Laboratorio del Ataque de Mitnick

1 Descripción

Kevin Mitnick es probablemente uno de los hackers más conocidos en USA. Estuvo en la lista de los criminales más buscados del FBI. Mientras huía, se interesó en el hackeo de redes de telefonía celular y tenía la necesidad de un software especializado que lo ayude en esta tarea. Eso lo llevó a Tsutomu Shimomura, un investigador que trabajaba en el San Diego Supercomputer Center y quien era uno de los investigadores principales en la seguridad de las redes de telefonía celular. Tsutomu Shimomura tenía el código que Mitnick estaba necesitando.

En 1994, Mitnick ejecutó un ataque exitoso en la computadora de Shimomura, explotando vulnerabilidades en el protocolo TCP y la relación de confianza entre dos computadores de Shimomura. Este ataque desencadenó una pesecución que eventualmente llevó al arresto de Mitnick. Este suceso fue plasmado en muchos libros y películas de Hollywood tiempo después. Este ataque es conocido como el Ataque de Mitnick, que es un tipo especial de TCP session hijacking.

El objetivo de este laboratorio es recrear el clásico ataque de Mitnick, de esta forma los estudiantes podrán tener experiencia en tal ataque. Emularemos la configuración original de la computadora de Shimomura y lanzaremos el ataque de Mitnick para crear una sesión TCP falsificada entre las dos computadoras de Shimomura. Si el ataque es exitoso, deberíamos de poder correr cualquier comando en la computadora de Shimomura.

Este laboratorio cubre los siguientes tópicos:

- TCP session hijacking
- El Protocolo TCP three-way handshake
- El Ataque de Mitnick
- Shell Remota `rsh`
- Sniffing y Spoofing de Paquetes

Lecturas y Videos. Para una cobertura más detallada sobre TCP session hijacking puede consultar:

- Capítulo 16 del libro de SEED, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. Para más detalles `https://www.handsonsecurity.net`.
- Sección 6 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 Como funciona el Ataque de Mitnick

El ataque de Mitnick es un caso especial de un ataque de TCP session hijacking. En vez de hacer hijacking sobre una conexión TCP entre una víctima A y otra B, el ataque de Mitnick primero crea una conexión TCP entre A y B en su nombre y luego procede a hacer el hijacking de la conexión.

En el ataque de Mitnick real, el host A fue llamado la X-Terminal y era el objetivo. Mitnick quería loguearse dentro de la X-Terminal y ejecutar comandos dentro de esa máquina. El Host B era un servidor de confianza, que permitía loguear dentro de la X-Terminal sin usar un password. Para loguearse dentro de la X-Terminal, Mitnick tuvo que impersonar el servidor de confianza, para que así no tenga que ingresar ningún password. La Figura 1 ilustra el panorama general del ataque. Existen cuatro pasos primarios en este ataque.

