

Informe Laboratorio 5

Sección 1

Felipe Gutiérrez Lazo
e-mail: felipe.gutierrez_l@mail.udp.cl

Noviembre 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	6
2.1.3. C2	7
2.1.4. C3	7
2.1.5. C4	8
2.2. Creación de las credenciales para S1	8
2.3. Tráfico generado por C1 (detallado)	9
2.4. Tráfico generado por C2 (detallado)	11
2.5. Tráfico generado por C3 (detallado)	12
2.6. Tráfico generado por C4 (4 (iface lo) (detallado)	13
2.7. Diferencia entre C1 y C2	14
2.8. Diferencia entre C2 y C3	16
2.9. Diferencia entre C3 y C4	16
3. Desarrollo (Parte 2)	16
3.1. Identificación del cliente ssh	16
3.2. Replicación de tráfico (paso por paso)	17
4. Desarrollo (Parte 3)	19
4.1. Replicación de tráfico (paso por paso)	19

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

Repite 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.

TCP	66 42350 → 22 [ACK] Seq=2 Ack=
TCP	74 42398 → 22 [SYN] Seq=0 Win=
TCP	74 22 → 42398 [SYN, ACK] Seq=0
TCP	66 42398 → 22 [ACK] Seq=1 Ack=
SSHv2	87 Client: Protocol (SSH-2.0-0)
TCP	66 22 → 42398 [ACK] Seq=1 Ack=
SSHv2	107 Server: Protocol (SSH-2.0-0)
TCP	66 42398 → 22 [ACK] Seq=22 Ack=
SSHv2	1570 Client: Key Exchange Init
TCP	66 22 → 42398 [ACK] Seq=42 Ack=
SSHv2	298 Server: Key Exchange Init
TCP	66 42398 → 22 [ACK] Seq=1526 Ack=

Figura 2: Captura del Key Exchange

2. Desarrollo (Parte 1)

2.1. Códigos de cada Dockerfile

2.1.1. S1

Para la creación de este Dockerfile, se utilizan instrucciones específicas para mantener este contenedor, que alberga tanto a S1 como a C4, con la doble funcionalidad de cliente/servidor.

El código que se presenta a continuación pertenece también a la configuración de *C4*.

```

1 # Utiliza la imagen base de Ubuntu 20.10
2 FROM ubuntu:20.10
3
4 RUN sed -i 's/archive/old-releases/g' /etc/apt/sources.list
5 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/
   sources.list
6
7 # Actualiza los paquetes e instala sudo, net-tools,
8 # openssh-client y tshark
9 RUN apt-get update && apt-get install -y sudo net-tools
   openssh-server \
10 openssh-client tshark
11
12 # Configura el usuario "test" con la contraseña a "test"
13 RUN useradd -m test && echo "test:test" | chpasswd &&
   adduser test sudo
14
15 # Exponer el puerto 22 para SSH
16 EXPOSE 22
17
18 # Inicia el servicio SSH
19 CMD ["/usr/sbin/sshd", "-D"]

```

Listing 1: Dockerfile para creación de contenedor cliente/servidor.

Con estas instrucciones, se logra crear el *Dockerfile* que permite generar el contenedor que actúa como cliente y como servidor.

El Dockerfile realiza la sustitución del sistema de archivos del contenedor, además de las configuraciones de seguridad, ya que agrega el símbolo `#` al inicio de cada línea que coincide con la expresión regular que se declara.

Luego, se realiza una actualización de los paquetes del sistema operativo, en conjunto con la instalación de **sudo**, **net-tools**, **servidor** y **cliente** de **OpenSSH**, además de **tshark**, que es una herramienta de línea de comandos perteneciente a Wireshark. Más adelante será útil.

Posteriormente se agrega un nuevo usuario, llamado *test*, con su respectiva contraseña, que también equivale a *test*.

Luego, se expone el puerto **22**. Este puerto se expone para realizar las conexiones **SSH** correspondientes al momento de querer conectarse con los otros contenedores.

Para poder ejecutarlo de manera correcta, es necesario crear la imagen de este Dockerfile. Para eso, se utiliza el comando **sudo docker build -t c4s1 -f c4s1.dockerfile ..**. Este comando mediante el flag **-t** permite indicar un nombre personalizado a la imagen que esta creando, mientras que **-f** especifica el archivo que se utiliza para crear la imagen. El final

del comando, . , le indica que el archivo está en el directorio donde se está ejecutando la instrucción.

Para corroborar que funciona, se adjunta una imagen demostrativa de la ejecución en la consola del sistema.

```
felipe: ~/.../labs/lab5
.. sudo docker run -p 2222:22 --cap-add=NET_RAW --cap-add=NET_ADMIN --name client4server1 -it c
4$1 bash
[sudo] contraseña para felipe:
root@79caa36c3f6f:/# getent groups
```

Figura 3: Ejecución contenedor C4S1.

2.1.2. C1

Para la creación del contenedor que actua como cliente 1 (*c1*), se utilizan las siguientes instrucciones dentro del Dockerfile.

```
1 FROM ubuntu:14.10
2 RUN sed -i 's/http://\archive.ubuntu.com/ubuntu/http://\old-releases.ubuntu.com/ubuntu/g' /etc/apt/sources.list
3 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/
sources.list
4 RUN apt-get update && apt-get install -y sudo net-tools
openssh-client
```

Listing 2: Dockerfile para la creación de contenedor c1.

En este caso, en la línea 1 se indica la versión de Ubuntu sobre la que se desea trabajar. En este caso, es requisito que el contenedor *c1* trabaje con la versión **14.10** de Ubuntu.

Posteriormente, en la línea 2, se crea una expresión regular que permite sustituir en línea en el archivo */etc/apt/sources.list* del sistema de archivos del contenedor. La instrucción específicamente indica que se deben cambiar todas las ocurrencias de la forma *archive* por *old-releases*. Esto con el fin de poder acceder a los repositorios antiguos, debido a que la versión a utilizar es bastante antigua.

En la línea 3, también se realiza una instrucción en el sistema de archivos del contenedor. Esta vez se procede a comentar cada línea que coincide con los parámetros de la expresión regular. Esto es para no tener problemas o inconvenientes a la hora de trabajar con los protocolos de seguridad presentes en la versión utilizada.

