

Informe Laboratorio 5

Sección 1

Sebastián Riquelme
e-mail: sebastian.riquelme1@mail.udp.cl

Junio de 2023

Índice

1. Descripción de actividades	2
2. Desarrollo (Parte 1)	4
2.1. Códigos de cada Dockerfile	4
2.1.1. S1	4
2.1.2. C1	5
2.1.3. C2	5
2.1.4. C3	5
2.1.5. C4	5
2.2. Creación de las credenciales para S1	6
2.3. Tráfico generado por C1 (detallado)	6
2.4. Tráfico generado por C2 (detallado)	6
2.5. Tráfico generado por C3 (detallado)	7
2.6. Tráfico generado por C4 (4 iface lo) (detallado)	7
2.7. Diferencia entre C1 y C2	8
2.8. Diferencia entre C2 y C3	9
2.9. Diferencia entre C3 y C4	9
3. Desarrollo (Parte 2)	10
3.1. Identificación del cliente ssh	10
3.2. Replicación de tráfico (paso por paso)	10

1. Descripción de actividades

Para este último laboratorio, nuestro informante ya sabe que puede establecer un medio seguro sin un intercambio previo de una contraseña, gracias al protocolo diffie-hellman. El problema es que ahora no sabe si confiar en el equipo con el cual establezca comunicación, ya que las credenciales de usuario pueden haber sido divulgadas por algún soplón.

Para el presente laboratorio deberá:

- Crear 4 contenedores en Docker, donde cada uno tendrá el siguiente SO: Ubuntu 14.10, Ubuntu 16.10, Ubuntu 18.10 y Ubuntu 20.10, a los cuales llamaremos C1,C2,C3,C4/S1 respectivamente.
- Para cada uno de ellos, deberá instalar la última versión, disponible en sus repositorios, del cliente y servidor openssh.
- En S1 deberá crear el usuario test con contraseña test, para acceder a él desde los otros contenedores.
- En total serán 4 escenarios, donde cada uno corresponderá a los siguientes equipos:
 - C1 → S1
 - C2 → S1
 - C3 → S1
 - C4 → S1

Pasos:

1. Para cada uno de los 4 escenarios, solo deberá establecer la conexión y no realizar ningún otro comando que pueda generar tráfico (como muestra la Figura). Deberá capturar el tráfico de red generado y analizar el patrón de tráfico generado por cada cliente. De esta forma podrá obtener una huella digital para cada cliente a partir de su tráfico.

Indique el tamaño de los paquetes del flujo generados por el cliente y el contenido asociado a cada uno de ellos. Luego, indique qué información distinta contiene el escenario siguiente (diff incremental). El objetivo de esta tarea es identificar claramente los cambios entre las distintas versiones de ssh.

2. Para poder identificar que el usuario efectivamente es el informante, éste utilizará una versión única de cliente. ¿Con qué cliente SSH se habrá generado el siguiente tráfico?

Protocol	Length	Info
TCP	74	34328 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=14
TCP	66	34328 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0
SSHv2	85	Client: Protocol (SSH-2.0-OpenSSH_?)
TCP	66	34328 → 22 [ACK] Seq=20 Ack=42 Win=64256 Len=
SSHv2	1578	Client: Key Exchange Init
TCP	66	34328 → 22 [ACK] Seq=1532 Ack=1122 Win=64128
SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exc
TCP	66	34328 → 22 [ACK] Seq=1580 Ack=1574 Win=64128
SSHv2	82	Client: New Keys
SSHv2	110	Client: Encrypted packet (len=44)
TCP	66	34328 → 22 [ACK] Seq=1640 Ack=1618 Win=64128
SSHv2	126	Client: Encrypted packet (len=60)
TCP	66	34328 → 22 [ACK] Seq=1700 Ack=1670 Win=64128
SSHv2	150	Client: Encrypted packet (len=84)
TCP	66	34328 → 22 [ACK] Seq=1784 Ack=1698 Win=64128
SSHv2	178	Client: Encrypted packet (len=112)
TCP	66	34328 → 22 [ACK] Seq=1896 Ack=2198 Win=64128

Figura 1: Tráfico generado del informante

Replique este tráfico generado en la imagen. Debe generar el tráfico con la misma versión resaltada en azul.