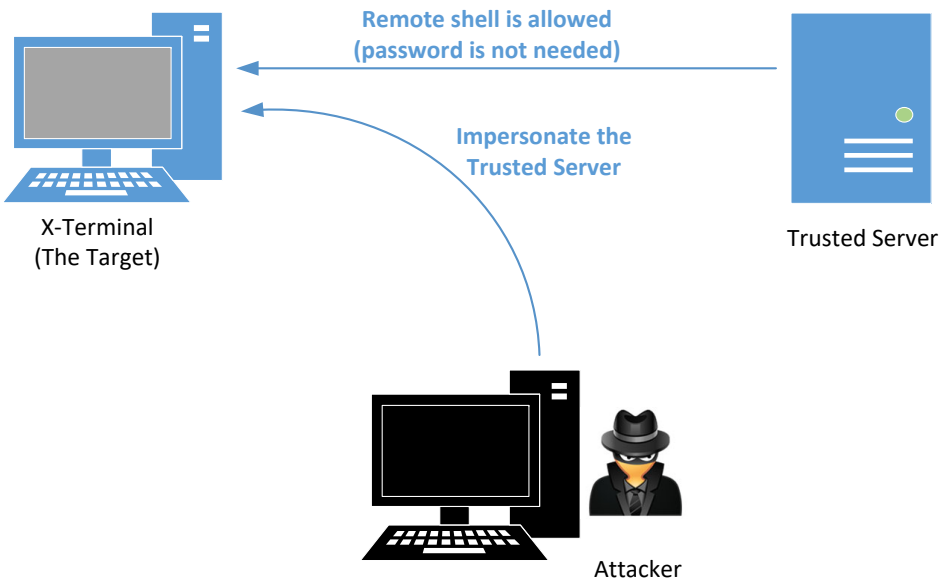

Figure 1: Ilustración del Ataque de Mitnick

Paso 1: Predicción del Número de Secuencia. Antes del ataque, Mitnick necesitaba aprender el patrón del inicio de los números de secuencia (ISN) de la X-Terminal (en esos días los ISN no eran aleatorios). Mitnick enviaba peticiones SYN a la X-Terminal y recibia respuestas SYN+ACK, entonces enviaba un paquete RESET a la X-Terminal, para así limpiar la cola de half-open connections en esta máquina (esto lo hacía para prevenir que la cola se llene). Después de repetir esto veinte veces. Encontro que había un patrón entre dos TCP ISNs sucesivos. Esto le permitió a Mitnick predecir el ISN, que era un elemento esencial para el ataque.

Paso 2: Ataque de SYN flooding en el Servidor de Confianza. Para enviar una petición de conexión desde el Servidor de Confianza hacia la X-Terminal, Mitnick necesitaba enviar un paquete SYN desde el Servidor de Confianza hacia la X-Terminal. Entonces la X-Terminal respondería un paquete SYN+ACK que se enviaría al Servidor de Confianza. Dado que este Servidor de Confianza no era el que iniciaba la petición, enviaría un paquete RESET a la X-Terminal para detener el protcolo de 3-way handshake. Este comportamiento era problemático para el ataque de Mitnick. Para resolver este problema. Mitnick debía de

silenciar al Servidor de Confianza. Para esto, antes de hacer el spoofing, Mitnick lanzaba un ataque de SYN flooding sobre el servidor. En esa época los servidores eran mucho más vulnerables a este tipo de ataques. El ataque podría dejar a la máquina fuera de servicio, silenciándola por completo.

Paso 3: Spoofear la Conexión TCP. Mitnick quería usar `rsh` (shell remota o remote shell) para ejecutar un comando a modo de backdoor en la X-Terminal; una vez que el backdoor haya sido instalado, podría loguearse dentro de la X-Terminal. Para correr una shell remota en la X-Terminal, Mitnick necesitaba poder autenticarse, es de cir necesitaba tener una cuenta válida en la X-Terminal y saber su password. Obviamente, no tenía esta información.

Shimomura solía loguearse a menudo dentro de la X-Terminal desde el Servidor de Confianza. Para evitar tipear el password cada vez que entraba, agregó una entrada en el archivo `.rhosts` dentro de la X-Terminal, por lo que al loguearse dentro de la X-Terminal desde el Servidor de Confianza, no se le pediría ningún password. En ese entonces esta era una práctica común. En este escenario Shimomura podía correr cualquier comando en la X-Terminal desde el Servidor de Confianza usando `rsh`, o corriendo `rlogin` para loguearse en la X-Terminal, sin la necesidad de tipear un password. Mitnick quería explotar esta brecha.

Mitnick necesitaba crear una conexión TCP entre el servidor de confianza y la X-Terminal y luego correr `rsh` dentro de esta conexión Primero envió una petición SYN hacia la X-Terminal, usando la dirección IP del servidor de confianza como la dirección IP de origen. La X-Terminal respondió con un SYN+ACK hacia el servidor. Dado que el servidor se encontraba fuera de servicio, no enviaría un RESET para cerrar la conexión.

Para completar el proceso de 3-way handshake, Mitnick necesitaba spoofear un paquete ACK, que debía de confirmar el número de secuencia en el paquete SYN+ACK de la X-Terminal. Desafortunadamente la respuesta SYN+ACK sólo iba hacia el servidor de confianza y no hacia Mitnick, él no podía ver cual era este número de secuencia. Sin embargo, debido a investigaciones previas, Mitnick podía predecir cual podría ser este número, por lo que podía enviar un paquete ACK spoofeado hacia la X-Terminal para completar el 3-way handshake de forma exitosa.