En la última línea, se realiza la actualización de los paquetes, además de la instalación de **sudo**, **net-tools** y el **cliente** de OpenSSH.

Al igual que para la creación de la imagen del contenedor anterior, se debe utilizar el comando **sudo docker build -t c1 -f c1.dockerfile .**

A continuación, se adjunta la imagen.

```

felipe: ~/.../labs/lab5
- sudo docker run --name client -it c1 bash
root@e8444b8a7ee4:/# ssh test@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.

```

Figura 4: Ejecución contenedor C1.

2.1.3. C2

Para la creación de este contenedor, se siguen exhaustivamente los mismos pasos explicados para la subsección anterior. El código del Dockerfile utilizado se adjunta a continuación.

```

1 FROM ubuntu:16.10
2 RUN sed -i 's/http://\archive.ubuntu.com\/ubuntu/http://\\
old-releases.ubuntu.com\ubuntu/g' /etc/apt/sources.list
3 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/
sources.list
4 RUN apt-get update && apt-get install -y sudo net-tools
openSSH-client

```

Listing 3: Dockerfile para la creación de contenedor c2.

En este caso, la única diferencia en el código de creación es la versión sobre la que se construye el contenedor, que en este caso corresponde a la versión 16.10. El resto de comandos e instrucciones son los mismos, cumpliendo exactamente las funciones descritas con anterioridad.

La prueba de que el contenedor se ejecuta correctamente se adjunta a continuación.

```

felipe: ~/.../labs/lab5
- sudo docker run --name client2 -it c2 bash
root@7fc7b17c7ac:/# ssh test@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.

```

Figura 5: Ejecución contenedor C2.

2.1.4. C3

Para el contenedor *c3*, el caso es el mismo. La única diferencia es la versión sobre la que se monta el contenedor. El código se adjunta a continuación.

```

1 FROM ubuntu:18.10
2 RUN sed -i 's/http:\/\/archive.ubuntu.com\/ubuntu/http:\/\/
   old-releases.ubuntu.com\/ubuntu/g' /etc/apt/sources.list
3 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/
   sources.list
4 RUN apt-get update && apt-get install -y sudo net-tools
   openssh-client

```

Listing 4: Dockerfile para la creación de contenedor c3.

Al igual que para el contenedor 1 y 2, se sustituye con la expresión regular toda la descarga de datos de repositorios actuales por las versiones antiguas, además de desactivar todas las actualizaciones de seguridad que pueden causar problemas en la ejecución de la actividad.

Se logra ejecutar de manera correcta. A continuación, se despliega la imagen ilustrativa.

```

felipe: ~/.../labs/lab5
$ sudo docker run --name client3 -it c3 bash
[sudo] contraseña para felipe:
root@377394e4f500:/# ssh test@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.

```

Figura 6: Ejecución contenedor C3.

2.1.5. C4

Para poder crear este contenedor, se configura un Dockerfile especial. Este Dockerfile se presentó en la subsección 2.1.1, y permite crear un servidor y un cliente dentro de si mismo. Dentro de la configuración de este archivo, se instalan las dependencias para poder funcionar como servidor (*openssh-server*), y las dependencias para actuar como cliente (*openssh-client*).

Además, se debe exponer el puerto **22**, que es el que permite realizar las conexiones mediante **SSH** con los demás contenedores.

La imagen ilustrativa ya se encuentra adjunta en el documento, específicamente en la Figura 15.

2.2. Creación de las credenciales para S1

Para crear las credenciales requeridas en S1, se define en el Dockerfile la siguiente instrucción:

- RUN useradd -m test && echo "test:test" | chpasswd && adduser test sudo

La primera parte de la instrucción *useradd -m test* genera la creación del usuario ***test***, con el directorio de inicio. La instrucción *echo "test:test" | chpasswd* establece la contraseña del usuario como *test*. Finalizando la instrucción, se agrega al usuario *test* al grupo *sudo*, para otorgarle permisos de superusuario.

2.3. Tráfico generado por C1 (detallado)

El primer paso es montar los contenedores. En los pasos anteriores, se crearon las imágenes con el comando `sudo docker build -t c1 -f c1.dockerfile` . (para contenedor 1) y `sudo docker build -t c4s1 -f c4s1.dockerfile` . (para el servidor).

Ahora, con los contenedores activos, para poder generar el tráfico a través de la comunicación del contenedor 1 y el servidor 1, es necesario saber a qué dirección IP se debe realizar la conexión SSH. Para esto, en la bash del servidor, se aplica el comando `ifconfig` que permite observar la información relevante sobre las interfaces de red del sistema. A partir de esta información, se extrae la dirección IP sobre la cual se realiza la conexión SSH. A continuación, se adjunta una imagen demostrativa.

```
root@79caa36c3f6f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.17.0.5 netmask 255.255.0.0 broadcast 172.17.255.255
                ether 02:42:ac:11:00:05 txqueuelen 0 (Ethernet)
                RX packets 36 bytes 4010 (4.0 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 0 bytes 0 (0.0 B)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 7: Información sobre interfaces de red en servidor 1.

Como se observa en la figura anterior, la dirección IP correspondiente al servidor es **172.17.0.5**. Ya con esto, se puede realizar la conexión SSH desde los clientes hacia el servidor.

El siguiente paso es realizar la conexión mediante ssh desde **C1** a **S1**. Primero se corrobora que SSH esté correctamente instalado dentro del contenedor C1 para ejercer la conexión como cliente. La siguiente imagen refleja los resultados.

```
felipe: ~/.../labs/lab5
- sudo docker run --name client -it c1 bash
root@0704dd9ce4d4:/# ssh
usage: ssh [-1246AaCfgKkMNnqsTtVvXXYY] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-L [bind_address:]port:host:hostport] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-p port]
           [-Q cipher | cipher-auth | mac | kex | key]
           [-R [bind_address:]port:host:hostport] [-S ctl_path] [-W host:port]
           [-w local_tun[:remote_tun]] [user@]hostname [command]
root@0704dd9ce4d4:/#
```

Figura 8: Inicio de contenedor C1 y conexión SSH a S1.

Ahora, procede realizar la conexión hacia el servidor. El comando se ve reflejado en la imagen, e indica el tipo de conexión (**ssh**), el usuario al que se desea conectar (**test**) y la dirección IP del host al que se desea conectar (**172.17.0.5**). Es importante señalar que antes de ejecutar la instrucción de conexión, es necesario estar escuchando en la interfaz **docker0**

2.3 Tráfico generado por C1 (detallado)

2 DESARROLLO (PARTE 1)

de Wireshark para poder capturar el tráfico de la conexión. Una vez se está escuchando esa interfaz, se ejecuta el comando para realizar la conexión. La imagen que demuestra lo anterior se adjunta a continuación.