3. Para que el informante esté seguro de nuestra identidad, nos pide que el patrón del tráfico de nuestro server también sea modificado, hasta que el Key Exchange Init del server sea menor a 300 bytes. Indique qué pasos realizó para lograr esto.

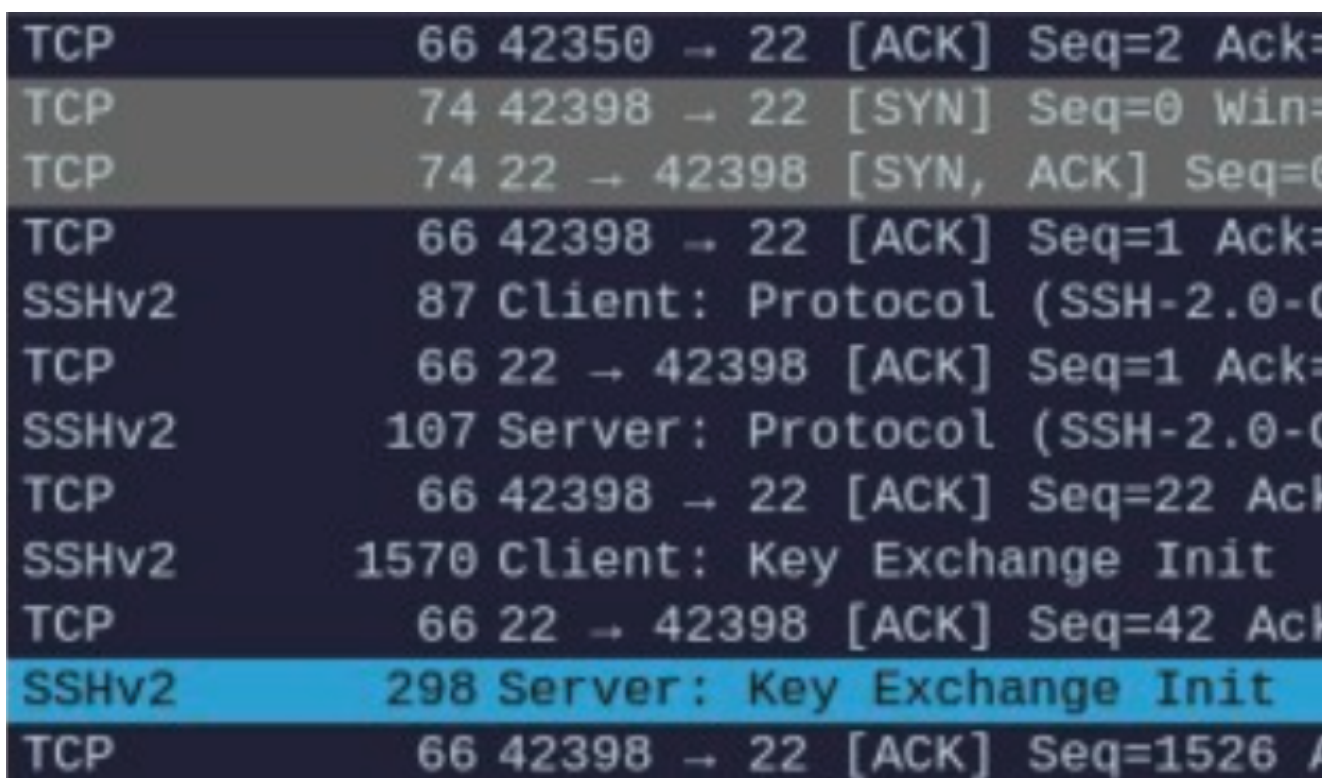


Figura 2: Captura del Key Exchange

2. Desarrollo (Parte 1)

2.1. Códigos de cada Dockerfile

2.1.1. S1