Paso 4: Correr una shell remota. Usando la conexión TCP establecida entre el servidor de confianza y la X-Terminal, Mitnick podía enviar una petición de shell remota a la X-Terminal, con el objetivo de ejecutar un comando. Usando este comando, Mitnick quería plantar un backdoor en la X-Terminal que le otorgará una shell de forma persistente para no tener que volver a repetir el flujo de ataque.

Todo lo que necesitaba era agregar `"+ +"` en el archivo `.rhosts` dentro la X-Terminal. Pudo hacerlo ejecutando el siguiente comando usando `rsh` en la X-Terminal: `"echo + + > .rhosts"`. Dado que los programas `rsh` y `rlogin` usaban el archivo `.rhosts` para autenticación, con este agregado de Mitnick, la X-Terminal confiaría en cualquier petición `rsh` y `rlogin` sin importar de quien sea o su dirección IP.

3 Setup del Entorno de Laboratorio usando Contenedores

3.1 Setup del Contenedor

En este laboratorio, necesitamos tres máquinas, una para la X-Terminal, la otra para el Servidor de Confianza y la otra será la del atacante. En el escenario real del ataque de Mitnick, la máquina del atacante era una máquina remota. En este laboratorio, por un tema de simplicidad, hemos puesto estas tres máquinas en la misma red. Los estudiantes pueden usar tres máquina virtuales por separado para hacer el laboratorio, pero es mucho mas conveniente usar los contenedores. El entorno de laboratorio se muestra en la Figura 2.

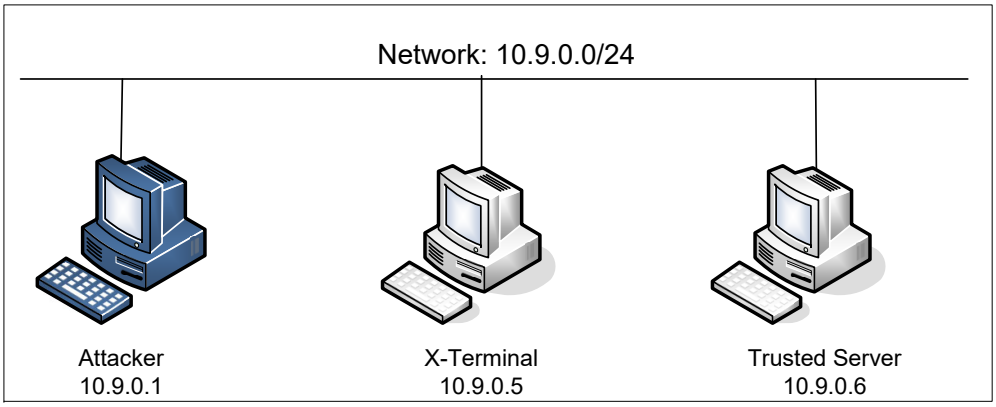

Figure 2: Setup del Entorno de Laboratorio

3.2 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.

3.3 Sobre el Contenedor del Atacante

I 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.

- *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
  ```

- *Host Mode.* En este laboratio, el atacante va a necesitar sniffear los paquetes, pero correr el programa de sniffing dentro del contenedor del atacante tiene sus inconvenientes, ya que el contenedor está atachado a un switch virtual y sólo podrá ver su propio tráfico y no el del resto de los contenedores. Para solucionar este problema, usaremos el modo `host` para el contenedor del atacante. Esto permite que el contenedor del atacante vea el tráfico de toda la red. La siguiente entrada es usada para el contenedor del atacante:

  ```
  network_mode: host
  ```

  Cuando un contenedor está en modo `host`, este puede ver todas las interfaces de red de los hosts que la componen, inclusive tiene la misma dirección IP como si fuera el host principal. Básicamente es ponerlo en el mismo espacio de red como si fuera la Máquina Virtual de Host. Sin embargo, el contenedor sigue siendo una máquina diferente.

- *Privileged Mode.* 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
  ```

3.4 Instalando `rsh`

Remote shell `rsh` es un programa de consola que sirve para ejecutar comandos shell de manera remota. Aunque usaremos `rsh` en esta tarea, deberíamos de saber que `rsh` y `rlogin` no son seguros y ya no se usan mas. Han sido reemplazados por programas más seguros como `ssh`. Debido a esto en los sistemas operativos linux actuales, el comando `rsh` es un link simbólico al programa `ssh`.

```
$ ls -al /etc/alternatives | grep rsh
lrwxrwxrwx   1 root root    12 Jul 25  2017 rsh -> /usr/bin/ssh
```