```

root@e8444b8a7ee4:/# ssh test@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.
ECDSA key fingerprint is 89:8d:1a:f2:04:90:31:d3:62:9a:3c:b8:fc:14:6e:22.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.17.0.5' (ECDSA) to the list of known hosts.
test@172.17.0.5's password:
Welcome to Ubuntu 20.10 (GNU/Linux 6.2.0-37-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Wed Nov 22 00:49:11 2023 from 172.17.0.2
$ 

```

Figura 9: Conexión de C1 a S1 mediante SSH.

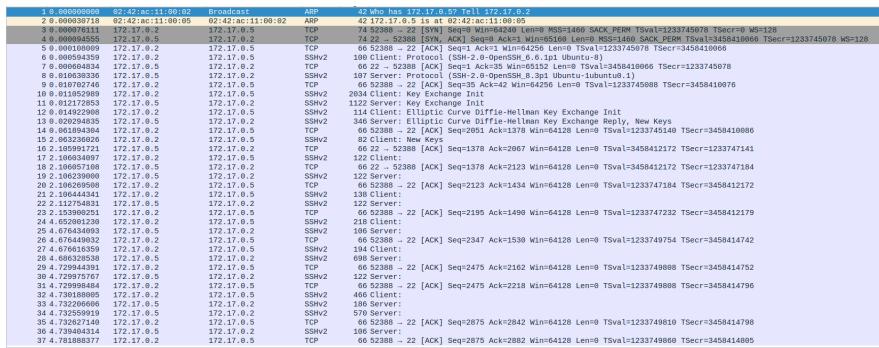


Figura 10: Tráfico generado por comunicación mediante SSH de contenedor 1 y servidor 1.

Ya capturado el tráfico, se observa que el primer paso en la conexión es realizar una consulta ARP para encontrar el destinatario al que se desea conectar. Una vez ya encontrado, se le indica al cliente quién posee la dirección a la cual se desea conectar.

Posteriormente, se realiza el triple handshake correspondiente al protocolo TCP. SSH opera sobre este protocolo, por lo que es fundamental poder realizar esta comunicación para establecer la conexión.

Una vez ya establecida la conexión TCP mediante el triple handshake, tanto el cliente como el servidor se envían información relevante para establecer la conexión SSH. Esta información incluye la versión de SSH, la versión de ubuntu, el tiempo en el que se envía el mensaje, cuánto demora en ser respondido, entre otros. Luego de generar este contacto y establecer los acuerdos para la conexión, se realiza la negociación de claves de intercambio

de **Diffie-Hellman**.

El cliente se encarga de iniciar la negociación indicando que desea realizar el intercambio de claves utilizando el algoritmo **Diffie-Hellman** con curvas elípticas. Es aquí donde el cliente entrega información sobre las curvas elípticas que soporta, en conjunto con otros detalles relacionados con el intercambio de claves.

Como respuesta a esto, el servidor le indica al cliente que también está dispuesto a intercambiar las claves utilizando las curvas elípticas. Aquí, el servidor selecciona una curva elíptica que sea soportado por los dos extremos y procede a generar las claves necesarias.

Finalmente, el cliente le hace saber al servidor que recibe las nuevas llaves correctamente. A partir de este momento, ambos utilizan las nuevas llaves para poder cifrar la información de las siguientes comunicaciones entre ellos.

2.4. Tráfico generado por C2 (detallado)

Para la comunicación entre C2 y S1, se debe repetir el procedimiento explicado en la subsección anterior. Se adjunta la imagen que demuestra el levantamiento del contenedor exitosamente.

```
felipe: ~/.../labs/lab5
- sudo docker run --name client2 -it c2 bash
root@7fc7b17c7ac:/# ssh test@172.17.0.5
The authenticity of host '172.17.0.5' can't be established.
```

Figura 11: Ejecución del contenedor C2.

En la imagen anterior se refleja la conexión mediante SSH con el S1. A continuación se observa el tráfico capturado para esta comunicación.

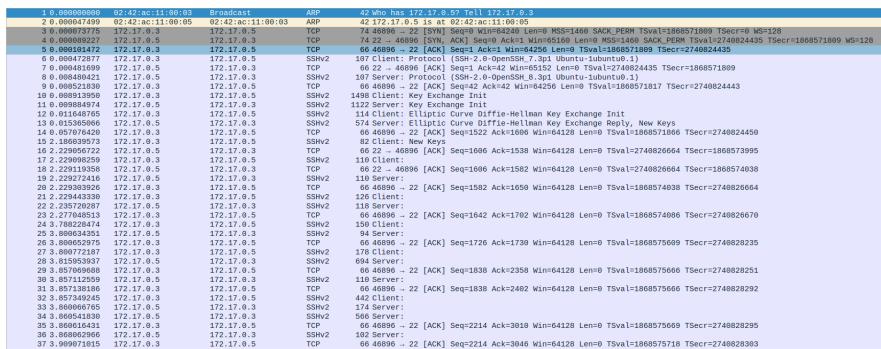


Figura 12: Tráfico generado por comunicación mediante SSH de contenedor 2 y servidor 1.

Como se observa en la figura 12, existe también un reconocimiento de la IP a la que se desea conectar el cliente. Una vez le llega el propietario de dicha dirección, se procede a

realizar el *three way handshake*, al igual que en el paso anterior, para establecer la conexión TCP entre el cliente y el servidor.

Posteriormente, se genera el mismo intercambio de llaves realizado en el paso anterior. Cabe destacar que en esta captura de datos se ve reflejado un cambio en el tamaño de ciertos paquetes. Esto es debido a las distintas configuraciones que se tienen, mayormente afectadas por las versiones del contenedor que se están utilizando (*Ubuntu 16.10*). En base a esto, también se ve reflejada el cambio de versión de SSH pertinente, ya que C2 cuenta con una versión más actualizada de este.

2.5. Tráfico generado por C3 (detallado)

Para este contenedor siguen siendo pasos muy similares. Como primer paso se debe ejecutar el contenedor que funciona como cliente, realizar la conexión SSH al servidor y capturar el tráfico. A continuación, se adjunta una imagen de la ejecución del contenedor.