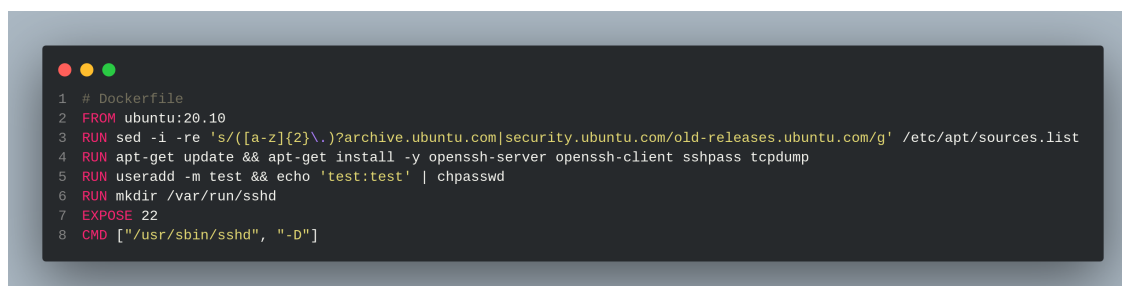
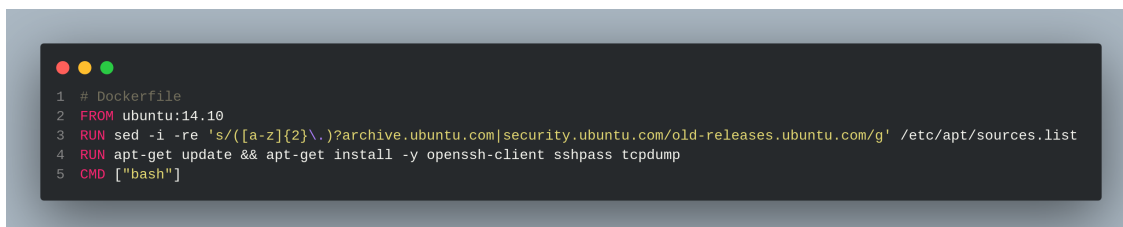


Figura 3: Dockerfile de s1.

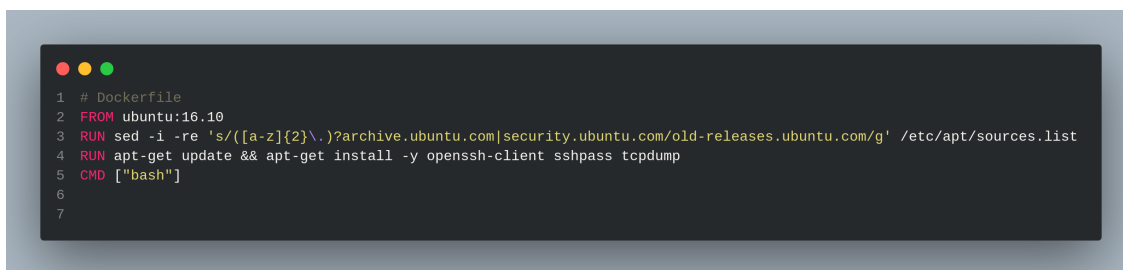
2.1.2. C1



```
1 # Dockerfile
2 FROM ubuntu:14.10
3 RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4 RUN apt-get update && apt-get install -y openssh-client sshpass tcpdump
5 CMD [\"bash\"]
```

Figura 4: Dockerfile de c1.