Para recrear el ataque de Mitnick, necesitamos instalar la version insegura del programa `rsh`. Obviamente la vieja versión de `rsh` ya no funciona, pero existe un proyecto de código abierto que implementa el cliente y el servidor remoto de shell. Podemos usar los siguientes comandos para instalar el servidor y el cliente `rsh`: **Nota:** Los programas `rsh` ya fueron instalados en los contenedores X-Terminal y el Servidor de Confianza (vea dentro del archivo `Dockerfile` dentro del directorio de las imágenes).

```
$ sudo apt-get install rsh-redone-client
$ sudo apt-get install rsh-redone-server
```

3.5 Configuración

El servidor `rsh` usa dos archivos para autenticación, `.rhosts` y `/etc/hosts.equiv`. Cada vez que el servidor recibe una petición de comando de manera remota, chequeará the `/etc/hosts.equiv`. Si la petición proviene de un hostname que se encuentra dentro de este archivo, el servidor la aceptará sin pedir password. Si `/etc/hosts.equiv` no existe o no contiene ese hostname, `rsh` chequeará el archivo `.rhosts` en el directorio home del usuario.

Shimomura necesitaba correr con frecuencia comandos remotos en la X-Terminal desde el servidor de confianza. Para evitar ingresar un password cada vez que le era necesario, creo el archivo `.rhosts` en la X-Terminal y puso la dirección IP del servidor dentro del mismo. Note que el archivo `.rhosts` debe de ubicarse en la raíz del directorio home del usuario y puede ser escrito solamente por el **usuario propietario**.

Por favor ejecute los siguientes comandos en la X-Terminal para hacer el setup del archivo `.rhosts`. Cabe señalar que cuando entramos en este contenedor, lo haremos con una cuenta de root. En este laboratorio necesitamos cambiar a una cuenta de usuario normal llamada seed, que fue creada dentro del contenedor:

```
# su seed          ← Switch to the seed account
$ cd               ← Go to seed's home directory
$ touch .rhosts    ← Create an empty file
$ echo [Server's IP address] > .rhosts
$ chmod 644 .rhosts
```

Para verificar su configuración, intente ejecutar el siguiente comando en el servidor de confianza.

```
# su seed          ← Switch to the seed account
$ rsh [X-Terminal's IP] date
```

Si el comando despliega en pantalla la fecha y hora actual, su configuración esta funcionando. Si ud. ve "Authentication Failure", quiere decir que fue mal configurado. Un error común es el permiso en el archivo `.rhosts`: debería de asegurarse que solamente el usuario propietario posea permisos de escritura sobre el mismo.

Permitir a Todos. Para hacer que cualquier usuario con cualquier dirección IP pueda ejecutar comandos en la X-Terminal, necesitamos poner dos signos de suma (`"+ +"`) dentro del archivo `.rhosts`. Esto es algo peligroso y nadie debería de hacerlo. Pero si ud. es el atacante, es una forma conveniente de setear un backdoor. Como hemos mencionado anteriormente, esto es lo que se usó en el ataque de Mitnick.

4 Tarea 1: SYN flooding Simulado

En la época del ataque de Mitnick, los sistemas operativos eran vulnerables a ataques de SYN flooding, que ocasionaban que una máquina remota quede fuera de servicio o sin responder. Hoy en día con el avance de los sistemas operativos modernos, los ataques de SYN flooding no causan semejante daño. Haremos una simulación de los daños de este tipo de ataque.

Podemos detener el contenedor del servidor de confianza manualmente, pero eso no es suficiente. Cuando la X-Terminal recibe un paquete SYN del servidor de confianza, responderá con un paquete SYN+ACK. Antes de enviar este paquete, necesita saber la dirección MAC del servidor de confianza. La caché ARP será chequeada primero. Si no existe ninguna entrada para el servidor de confianza, la X-Terminal enviará un petición ARP para preguntar por esa dirección MAC. Dado que el servidor confianza ha sido silenciado, nadie va a responder esa petición ARP, por ende X-Terminal no podrá enviar la respuesta. Como resultado la conexión TCP no podrá ser establecida.

