemos hecho un código base de nuestro programa de sniff-and-spoof. El programa captura todos los paquetes TCP, y realiza algunos cambios en los paquetes que viajan de A hacia B (la parte que realiza esta modificación no está incluída en el código, ya que es parte de la tarea). Para los paquetes que viajan de B hacia A, el programa no realiza ninguna modificación.

```
#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"


def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
         # Create a new packet based on the captured one.
         # 1) We need to delete the checksum in the IP & TCP headers,
         #    because our modification will make them invalid.
         #    Scapy will recalculate them if these fields are missing.
         # 2) We also delete the original TCP payload.

         newpkt = IP(bytes(pkt[IP]))
         del(newpkt.chksum)
         del(newpkt[TCP].payload)
         del(newpkt[TCP].chksum)

         ################################################################
         # Construct the new payload based on the old payload.
         # Students need to implement this part.

         if pkt[TCP].payload:
             data = pkt[TCP].payload.load  # The original payload data
             newdata = data   # No change is made in this sample code

             send(newpkt/newdata)
         else:
```

```
            send(newpkt)
        ###############################################################

    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # Create new packet based on the captured one
        # Do not make any change

        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)


f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Cabe señalar que el código anterior, captura todo los paquetes TCP, incluído el que es generado por el mismo programa. Eso no es conveniente, y afectará a la performance. Los estudiantes necesitan cambiar el filtro para que el programa no capture sus propios paquetes.

Comportamiento de Telnet. Típicamente todo caracter tipeado en una ventana de Telnet dispara un paquete TCP, pero si ud. escribe muy rápido, algunos caracteres pueden ser enviados de manera conjunta en el mismo paquete. Es por eso que el payload de un paquete típico de telnet desde un cliente hacia un servidor contiene solamente un caracter. El caracter enviado será repetido por el servidor y el cliente lo mostrará en su ventana. Lo que vemos en la ventana del cliente no es el resultado directo de lo que se está escribiendo; lo que sea que se escriba en la ventana del cliente realiza un recorrido de ida y vuelta antes de ser mostrado. Si la red está desconectada, lo que se escriba en la ventana del cliente no será mostrado hasta que la red vuelva a estar disponible. Del mismo modo, si los atacantes cambian el caracter a Z durante el ida y vuelta, Z se mostrará en la ventana del cliente Telnet, aunque eso no haya sido lo que se escribió.

5 Tarea 3: Ataque MITM en Netcat Usando ARP Cache Poisoning

Esta tarea es similar a la Tarea 2, con la exepción que los Hosts A y B se comunican usando `netcat`, en lugar de `telnet`. El Host M quiere interceptar sus comunicaciones el Host M quiere interceptar sus comunicaciones, para poder realizar cambios en los datos enviados entre A y B. Puede utilizar los siguientes comandos para establecer una conexión `netcat` TCP entre A y B:

```
On Host B (server, IP address is 10.9.0.6), run the following:
# nc -lp 9090

On Host A (client), run the following:
# nc 10.9.0.6 9090
```

Una vez que la conexión está hecha, puede tipear mensajes en A. Cada linea de mensajes será colocada en un paquete TCP que será enviado a B, quién mostrará el mensaje. Su tarea es reemplazar cada ocurrencia de su nombre dentro del mensaje con una secuencia de letras A. La longitud de la secuencia debería de ser la misma que la longitud de su nombre de lo contrario el número de secuencia TCP se verá corrompido y por ende toda la conexión TCP. Necesita usar su nombre real, de esta forma sabremos que el trabajo fue hecho por ud.

6 Informe del Laboratorio

Debe enviar un informe de laboratorio detallado, con capturas de pantalla, para describir lo que ha hecho y lo que ha observado. También debe proporcionar una explicación a las observaciones que sean interesantes o sorprendentes. Enumere también los fragmentos de código más importantes seguidos de una explicación. No recibirán créditos aquellos fragmentos de códigos que no sean explicados.

Agradecimientos

Este documento ha sido traducido al Español por Facundo Fontana