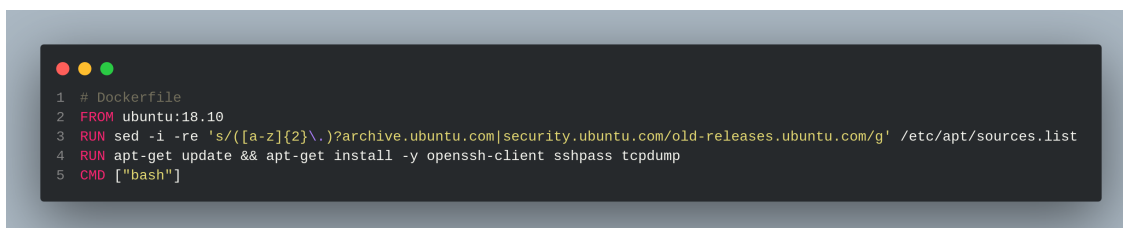
2.1.3. C2



```
1 # Dockerfile
2 FROM ubuntu:16.10
3 RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4 RUN apt-get update && apt-get install -y openssh-client sshpass tcpdump
5 CMD [\"bash\"]
6
7
```

Figura 5: Dockerfile de c2.

2.1.4. C3



```
1 # Dockerfile
2 FROM ubuntu:18.10
3 RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4 RUN apt-get update && apt-get install -y openssh-client sshpass tcpdump
5 CMD [\"bash\"]
```

Figura 6: Dockerfile de c3.

2.1.5. C4

El Dockerfile es el mismo que S1.

2.2. Creación de las credenciales para S1

2.3. Tráfico generado por C1 (detallado)

1. **Client: Protocol (SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-8)**: Este es el inicio de la sesión SSH, donde el cliente C1 se está comunicando con el servidor utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 102 bytes.
2. **Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)**: El servidor responde al cliente confirmando que también está utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.
3. **Client: Key Exchange Init**: El cliente inicia el intercambio de claves para establecer una sesión segura. Este paquete es significativamente más largo, con una longitud de 2036 bytes, lo que probablemente se deba a la información de la clave que se está enviando.
4. **Server: Key Exchange Init**: El servidor responde con su propio paquete de inicio de intercambio de claves, que tiene una longitud de 1124 bytes.
5. **Client: Elliptic Curve Diffie-Hellman Key Exchange Init**: El cliente inicia el intercambio de claves utilizando el algoritmo de intercambio de claves Diffie-Hellman de curva elíptica (ECDH). Este paquete tiene una longitud de 116 bytes.
6. **Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys**: El servidor responde al cliente, completando el intercambio de claves ECDH y estableciendo nuevas claves para la sesión. Este paquete tiene una longitud de 348 bytes.

2.4. Tráfico generado por C2 (detallado)

1. **Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)**: Este es el inicio de la sesión SSH, donde el cliente C2 se está comunicando con el servidor utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.
2. **Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)**: El servidor responde al cliente confirmando que también está utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.
3. **Client: Key Exchange Init**: El cliente inicia el intercambio de claves para establecer una sesión segura. Este paquete es significativamente más largo, con una longitud de 1500 bytes, lo que probablemente se deba a la información de la clave que se está enviando.
4. **Server: Key Exchange Init**: El servidor responde con su propio paquete de inicio de intercambio de claves, que tiene una longitud de 1124 bytes.

5. **Client: Elliptic Curve Diffie-Hellman Key Exchange Init:** El cliente inicia el intercambio de claves utilizando el algoritmo de intercambio de claves Diffie-Hellman de curva elíptica (ECDH). Este paquete tiene una longitud de 116 bytes.
6. **Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys:** El servidor responde al cliente, completando el intercambio de claves ECDH y estableciendo nuevas claves para la sesión. Este paquete tiene una longitud de 576 bytes.