En el ataque real, la dirección MAC del servidor de confianza se encontraba dentro de la caché ARP de la X-Terminal. Incluso si no lo estaba, antes de dejar inhabilitado al servidor de confianza, podíamos spoofear una petición echo ICMP desde el servidor de confianza hacia la X-Terminal, lo que haría que la X-Terminal respondiera al servidor de confianza y obtendríamos la dirección MAC del mismo y la guardaríamos en la caché.

Para simplificar la tarea, antes de detener el servidor de confianza, haremos un ping de la X-Terminal una sola vez, y usaremos el comando `arp` para asegurarnos que la dirección MAC se encuentre en la caché. Cabe señalar que esta entrada puede ser removida por el sistema operativo si este falla a la hora encontrar el destino usando la dirección MAC que está en la caché. Para simplificar su ataque, puede ejecutar el siguiente comando en la X-Terminal para agregar una entrada de manera permanente en la caché ARP (necesita hacerlo como usuario root):

```
# arp -s [Server's IP] [Server's MAC]
```

5 Tarea 2: Spoofear Conexiones TCP y Sesiones `rsh`

Ahora que hemos "desactivado" al servidor de confianza, podemos suplantarlo y tratar de lanzar una sesión `rsh` con X-Terminal. Dado que `rsh` corre sobre TCP, primero necesitamos establecer una conexión TCP entre el servidor de confianza y la X-Terminal y recién después correr `rsh` sobre esta conexión.

Una de las dificultades en el ataque de Mitnick es predecir los números de secuencia TCP. En aquel tiempo era posible cuando los números de secuencia TCP no eran generados de forma aleatoria. Hoy en día los sistemas operativos randomizan el proceso de generación de los números de secuencia TCP (como medida de protección contra ataques de TCP session hijacking), por lo que predecirlos se vuelve imposible. Para simular la situación del ataque original, los estudiantes pueden sniffear los paquetes y obtener los números de secuencia, en vez de adivinarlos.

Restricción. Para poder simular el ataque original de Mitnick lo más parecido posible al que fue en ese tiempo, los estudiantes pueden sniffear paquetes TCP de la X-Terminal, pero no pueden usar todos los

campos de estos paquetes, porque en los ataques reales Mitnick no pudo sniffear paquetes. Cuando los estudiantes escriban sus programas de ataque, solamente podrán usar los siguientes campos de los paquetes capturados, de forma contraria se aplicará una pena.

- **Campo de Número de Secuencia TCP** (Esto no incluye el campo de Acknowledgment).
- **Campo flag TCP**. Este nos permite saber que tipo de paquete TCP es. En el ataque real de Mitnick, Mitnick sabía que tipo de paquetes eran enviados por la X-Terminal, porque eran parte del protocolo de 3way-handshake. Para simplificar el ataque se les permite a los estudiantes usar este campo.
- **Todos los campos de Longitud**, incluídos el campo de longitud del encabezado IP, la longitud total IP y la longitud del encabezado TCP. Estas piezas de información no son necesarias para los ataques. En el ataque de Mitnick , Mitnick sabía exactamente cuales eran estos valores. Para simplificar el ataque se les permite a los estudiantes usar estos campos.

El Comportamiento `rsh`. Para crear una sesión `rsh` spoofeada entre el servidor de confianza y la X-Terminal, necesitamos entender el comportamiento de `rsh`. Vamos a crear una sesión `rsh` desde el servidor de confianza hacia la X-Terminal y usaremos Wireshark para capturar los paquetes enviados entre ellos (Nota: correremos Wireshak desde la máquina virtual del atacante; asegurése de seleccionar la interfaz de red correcta que corresponde a la red `10.9.0.0/24`). Usaremo el siguiente comando para ejecutar el comando `date` en el Host B desde el Host A a través de `rsh`.

```
// On Trusted Server
$ rsh 10.9.0.5 date
```

El packet trace de la sesión `rsh` es motrado a continuación. En este entorno `10.9.0.6` es la dirección IP del servidor de confianza y `10.9.0.5` es la dirección IP de la X-Terminal. Si el paquete no lleva ningún dato TCP, la información de longitud (es decir `Len=0`) es omitida.