```

felipe: ~/.../labs/lab5
$ sudo docker run --name client3 -it c3 bash
[sudo] contraseña para felipe:
root@377394e4f500:/# ssh test@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.

```

Figura 13: Ejecución del contenedor C3.

Luego de ejecutar la conexión mediante SSH, se capturan los datos a través de la interfaz *docker0*. El resultado se observa a continuación.

ID	Protocolo	Origen	Destino	TCP	ICMP	IP	ICMP	Traffic	Traffic
3.0. 0.009044893	02:42:ac:11:98:05	02:42:ac:11:98:04		TCP	42 172.17.0.5 172.17.0.4	74 60958 - 22 [SYN] Seq=0 Win=1460 SACK_PERM TSeq=164971384 TSer=0 WS=128			
3.0. 0.009044893	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1 Ack=1 Win=1460 TSeq=164971384 TSer=27960609058 TSecr=164971384 WS=128				
5.0. 0.009122910	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1 Ack=1 Win=1460 TSeq=164971384 TSer=27960609328 TSecr=164971384 WS=128				
6.0. 0.009595413	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1 Ack=1 Win=1460 TSeq=164971384 TSer=27960609328 TSecr=164971384 WS=128				
8.0. 0.012711984	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1 Ack=1 Win=1460 TSeq=164971384 TSer=27960609328 TSecr=164971384 WS=128				
8.0. 0.012711984	172.17.0.5	172.17.0.4		SSHv2	107 Client: Protocol [SSH-2.0-OpenSSH_7.1pl1 Ubuntu-16.04.3]				
9.0. 0.013062211	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=42 Ack=42 Win=1428 Len=0 TSeq=164971314 TSer=27960609448				
10.0. 0.013062211	172.17.0.4	172.17.0.5		SSHv2	124 Client: Key Exchange Init				
11.0. 0.011845252	172.17.0.5	172.17.0.4		SSHv2	114 Server: Elliptic Curve Diffie-Hellman Key Exchange Init				
12.0. 0.012955371	172.17.0.4	172.17.0.5		SSHv2	154 Client: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys				
13.0. 0.012955371	172.17.0.5	172.17.0.4		TCP	66 60958 - 22 [ACK] Seq=156 Ack=1608 Win=64128 Len=0 TSeq=1649713033 TSer=2796060954				
14.0. 0.058887893	172.17.0.4	172.17.0.5		SSHv2	82 Client: New Keys				
15.0. 2.008817889	172.17.0.5	172.17.0.4		TCP	66 22 60958 - 22 [ACK] Seq=1606 Ack=1456 Win=64128 Len=0 TSeq=164971385 TSecr=497285				
17.1. 3.232949727	172.17.0.4	172.17.0.5		TCP	66 22 60958 [ACK] Seq=1606 Ack=1456 Win=64128 Len=0 TSeq=164971385 TSecr=497285				
18.1. 3.232970798	172.17.0.5	172.17.0.4		SSHv2	119 Client:				
19.1. 3.232970798	172.17.0.5	172.17.0.4		SSHv2	66 22 60958 [ACK] Seq=1606 Ack=1510 Win=64128 Len=0 TSeq=164971385 TSecr=497285				
20.1. 3.232338543	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1510 Ack=1608 Win=64128 Len=0 TSeq=164972627 TSer=2796061361				
21.1. 3.232338866	172.17.0.4	172.17.0.5		SSHv2	126 Client:				
22.1. 3.232338866	172.17.0.4	172.17.0.5		SSHv2	158 Client:				
23.1. 3.232338866	172.17.0.4	172.17.0.5		SSHv2	154 Client:				
23.1. 3.232338866	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1570 Ack=1782 Win=64128 Len=0 TSeq=164972675 TSer=2796061367				
24.3. 0.270053896	172.17.0.4	172.17.0.5		SSHv2	94 Client:				
24.3. 0.270053896	172.17.0.5	172.17.0.4		TCP	66 60958 - 22 [ACK] Seq=1654 Ack=1739 Win=64128 Len=0 TSeq=164974343 TSer=2796063977				
25.3. 0.270053896	172.17.0.4	172.17.0.5		SSHv2	176 Client:				
26.3. 0.270053896	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1654 Ack=1739 Win=64128 Len=0 TSeq=164974343 TSer=2796063977				
27.3. 0.270053896	172.17.0.4	172.17.0.5		SSHv2	164 Server:				
29.3. 0.099191393	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=1766 Ack=2358 Win=64128 Len=0 TSeq=164974395 TSer=2796063086				
30.3. 0.099191393	172.17.0.4	172.17.0.5		SSHv2	119 Client:				
31.3. 0.099191393	172.17.0.4	172.17.0.5		SSHv2	66 60958 - 22 [ACK] Seq=1766 Ack=2402 Win=64128 Len=0 TSeq=164974395 TSer=2796063129				
32.3. 0.091255770	172.17.0.4	172.17.0.5		TCP	442 Client:				
33.3. 0.091255770	172.17.0.4	172.17.0.5		SSHv2	176 Client:				
34.3. 0.094385754	172.17.0.5	172.17.0.4		SSHv2	560 Server:				
35.3. 0.094467866	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=2142 Ack=3010 Win=64128 Len=0 TSeq=164974398 TSer=2796063132				
36.3. 0.103389448	172.17.0.4	172.17.0.5		SSHv2	182 Client:				
37.3. 0.145698981	172.17.0.4	172.17.0.5		TCP	66 60958 - 22 [ACK] Seq=2142 Ack=3046 Win=64128 Len=0 TSeq=164974447 TSer=2796063140				

Figura 14: Tráfico generado por comunicación mediante SSH de contenedor 3 y servidor 1.

El procedimiento es similar al tráfico generado por C1 y C2. Se establece la conexión TCP, se intercambian las llaves, y el cliente acepta las llaves nuevas que le entrega el servidor.

Al igual que en el caso anterior, las diferencias más visibles son la versión utilizada de OpenSSH, que en este caso utiliza **7.7p1** para el caso del cliente. Además, en el intercambio de llaves (*Key Exchange Init*) se observa un length variable entre los tráficos generados.

La respuesta a largos variables en el mismo paquete en distintas conexiones es por la cantidad de algoritmos que vienen configurados para cada versión de OpenSSH. Esto genera una diferencia en el tamaño del paquete, ya que algunas versiones contienen más, o incluso menos algoritmos de cifrado de información.

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

Para este caso es diferente el procedimiento realizado. Como este contenedor tiene que actuar como cliente, y además, como servidor, es necesario ejecutarlo de una forma distinta. El comando que permite acceder a todo lo necesario dentro de la ejecución es **sudo docker run -p 2222:22 --cap-add=NET_RAW --cap-add=NET_ADMIN -it nombre_imagen bash**. Este comando permite que el contenedor pueda tener privilegios específicos con operaciones de red de bajo nivel, como mensajes ICMP, además de la administración de la red. A continuación, se adjunta una foto demostrativa.