2.5. Tráfico generado por C3 (detallado)

1. **Client: Protocol (SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3):** Este es el inicio de la sesión SSH, donde el cliente C3 se está comunicando con el servidor utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.
2. **Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1):** El servidor responde al cliente confirmando que también está utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.
3. **Client: Key ExchangeInit:** El cliente inicia el intercambio de claves para establecer una sesión segura. Este paquete es significativamente más largo, con una longitud de 1428 bytes, lo que probablemente se deba a la información de la clave que se está enviando.
4. **Server: Key Exchange Init:** El servidor responde con su propio paquete de inicio de intercambio de claves, que tiene una longitud de 1124 bytes.
5. **Client: Elliptic Curve Diffie-Hellman Key Exchange Init:** El cliente inicia el intercambio de claves utilizando el algoritmo de intercambio de claves Diffie-Hellman de curva elíptica (ECDH). Este paquete tiene una longitud de 116 bytes.
6. **Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys:** El servidor responde al cliente, completando el intercambio de claves ECDH y estableciendo nuevas claves para la sesión. Este paquete tiene una longitud de 576 bytes.

2.6. Tráfico generado por C4 (4 (iface lo) (detallado)

1. **Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1):** Este es el inicio de la sesión SSH, donde el cliente C4 se está comunicando con el servidor utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.
2. **Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1):** El servidor responde al cliente confirmando que también está utilizando el protocolo SSH versión 2.0. La longitud de este paquete es de 109 bytes.

3. **Client: Key Exchange Init:** El cliente inicia el intercambio de claves para establecer una sesión segura. Este paquete es significativamente más largo, con una longitud de 1580 bytes, lo que probablemente se deba a la información de la clave que se está enviando.
4. **Server: Key Exchange Init:** El servidor responde con su propio paquete de inicio de intercambio de claves, que tiene una longitud de 1124 bytes.
5. **Client: Elliptic Curve Diffie-Hellman Key Exchange Init:** El cliente inicia el intercambio de claves utilizando el algoritmo de intercambio de claves Diffie-Hellman de curva elíptica (ECDH). Este paquete tiene una longitud de 116 bytes.
6. **Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys:** El servidor responde al cliente, completando el intercambio de claves ECDH y estableciendo nuevas claves para la sesión. Este paquete tiene una longitud de 576 bytes.

2.7. Diferencia entre C1 y C2

Al comparar el tráfico SSH generado por los clientes C1 y C2, se pueden observar varias diferencias:

1. **Versión de SSH:** C1 utiliza la versión 6.6.1p1 de OpenSSH, mientras que C2 utiliza la versión 7.3p1. Esto puede afectar a las características y la seguridad de la sesión SSH.
2. **Longitud del paquete de inicio de intercambio de claves:** El paquete de inicio de intercambio de claves del cliente en C1 tiene una longitud de 2036 bytes, mientras que en C2 tiene una longitud de 1500 bytes. Esto puede indicar que se están utilizando diferentes algoritmos o configuraciones de clave.
3. **Respuesta del servidor al intercambio de claves:** En ambos casos, el servidor responde con un paquete de inicio de intercambio de claves de 1124 bytes. Esto sugiere que el servidor puede estar utilizando la misma configuración para ambos clientes.
4. **Iniciación del intercambio de claves ECDH:** En ambos casos, el cliente inicia el intercambio de claves ECDH con un paquete de 116 bytes. Esto indica que ambos clientes están utilizando el mismo algoritmo de intercambio de claves.
5. **Respuesta del servidor al intercambio de claves ECDH:** El servidor responde con un paquete de 348 bytes en C1 y un paquete de 576 bytes en C2. Esto puede indicar que se están utilizando diferentes configuraciones de clave o que el servidor está respondiendo de manera diferente a cada cliente.