Listing 1: Packet trace de una sesión `rsh`

```
# The first connection
   SRC IP    DEST IP   TCP Header
1  10.9.0.6  10.9.0.5  1023 -> 514 [SYN] Seq=778933536
2  10.9.0.5  10.9.0.6  514 -> 1023 [SYN,ACK] Seq=10879102 Ack=778933537
3  10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933537 Ack=10879103
4  10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933537 Ack=10879103 Len=20
                       RSH Session Establishment
                       Data: 1022\x00seed\x00seed\x00date\x00
5  10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879103 Ack=778933557

# The second connection
6  10.9.0.5  10.9.0.6  1023 -> 1022 [SYN] Seq=3920611526
7  10.9.0.6  10.9.0.5  1022 -> 1023 [SYN,ACK] Seq=3958269143 Ack=3920611527
8  10.9.0.5  10.9.0.6  1023 -> 1022 [ACK] Seq=3920611527 Ack=3958269144


# Going back to the first connection
9  10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879103 Ack=778933557 Len=1
                       Data: \x00
10 10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933557 Ack=10879104
11 10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879104 Ack=778933557 Len=29
                       Data: Sun Feb 16 13:41:17 EST 2020
```

Podemos observar que una sesión `rsh` consiste en dos conexiones TCP. La primera conexión es iniciada por el Host A (el cliente). El proceso `rshd` en el Host B está a la escucha de peticiones entrantes en el puerto 514. Los paquetes del 1 a 3 son para del protocolo del 3way-handshake. Después de que la conexión ha sido establecida, el cliente envía los datos `rsh` (incluídos los IDs de usuario y los comandos) al Host B (paquete 4). El proceso `rshd` autenticará al usuario y si este es autenticado de forma exitosa, `rshd` inicia una conexión TCP separada con el cliente.

La segunda conexión es usada para enviar mensajes de error. En la traza anterior, dado que no hubo errores, esta conexión no fue usada, pero la conexión debe ser establecida o `rshd` no podrá continuar. Los paquetes del 6 al 7 son para el protocolo de 3way-handshake de la segunda conexión.

Después de que la segunda conexión ha sido establecida, el Host B enviará un zero byte al cliente (usando la primera conexión) el Host A aceptará este paquete. Después de eso, el proceso `rshd` en el Host B ejecutará el comando enviado por el cliente y su salida será enviada de regreso al cliente, usando la primera conexión. Los estudiantes pueden usar Wireshak para capturar la sesión `rsh` y estudiar sus comportamientos antes de lanzar el ataque de Mitnick. Hemos dividido el ataque en dos sub-tareas, cada una se centra en una conexión.

5.1 Tarea 2.1: Spoofear la primera Conexión TCP

La primera conexión TCP es iniciada por el atacante por medio de un paquete SYN spoofeado. Como puede ver en la Figura 3, después de que la X-Terminal recibe el paquete SYN, enviará un paquete SYN+ACK al servidor de confianza. Dado que el servidor se encuentra fuera de línea, no reseteará la conexión. El atacante, quien esta en la misma red, puede sniffear los paquetes y obtener el número de secuencia.

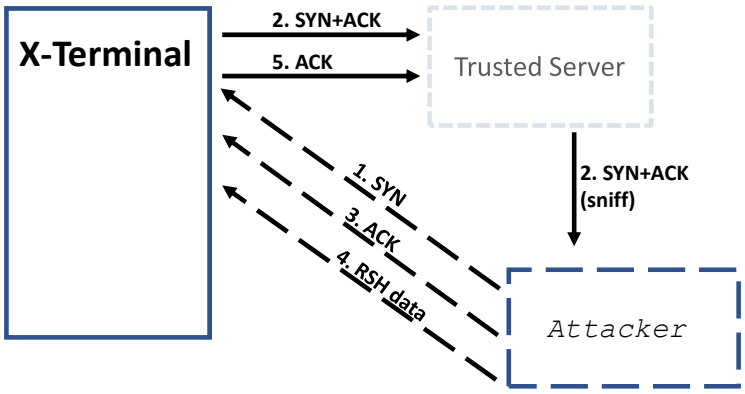

Figure 3: Primera Conexión

Paso 1: Spoofear un paquete SYN. Los estudiantes deberían de escribir un programa para spoofear un paquete SYN desde servidor de confianza hacia la X-Termina (Vea Paquete 1 en el Listado 1). Hay seis bits que representan códigos standard en el protocolo TCP, y pueden ser seteadas a través del campo flag del encabezado TCP. El siguiente ejemplo de código muestra como setear el campo flag y como chequear si ciertos bits están presentes dentro de este campo.

```
# 'U': URG bit
# 'A': ACK bit
# 'P': PSH bit
# 'R': RST bit
# 'S': SYN bit
```

```
# 'F': FIN bit

tcp = TCP()

# Set the SYN and ACK bits
tcp.flags = "SA"

# Check whether the SYN and ACK are the only bits set
if tcp.flags == "SA":

# Check whether the SYN and ACK bits are set
if 'S' in tcp.flags and 'A' in tcp.flags:
```