```
felipe: ~/.../labs/lab5
$ sudo docker run -p 2222:22 --cap-add=NET_RAW --cap-add=NET_ADMIN --name client4server1 -it c
4s1 bash
[sudo] contraseña para felipe:
root@79caa36c3f6f:/# getent groups
```

Figura 15: Ejecución de contenedor C4S1.

Posteriormente, se instala Wireshark. El comando que permite esta instalación es *sudo apt install wireshark*. Una vez que se encuentra instalado, se ejecuta el comando **dpkg-reconfigure wireshark-common**, que permite elegir los tipos de usuarios que pueden capturar los paquetes. En este caso, se desea que los *no super usuarios* puedan acceder a capturar el tráfico de la red.

Luego, se debe agregar el usuario creado al grupo **wireshark**. El comando para dicha instrucción es **sudo usermod -aG wireshark test**. Esto es para que el usuario en cuestión (*test*) tenga acceso a las funcionalidades del software *Wireshark*.

Posteriormente, se realiza la creación de una carpeta, desde el usuario *root* para almacenar la captura a realizar. A dicha carpeta se le otorgan los permisos máximos (777) para manipularla sin problemas.

El siguiente paso consiste en realizar el cambio de usuario con el comando **su test**. Esto permite navegar desde el usuario **root** al usuario indicado en la instrucción anterior. Ya siendo el usuario *test*, se procede a realizar el inicio de la captura del tráfico. Para esto, se ejecuta la siguiente instrucción: **tshark -i lo -w captura1.pcapng**. Esto permite que se escanee la red local (loopback), donde se está realizando la comunicación entre C4 y S1.

En otra terminal, se debe ejecutar nuevamente el mismo contenedor. Esta vez se ejecuta con el comando **sudo docker exec -it id_contenedor bash**. Esto permite acceder al contenedor aunque ya esté funcionando. Dentro de esta terminal, se genera la solicitud de conexión

mediante SSH. El formato para comenzar esta conexión es la siguiente: **ssh test@localhost**. Esta vez el comando indica que la conexión que se quiere realizar es al usuario *test*, y la dirección de la red a la que se desea conectar es la red local.

Al ser primera vez que se realiza esto con el contenedor, se debe manifestar si se confía o no en la dirección entregada. Para proseguir con el laboratorio, se debe seguir y confiar en la dirección de destino. Posteriormente, se debe ingresar la contraseña del usuario para finalmente concretar la conexión. Una vez realizado esto, la conexión se detiene y queda esperando al envío de información entre los dispositivos.

Aquí, al igual que antes, en el tráfico cambian parámetros tales como el largo de los paquetes, las versiones utilizadas, el intercambio de las llaves, entre otros. A continuación se adjunta una imagen ilustrativa del tráfico generado en esta conexión.

Frame	Source	Destination	Protocol	Information
1	10.0.0.1	127.0.0.1	TCP	74.49928 - 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSeqr=2188738760 TSeqr=0 WS=128
2	127.0.0.1	10.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSeqr=2188738760 TSeqr=2188738760 WS=128
3	10.0.0.1	127.0.0.1	SSHv2	187 Client: Protocol (SSH-2.0-OpenSSH_8.3_3) Ubuntu-18.04.6-Ubuntu-1.1
4	127.0.0.1	10.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1998 Ack=1998 Win=65536 Len=0 TSeqr=2188738761 TSeqr=2188738761
5	10.0.0.1	127.0.0.1	SSHv2	187 Server: Protocol (SSH-2.0-OpenSSH_8.3_3) Ubuntu-18.04.6-Ubuntu-1.1
6	127.0.0.1	10.0.0.1	TCP	66.49928 - 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSeqr=2188738770 TSeqr=2188738770
7	10.0.0.1	127.0.0.1	SSHv2	151 Client: Key Exchange Init
8	127.0.0.1	10.0.0.1	SSHv2	1122 Server: Key Exchange Init
9	10.0.0.1	127.0.0.1	SSHv2	114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
10	127.0.0.1	10.0.0.1	SSHv2	241 Server: Elliptic Curve Diffie-Hellman Key Exchange Init
11	10.0.0.1	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1606 Ack=1606 Win=65536 Len=0 TSeqr=2188738775
12	127.0.0.1	10.0.0.1	SSHv2	110 Client: New Keys
13	10.0.0.1	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1606 Ack=1606 Win=65536 Len=0 TSeqr=2188738775
14	127.0.0.1	10.0.0.1	SSHv2	110 Server: New Keys
15	4.927096343	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1606 Ack=1606 Win=65536 Len=0 TSeqr=2188734747
16	10.0.0.1	127.0.0.1	SSHv2	110 Client:
17	4.927186151	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1606 Ack=1606 Win=65536 Len=0 TSeqr=2188734787
18	127.0.0.1	10.0.0.1	SSHv2	110 Server:
19	4.927204480	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1606 Ack=1606 Win=65536 Len=0 TSeqr=2188734787
20	127.0.0.1	10.0.0.1	SSHv2	118 Client:
21	4.933694527	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1722 Ack=1722 Win=65536 Len=0 TSeqr=2188734835 TSeqr=2188734793
22	127.0.0.1	10.0.0.1	SSHv2	118 Server:
23	6.746878534	127.0.0.1	TCP	94 Server:
24	127.0.0.1	10.0.0.1	SSHv2	178 Client:
25	6.747911287	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1806 Ack=1739 Win=65536 Len=0 TSeqr=2188737597 TSeqr=2188737597
26	6.762048825	127.0.0.1	SSHv2	178 Client:
27	127.0.0.1	10.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1818 Ack=2358 Win=65536 Len=0 TSeqr=2188737563 TSeqr=2188737522
28	6.892947914	127.0.0.1	SSHv2	110 Server:
29	6.892948459	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1918 Ack=2402 Win=65536 Len=0 TSeqr=2188737563 TSeqr=2188737563
30	127.0.0.1	10.0.0.1	SSHv2	174 Client:
31	6.895993128	127.0.0.1	TCP	66.49928 - 22 [ACK] Seq=1918 Ack=2402 Win=65536 Len=0 TSeqr=2188737563 TSeqr=2188737563
32	6.896351938	127.0.0.1	SSHv2	174 Server:
33	127.0.0.1	10.0.0.1	TCP	66.49928 - 22 [ACK] Seq=2294 Ack=3010 Win=65536 Len=0 TSeqr=2188737566 TSeqr=2188737566
34	6.813865087	127.0.0.1	SSHv2	56 Server:
35	127.0.0.1	10.0.0.1	TCP	66.49928 - 22 [ACK] Seq=2294 Ack=3046 Win=65536 Len=0 TSeqr=2188737615 TSeqr=2188737574

Figura 16: Tráfico generado por comunicación mediante SSH de contenedor 4 y servidor 1.

2.7. Diferencia entre C1 y C2

En este caso, se encuentran 3 relevancias importantes. A continuación, se despliega una imagen ilustrativa de las diferencias más significativas encontradas.