2.8. Diferencia entre C2 y C3

Al comparar el tráfico SSH generado por C2 y C3, se pueden observar algunas diferencias clave:

- **Versión del protocolo SSH:** C2 utiliza la versión 7.3p1 de OpenSSH, mientras que C3 utiliza la versión 7.7p1. Esta diferencia en las versiones puede llevar a variaciones en las características y funcionalidades soportadas por cada cliente.
- **Longitud del paquete de inicio de intercambio de claves:** En C2, el paquete de inicio de intercambio de claves tiene una longitud de 1500 bytes, mientras que en C3, este paquete tiene una longitud de 1428 bytes. Esta diferencia puede ser el resultado de los diferentes algoritmos de cifrado utilizados por cada cliente.
- **Algoritmo de cifrado:** Durante el inicio del intercambio de claves, C2 utiliza el cifrado aes128-ctr y el algoritmo hmac-sha1-etm, mientras que C3 utiliza el cifrado chacha20-poly1305. Estos diferentes algoritmos de cifrado pueden tener un impacto en la seguridad y el rendimiento de la conexión SSH.
- **Longitud de la clave pública efímera del cliente ECDH:** En ambos casos, la clave pública efímera del cliente ECDH consta de 32 bytes. Sin embargo, el paquete en C2 es de 116 bytes (44 bytes más 6 de padding) y en C3 es de 116 bytes (44 bytes más 6 de padding).
- **Uso de nuevas llaves:** En ambos casos, se menciona que se están utilizando nuevas llaves, donde el protocolo tiene un tamaño de 12 bytes y un padding de 10 bytes.
- **Interchange de mensajes cifrados:** En C2, los paquetes tienen un tamaño de 44, 60, 84, 112 y 376 bytes respectivamente. En C3, los paquetes tienen un tamaño de 44, 60, 84, 112 y 376 bytes respectivamente. Aunque los tamaños son iguales, los contenidos pueden variar debido a la diferencia en los algoritmos de cifrado utilizados.

2.9. Diferencia entre C3 y C4

Al comparar el tráfico SSHv2 generado por C3 y C4, se pueden observar varias diferencias clave:

1. **Dirección IP:** En C3, la comunicación se realiza entre las direcciones IP 172.18.0.5 (cliente) y 172.18.0.2 (servidor). En cambio, en C4, la comunicación se realiza entre la misma dirección IP 127.0.0.1 (tanto para el cliente como para el servidor), lo que indica que la comunicación se realiza en la misma máquina.
2. **Tiempo de inicio:** El tiempo de inicio de la comunicación en C3 es 6.373766 segundos, mientras que en C4 es significativamente más rápido, comenzando a los 0.000488 segundos.

3. **Versión del protocolo SSH:** Ambos, C3 y C4, utilizan la versión del protocolo SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1 para el servidor. Sin embargo, la versión del cliente difiere entre C3 y C4. C3 utiliza la versión SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3, mientras que C4 utiliza la misma versión que el servidor.
4. **Longitud de los paquetes:** La longitud de los paquetes durante la fase de "Key Exchange Init" también difiere entre C3 y C4. En C3, la longitud del paquete es de 1428 bytes, mientras que en C4 es de 1580 bytes.

Estas diferencias pueden ser atribuidas a las diferentes configuraciones de los sistemas operativos y las versiones de SSH utilizadas.

3. Desarrollo (Parte 2)

3.1. Identificación del cliente ssh

El tráfico SSH proporcionado parece corresponder a una modificación del cliente C4. Esto se basa en la observación de los tamaños de los paquetes durante el inicio del intercambio de claves y los paddings. En particular, el tamaño del paquete durante el inicio del intercambio de claves es de 1578 bytes, que es característico de C4. Además, el padding es de 10, que también es característico de C4.

3.2. Replicación de tráfico (paso por paso)

El objetivo principal era replicar el tráfico de red que se observó en la información de un paquete específico. En este paquete, el cliente utilizaba el protocolo SSH con una versión específica (SSH-2.0-OpenSSH_?). *Sin embargo, replicar este tráfico exacto no fue posible.*

Para intentar replicar el tráfico, se decidió modificar el código fuente del cliente OpenSSH y luego instalarlo. La modificación se realizó en el archivo `version.h`, que es donde se define la versión del protocolo SSH tal como aparece en el paquete.