Cabe señalar que el puerto de origen del paquete SYN debe ser del puerto `1023`. Si se usa un puerto diferente `rsh` reseteará la conexión después de que la conexión fue establecida. Si este paso es exitoso deberíamos de ver el paquete SYN+ACK que viene de la X-Terminal usando Wireshark (Vea Paquete 2 del Listado 1).

Paso 2: Responder al paquete SYN+ACK. Después de que la X-Terminal envía el SYN+ACK, el servidor de confianza necesita enviar un paquete ACK para completar el protocolo de 3way-handshake. El número de acknowledge en este paquete debería de ser `S+1`, donde `S` es el número de secuencia que se encuentra en el paquete SYN+ACK.

En el ataque real de Mitnick, el atacante no podía ver el paquete SYN+ACK, porque era enviado hacia el servidor de confianza y no hacia el atacante. Esto es el porque Mitnick tuvo que adivinar el valor del número de secuencia. En este laboratorio, a los estudiantes se les permitirá obtener el número de secuencia haciendo sniffing de paquetes.

Los estudiantes deben de escribir un programa de sniff-and-spoff usando `Scapy` y ejecutarlo en la máquina del atacante. A continuación se muestra el código base para el programa de sniff-and-spoof que podría ser de utilidad. Por favor asegúrese de seguir las restricciones descriptas al inicio de la sección o será penado.

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

x_ip      = "10.9.0.5"  # X-Terminal
x_port    = 514         # Port number used by X-Terminal

srv_ip    = "10.9.0.6"  # The trusted server
srv_port  = 1023        # Port number used by the trusted server

# Add 1 to the sequence number used in the spoofed SYN
seq_num     = 0x1000 + 1


def spoof(pkt):
  global seq_num   # We will update this global variable in the function

  old_ip  = pkt[IP]
  old_tcp = pkt[TCP]

  # Print out debugging information
```

```
  tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4  # TCP data length
  print("{}:{} -> {}:{}  Flags={} Len={}".format(old_ip.src, old_tcp.sport,
                        old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))



  # Construct the IP header of the response
  ip = IP(src=srv_ip, dst=x_ip)

  # Check whether it is a SYN+ACK packet or not;
  #   if it is, spoof an ACK packet

  # ... Add code here ...

myFilter = 'tcp'   # You need to make the filter more specific
sniff(iface='br-****', filter=myFilter, prn=spoof)
                  ↖ You need to set the correct value here.
```

Paso 3: Spoofear el paquete de datos de `rsh`. Una vez establecida la conexión, el atacante necesita enviar datos por medio de `rsh` hacia la X-Terminal. La estructura de los datos `rsh` se muestran a continuación.

```
[port number]\x00[uid_client]\x00[uid_server]\x00[your command]\x00
```

Esta estructura tiene cuatro partes: un número de puerto, un user ID de cliente, un user ID de servidor y un comando. El número de puerto será usado en la segunda conexión (Vea la Tarea 2.2). Tanto los IDs del cliente y del servidor tienen el valor de `seed` en nuestro contenedor. Los cuatro campos son separados por un byte 0. Note que hay un byte 0 al final de los datos `rsh`. Un ejemplo de esto se muestra abajo. En este ejemplo, le decimos a la X-Terminal que vamos a escuchar en el puerto 9090 para la segunda conexión y el comando que queremos correr es `"touch /tmp/xyz"`.