```
< Frame 10: 2034 bytes on wire (16272 bits), 2034 bytes captured (16272 bits) on interface docker0, id 0
...
> Frame 10: 1498 bytes on wire (11984 bits), 1498 bytes captured (11984 bits) on interface docker0, id 0
769c769
<   Arrival Time: Nov 21, 2023 21:52:23.509221938 -03
...
>   Arrival Time: Nov 21, 2023 21:58:21.771906240 -03
771,774c771,774
<   Epoch Time: 1700614343.509221938 seconds
<   [Time delta from previous captured frame: 0.000350243 seconds]
<   [Time delta from previous displayed frame: 0.000350243 seconds]
<   [Time since reference or first frame: 0.011052989 seconds]
...
>   Epoch Time: 1700614701.771906240 seconds
>   [Time delta from previous captured frame: 0.000392120 seconds]
>   [Time delta from previous displayed frame: 0.000392120 seconds]
>   [Time since reference or first frame: 0.008913950 seconds]
776,777c776,777
<   Frame Length: 2034 bytes (16272 bits)
<   Capture Length: 2034 bytes (16272 bits)
...
>   Frame Length: 1498 bytes (11984 bits)
>   Capture Length: 1498 bytes (11984 bits)
```

Figura 17: Diferencia n°1 C1-S1.

Esta figura representa las diferencias de tamaño que existen en los paquetes de intercambio de llaves entre el cliente y el servidor. Como se observa en la figura, el tamaño entre ambos es significativo, siendo más de 500 bytes la diferencia.

Figura 18: Explicación diferencia n°1 C1-S1.

Como se puede presenciar en esta figura, la cantidad de algoritmos presentes para el cifrado de la información en las diferentes versiones de OpenSSH empleadas es considerable. Esto genera que el tamaño del paquete de inicio de negociación para el intercambio de llaves sea menor. Las líneas de color rojo corresponden a la captura de tráfico generado por C1, mientras que las de color verde corresponden a la captura de tráfico generado por C2.

```
< Frame 13: 346 bytes on wire (2708 bits), 346 bytes captured (2708 bits) on interface docker0, id 0
...
> Frame 13: 574 bytes on wire (4592 bits), 574 bytes captured (4592 bits) on interface docker0, id 0
1139c1139
```

Figura 19: Diferencia n°2 C1-S1.

Otra de las diferencias que se observan en estas imágenes son la cantidad de curvas elípticas que puede elegir el servidor para realizar el intercambio de las llaves. C2 contiene una mayor cantidad de curvas elegibles, mientras que C1 queda obsoleto en cantidad de curvas que pueden ser elegidos por ambos dispositivos.

```
<      [TCP Segment Len: 280]
--->      [TCP Segment Len: 508]
```

Figura 20: Largo del segmento TCP entre C1 y S1.

En la figura 20 se observa que el largo de los segmentos TCP difiere en una gran cantidad. Esto puede deberse a:

- Mayor carga útil de datos.

- Fragmentación de los datos.
- Eficiencia de la transmisión.