El archivo Dockerfile muestra el proceso de instalación y configuración del cliente OpenSSH modificado. En este archivo, se descarga el código fuente de OpenSSH, se modifica el archivo `version.h` para cambiar la versión del protocolo SSH, y luego se compila e instala el cliente OpenSSH.

A continuación, se muestra el Dockerfile utilizado para este proceso:



```
1 # Dockerfile
2 FROM ubuntu:20.10
3
4 RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
5 RUN apt-get update && apt-get install -y build-essential zlib1g-dev libssl-dev libpam0g-dev unzip autoconf automake
6
7 RUN apt-get install -y wget git sshpass
8 RUN wget https://github.com/openssh/openssh-portable/archive/V_8_3_P1.zip
9 RUN unzip V_8_3_P1.zip && cd openssh-portable-V_8_3_P1 && \
10     sed -i 's/#define SSH_VERSION.*/#define SSH_VERSION "SSH-2.0-OpenSSH_?"/' version.h && \
11     autoreconf -fi && ./configure && make && make install
12
13 RUN useradd -m test && echo 'test:test' | chpasswd
14 RUN mkdir /var/run/sshd
15 EXPOSE 22
16 CMD ["/usr/local/sbin/sshd", "-D"]
17
```

Figura 7: Dockerfile utilizado para la replicación del tráfico.

A pesar de estos esfuerzos, no fue posible replicar exactamente el tráfico observado en el paquete original. Esto se debe a la complejidad inherente de la replicación exacta del tráfico de red, especialmente cuando se trata de protocolos seguros como SSH.

Conclusiones y comentarios

Este informe ha presentado un análisis detallado del tráfico SSH generado por diferentes clientes (C1, C2, C3, C4) y las diferencias clave entre ellos. Se observaron variaciones en la versión del protocolo SSH, la longitud del paquete de inicio de intercambio de claves, el algoritmo de cifrado utilizado y la longitud de la clave pública efímera del cliente ECDH.

En particular, se encontró que las diferencias en las versiones del protocolo SSH pueden llevar a variaciones en las características y funcionalidades soportadas por cada cliente. Además, las diferencias en los algoritmos de cifrado pueden tener un impacto en la seguridad y el rendimiento de la conexión SSH.

Además, se intentó replicar el tráfico de red observado en la información de un paquete específico modificando el código fuente del cliente OpenSSH y luego instalándolo. Aunque no fue posible replicar exactamente el tráfico, este enfoque proporcionó una valiosa experiencia práctica en la manipulación y el análisis del tráfico SSH.

En resumen, este informe ha proporcionado una visión profunda de cómo las diferencias en las configuraciones de los clientes SSH pueden afectar al tráfico generado, lo que tiene implicaciones importantes para la seguridad y el rendimiento de las conexiones SSH.