```
data = '9090\x00seed\x00seed\x00touch /tmp/xyz\x00'
send(IP()/TCP()/data, verbose=0)
```

Los estudiantes deberían de modificar el programa de sniff-and-program escrito en el Paso 2, para enviar un paquete de datos `rsh` a la X-Terminal (Vea Paquete 4 en el Listado 1). Si este paso es exitoso, desde Wireshark, podremos ver que la X-Terminal va a iniciar una conexión TCP hacia el puerto del servidor de confianza `9090`, que es especificado dentro de los datos de `rsh`.

Por favor describa en su informe del laboratorio, si el comando `touch` ha sido ejecutado en la X-Terminal o no. Por favor incluya capturas de pantalla de su Wireshark.

5.2 Paso 2.2: Spoofear la segunda Conexión TCP

Después de que la primera conexión ha sido establecida, la X-Terminal iniciará la segunda conexión. Esta conexión es usada por `rshd` para enviar mensajes de errores. En nuestro ataque no usaremos esta conexión pero si esta conexión no se establece, `rshd` se detendrá sin ejecutar nuestro comando. Ademas, necesitamos de la ayuda del spoofing para que la X-Terminal y el servidor de confianza, terminen de establecer esta conexión. Vea la Figura 4.

Los estudiantes necesitan escribir otro programa de sniff-and-program, el cual sniffea el tráfico TCP que se dirige al puerto 9090 del servidor de confianza (asumiendo que se uso el puerto `9090` de la Tarea

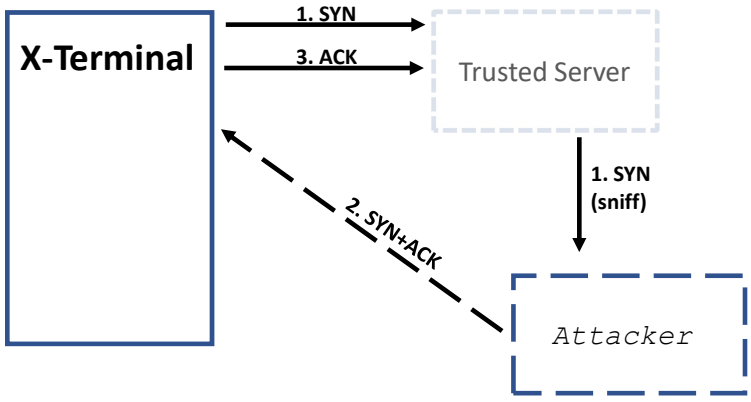

Figure 4: Segunda Conexión

2.1). Cuando este vea el paquete SYN, debería de responder con un paquete SYN+ACK. Vea Paquete 7 del Listado 1 a modo de ejemplo.

Si ambas conexiones son establecidas de forma exitosa, `rshd` ejecutará el comando contenido en el paquete de datos `rsh`. Por favor verifique que dentro del directorio `/tmp` que se haya creado el archivo `/tmp/xyz` y que su fecha creación coincida con el tiempo actual. Por favor incluya evidencia en su informe del laboratorio.

6 Tarea 3: Plantar el Backdoor

En nuestro ataque de la Tarea 2 solamente hemos corrido el comando `touch` para probar que podemos ejecutar un comando de forma exitosa en la X-Terminal. Si queremos ejecutar más comandos en un futuro, podemos lanzar el mismo ataque. Esto no es muy conveniente.

Mitnick hizo un plan que le permitia volver a la X-Terminal. En vez de lanzar el ataque una y otra vez, instaló un backdoor en la X-Terminal después de haber hecho su primer ataque. Este backdoor le permitía loguearse dentro de la X-Terminal de forma normal en cualquier momento que el quisiera sin necesida de escribir un password. Para lograr este objetivo, como hemos discutido en la Sección 3.5, todo lo que debemos de hacer es agregar la siguiente cadena `"+ +"` en el archivo `.rhosts` (en una sola línea). Podemos incluir el siguiente comando en nuestros datos `rsh`.

```
echo + + > .rhosts
```

Los estudiantes deberían de reemplazar el comando `touch` de la Tarea 2 con el comando `echo` mostrado arriba y lanzar el ataque nuevamente. Si el ataque funciona, el atacante debería de poder loguearse remotamente en la X-Terminal usando el comando que se muestr más abajo y sin necesidad de tipear ningún password:

```
$ rsh [X-Terminal's IP]
```

El programa `rsh` puede que no este instalado en el contenedor del atacante, para instalarlo puede ingresar los siguientes comandos:

```
# apt-get update && apt-get -y install rsh-redone-client
```

7 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