```

< SSH Version 2 (encryption:aes128-ctr compression:none)
--- Para la comunicación entre C2 y S1, se debe repetir el procedimiento explicado en la sección 2.8.
> SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)

```

Figura 21: Diferencia nº3 C1-S1.

En esta última figura se observa otra diferencia entre los datos capturados. Esta vez corresponde a la forma de encriptar que tiene cada SSH. Esto puede afectar en el largo de ciertos paquetes, como el cambio de llaves entre el cliente y el servidor, entre otros.

2.8. Diferencia entre C2 y C3

En este caso, las diferencias observables con la versión utilizada de SSH compatible con cada versión de OpenSSH, además del tamaño del paquete del intercambio de llaves de parte del cliente. Para no sobrecargar con imágenes, no se adjunta en el informe. Sin embargo, en el repositorio se encuentran las capturas para comparar además de las diferencias de ambos archivos.

2.9. Diferencia entre C3 y C4

Lo primero que se puede observar en esta comparativa es que no hay presencia de protocolo ARP, ya que el cliente se encuentra dentro de la misma red que el servidor. Luego, como en los casos anteriores, se evidencia una diferencia entre las versiones de SSH, además del tamaño de los paquetes del intercambio de llaves del cliente.

3. Desarrollo (Parte 2)

3.1. Identificación del cliente ssh

Para poder identificar al cliente ssh, se toma en consideración que cada versión del software utilizado genera un tamaño específico en el paquete para el intercambio de las llaves. Como se observa en la figura 1, dicho paquete tiene un largo de **1578 bytes**. Dentro del tráfico que se genera a partir de cada contenedor, se observa que el usuario informante corresponde al contenedor C4S1. En el cliente 4, al momento de querer realizar el intercambio de las llaves, se observa que envía un paquete con la misma cantidad de bytes que se busca: **1578 bytes**. Además, se descarta que el usuario sea C1, C2 o C3, ya que cada uno de estos contiene un tamaño equivalente a 2034, 1498 y 1426 bytes.

3.2. Replicación de tráfico (paso por paso)

Para poder generar la replicación del tráfico, se deben seguir los siguientes pasos:

1. Se debe ejecutar el contenedor C4S1.
2. Actualizar los paquetes del contenedor.
3. Instalar cada una de las librerías y dependencias necesarias para la correcta ejecución.
4. Clonar el repositorio de **openssh-portable**.
5. Modificar archivo version.h.
6. Ejecutar comandos para guardar y volver a compilar con la nueva configuración.

Para el primer paso, se utiliza el mismo comando mencionado anteriormente: **sudo docker run --cap-add=NET_RAW --cap-add=NET_ADMIN -it nombre_imagen bash**. Como en subsecciones anteriores se demuestra el funcionamiento del comando, se omite. El siguiente paso es la ejecución de este comando. Una vez dentro del contenedor, se utiliza **apt update** para actualizar los paquetes del contenedor. Luego, se instalan las librerías necesarias para el desarrollo de la actividad. El comando utilizado corresponde a **apt install autoconf libssl-dev zlib1g-dev gcc make git vim**.

Ya instaladas las dependencias necesarias, se procede a clonar el repositorio para poder modificar los archivos necesarios. El comando utilizado se adjunta a continuación.

```
root@b6e542ccc018:/# git clone https://github.com/openssh/openssh-portable
Cloning into 'openssh-portable'...
remote: Enumerating objects: 65740, done.
remote: Counting objects: 100% (123/123), done.
remote: Compressing objects: 100% (59/59), done.
remote: Total 65740 (delta 75), reused 102 (delta 63), pack-reused 65617
Receiving objects: 100% (65740/65740), 26.63 MiB | 21.25 MiB/s, done.
Resolving deltas: 100% (50811/50811), done.
root@b6e542ccc018:/#
```

Figura 22: Clonación del repositorio de openssh-portable.

Ya clonado el repositorio, es necesario modificar el archivo *version.h* que se ubica dentro de la carpeta *openssh-portable*. Dentro de este archivo, se elimina el número de la versión y además, el valor de *SSH_PORTABLE*. La imagen del archivo resultante se adjunta a continuación.

```
/* $OpenBSD: version.h,v 1.99 2023/10/04 04:04:09 djm Exp $ */
#define SSH_VERSION      "OpenSSH_?""
#define SSH_PORTABLE     "p1"
#define SSH_RELEASE      SSH_VERSION
```

Figura 23: Edición de archivo version.h.

La serie de comandos que se deben ejecutar posteriormente se lista a continuación:

- autoreconf
- ./configure
- make
- make install
- /usr/local/sbin/sshd

Esta serie de instrucciones permite recompilar el archivo con los datos entregados. Finalmente, se realiza la petición de conexión mediante SSH. A continuación, se adjunta la imagen demostrativa.

```

root@a5e9a05c434c:/openssh-portable# /usr/local/sbin/sshd
root@a5e9a05c434c:/openssh-portable# cd ..
root@a5e9a05c434c:# ssh test@localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:fPn9K0i0A300JgBQQS3Lp27Dkhh9WXZPXigqg1dN9M.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
test@localhost's password:
Welcome to Ubuntu 20.10 (GNU/Linux 6.2.0-37-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ exit
Connection to localhost closed.
root@a5e9a05c434c:# 
```

Figura 24: Conexión mediante SSH replicando el tráfico.

Finalmente, desde el servidor se repiten los pasos. Se ejecuta con el comando **sudo docker exec -it ip_contenedor bash**. Una vez dentro de este, se procede a escuchar la red mediante **tshark**. La instrucción para esto es **tshark -i lo -w /tmp/captura_paso2.pcapng**. Cuando se encuentra escuchando, es cuando el otro dispositivo empieza la conexión ssh, ejecutando el comando **ssh test@localhost**.

Luego, se debe extraer la captura hacia fuera de Docker. Esto es posible con el comando **sudo docker cp aea65d402c39:/tmp/captura_paso2.pcapng ..**. Finalmente, en la captura se observa la réplica del tráfico generado. A continuación se adjunta la imagen ilustrativa.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	127.0.0.1	127.0.0.1	TCP	74	42598 -- 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM Tsv1=22195947
2	0.0000024122	127.0.0.1	127.0.0.1	TCP	74	22 .. 42598 [SYN, ACK] Seq=1 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM Tsv1=22195947
3	0.0000039342	127.0.0.1	127.0.0.1	TCP	66	42598 .. 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 Tsv1=2219594789 Tsecr=2219594789
4	0.0000040000	127.0.0.1	127.0.0.1	TCP	66	42598 .. 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 Tsv1=2219594789 Tsecr=2219594789
5	0.0000043215	127.0.0.1	127.0.0.1	TCP	66	42598 .. 22 [ACK] Seq=1 Ack=28 Win=65536 Len=0 Tsv1=2219594789 Tsecr=2219594789
6	0.014288668	127.0.0.1	127.0.0.1	SSHv2	187	Server: Protocol (SSH-2.0-OpenSSH_8.3pi Ubuntu-1ubuntu6.1)
7	0.014306668	127.0.0.1	127.0.0.1	TCP	66	42598 .. 22 [ACK] Seq=29 Ack=42 Win=65536 Len=0 Tsv1=2219594883 Tsecr=2219594883

Figura 25: Captura de datos replicados.

Como se observa en la figura, la versión de SSH aparece incógnita, no permitiendo saber que versión se está utilizando, ocultando así información del cliente.

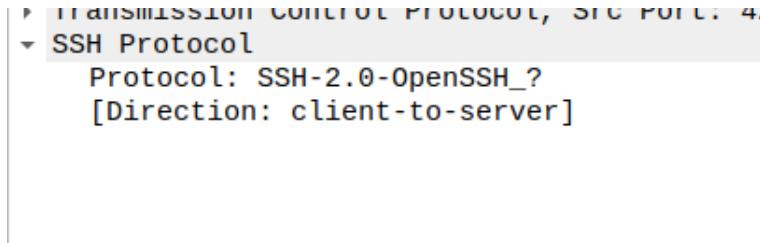


Figura 26: Versión oculta de SSH.

4. Desarrollo (Parte 3)

4.1. Replicación de tráfico (paso por paso)

Para replicar el tráfico objetivo, se realizan los siguientes pasos:

1. Ejecutar contenedor.
2. Instalar dependencias necesarias.
3. Clonar repositorio de openssh-portable.
4. Modificar archivo sshd_config.
5. Ejecutar comandos para guardar y compilar con la nueva información.
6. Establecer conexión SSH.

El contenedor vuelve a ejecutarse de la misma manera. **sudo docker run --cap-add=NET_RAW --cap-add=NET_ADMIN -it nombre_imagen bash** es el comando que permite esto. En la siguiente imagen se reflejan el punto 1 y 2 descritos anteriormente.