Anexo

Códigos SSH usados para automatizar el proceso, estos ejecutan docker, las capturas y copian los archivos .pcap al host.

```

1 #!/bin/bash
2 set -x # Imprime cada comando antes de ejecutarlo
3
4 # Crea el directorio "capturas" si no existe
5 mkdir -p capturas
6
7 # Detiene todos los contenedores en ejecución
8 docker stop $(docker ps -aq)
9
10 # Elimina todos los contenedores
11 docker rm $(docker ps -aq)
12
13 # Elimina todas las imágenes
14 docker rmi $(docker images -q)
15
16 # Elimina todas las redes
17 docker network rm $(docker network ls -q)
18
19 # Navega al directorio s1 y construye la imagen
20 cd s1
21 docker build -t my_image_s1 .
22
23 cd ..
24
25 # Crea una red
26 docker network create mynetwork
27
28 # Ejecuta el contenedor s1 en segundo plano
29 docker run --network=mynetwork -d --name s1 my_image_s1
30
31 # Obtiene la dirección IP del contenedor s1
32 s1_ip=$(docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' s1)
33
34 # Array de clientes
35 clients=("c1" "c2" "c3")
36
37 # Itera sobre cada cliente en el array
38 for client in ${clients[@]}; do
39     # Navega al directorio del cliente y construye la imagen
40     cd $client
41     docker build -t my_custom_${client} .
42     cd ..
43
44     # Ejecuta el contenedor del cliente
45     docker run --network=mynetwork -d -it --name $client my_custom_${client}
46
47     # Espera 10 segundos
48     sleep 10
49
50     # Inicia tcpdump en el contenedor del cliente
51     docker exec -d $client /bin/bash -c 'tcpdump -U -i any -w ./capture.pcap >/dev/null 2>&1 &'
52
53     # Espera 10 segundos
54     sleep 10
55
56     # Realiza una conexión SSH al contenedor del cliente
57     docker exec -it $client /bin/bash -c "sshpass -p 'test' ssh test@$s1_ip"
58
59     # Espera unos segundos para asegurar que tcpdump termina de capturar
60     sleep 10
61
62     # Copia el archivo capture.pcap del contenedor del cliente al directorio capturas en el host
63     timeout 10 docker cp ${client}:/capture.pcap ./capturas/captura_${client}.pcap
64     done
65
66     # Espera 10 segundos
67     sleep 10
68
69     # Inicia tcpdump en el contenedor s1
70     docker exec -d s1 /bin/bash -c 'tcpdump -U -i any -w ./capture.pcap >/dev/null 2>&1 &'
71
72     # Espera 10 segundos
73     sleep 10
74
75     # Realiza una conexión SSH al propio contenedor s1
76     docker exec -it s1 /bin/bash -c "sshpass -p 'test' ssh test@localhost"
77
78     # Espera unos segundos para asegurar que tcpdump termina de capturar
79     sleep 10
80
81     # Copia el archivo capture.pcap del contenedor s1 al directorio capturas en el host
82     timeout 10 docker cp s1:/capture.pcap ./capturas/captura_s1.pcap
83

```

Figura 8: Script sh principal para ejecutar todos los clientes y servidor.

```
1 #!/bin/bash
2 set -x # Imprime cada comando antes de ejecutarlo
3
4 # Crea el directorio "capturas" si no existe
5 mkdir -p capturas
6
7 # Detiene todos los contenedores en ejecución
8 docker stop $(docker ps -aq)
9
10 # Elimina todos los contenedores
11 docker rm $(docker ps -aq)
12
13 # Elimina todas las imágenes
14 docker rmi $(docker images -q)
15
16 # Elimina todas las redes
17 docker network rm $(docker network ls -q)
18
19 # Construye la imagen Docker
20 docker build -t my_image_s1 .
21
22 # Crea una red
23 docker network create mynetwork
24
25 # Ejecuta el contenedor s1 en segundo plano
26 docker run --network=mynetwork -d --name s1 my_image_s1
27
28 # Obtiene la dirección IP del contenedor s1
29 s1_ip=$(docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' s1)
30
31 # Inicia tcpdump en el contenedor s1
32 docker exec -d s1 /bin/bash -c 'tcpdump -U -i any -w /tmp/capture.pcap >/dev/null 2>&1 &'
33
34 # Espera 10 segundos
35 sleep 10
36
37 # Realiza una conexión SSH al propio contenedor s1
38 docker exec -it s1 /bin/bash -c "sshpass -p 'test' ssh -o StrictHostKeyChecking=no test@localhost"
39
40 # Espera unos segundos para asegurar que tcpdump termina de capturar
41 sleep 10
42
43 # Copia el archivo capture.pcap del contenedor s1 al directorio capturas en el host
44 docker cp s1:/tmp/capture.pcap ./capturas/captura_s1.pcap
45
```

Figura 9: Script sh para replicar tráfico, ejecuta cliente y servidor.