```

felipe: ~/.../labs/lab5
- sudo docker run --cap-add=NET_RAW --cap-add=NET_ADMIN -it c4s1 bash
root@fc19bfff863ce:/# apt update
Hit:1 http://old-releases.ubuntu.com/ubuntu groovy InRelease
Hit:2 http://old-releases.ubuntu.com/ubuntu groovy-updates InRelease
Hit:3 http://old-releases.ubuntu.com/ubuntu groovy-backports InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
root@fc19bfff863ce:/# apt install autoconf libssl-dev zlib1g-dev gcc make git vim
Reading package lists... Done
Building dependency tree

```

Figura 27: Ejecución e instalación de dependencias necesarias para funcionamiento del contenedor C4S1.

En la siguiente figura, se observa la ejecución de clonar el repositorio directamente desde GitHub.

```

root@fc19bfff863ce:/# git clone https://github.com/openssh/openssh-portable
Cloning into 'openssh-portable'...
remote: Enumerating objects: 65740, done.
remote: Counting objects: 100% (123/123), done.
remote: Compressing objects: 100% (59/59), done.
remote: Total 65740 (delta 75), reused 102 (delta 63), pack-reused 65617
Receiving objects: 100% (65740/65740), 26.63 MiB | 20.99 MiB/s, done.
Resolving deltas: 100% (50815/50815), done.
root@fc19bfff863ce:/# █

```

Figura 28: Clonación del repositorio de openssh-portable.

Ya con el repositorio clonado, se procede a modificar el archivo objetivo: **sshd_config**. Este se encuentra dentro de la carpeta *openssh-portable*.

Estando dentro del archivo, en la modificación, se agregan los siguientes algoritmos y parámetros de cifrado necesarios para el desarrollo de la actividad.

```

# Ciphers and keying
#RekeyLimit default none
#repkey
#Agregado
Ciphers aes128-ctr
HostKeyAlgorithms ecdsa-sha2-nistp256
KexAlgorithms ecdh-sha2-nistp256
MACs hmac-sha2-256

# Logging
#syslogFacility AUTH
#LogLevel INFO

```

Figura 29: Parámetros y algoritmos agregados a *sshd_config*.@

Posteriormente, se ejecutan los comandos **autoreconf**, **./configure**, **make**, **make install**, **/usr/local/sbin/sshd** para recompilar con el archivo editado.

Una vez finalizado esto, se reinicia el servicio de SSH.

4.1 Replicación de tráfico (paso por paso)

4 DESARROLLO (PARTE 3)

Finalmente, al igual que en los otros casos, se realiza la conexión SSH. Se ejecuta otra terminal con el contenedor activo mediante el comando **sudo docker exec -it id_contenedor bash**, y se comienza a escanear la interfaz loopback mediante tshark. En la terminal donde se realizan los cambios al archivo sshd_config, se genera la conexión ssh mediante *ssh test@localhost*.

Finalmente, la captura entrega el siguiente resultado.

	T	E	A	S	D	TCP	T	E	A	S	D	TCP	T	E	A	S	D																																																																																																																																																																																																																																										
1	1.1.8880000000	127.0.0.1	127.0.0.1	TCP	74	37546 - Zz [SYN] Seq=0 Win=5536 [Conn] RSS=65495 SACK_PERM Tso=2224487926 Tscurr=9405128	2	1.1.8880000000	127.0.0.1	127.0.0.1	TCP	74	37546 - Zz [SYN] Seq=0 Win=5536 [Conn] ACK=Seq=1 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487926 Ts=128	3	0.000859076	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=1 Ack=1 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487926	4	0.000859076	127.0.0.1	127.0.0.1	TCP	67	37846 + 22 [ACK] Seq=2 Ack=2 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487926	5	0.000842067	127.0.0.1	127.0.0.1	TCP	66	22 + 37846 [ACK] Seq=2 Ack=2 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487926	6	0.000819349	127.0.0.1	127.0.0.1	TCP	87	Server: Profound (SSH-2.0-OpenSSH_9.5)	7	0.000819349	127.0.0.1	127.0.0.1	TCP	66	22 + 37846 [ACK] Seq=3 Ack=3 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487934	8	0.000568317	127.0.0.1	127.0.0.1	SSH2	1578	Client: Key Exchange Init	9	0.000568317	127.0.0.1	127.0.0.1	SSH2	146	Client: Elliptic Curve Diffie-Hellman Key Exchange Init	10	0.000734515	127.0.0.1	127.0.0.1	SSH2	746	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys	11	0.016927875	127.0.0.1	127.0.0.1	TCP	66	22 + 37846 [ACK] Seq=1000 Ack=1000 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487937	12	0.016927875	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=1000 Ack=1000 Win=5536 Len=0 Tsoval=2224487926 Tscurr=2224487937	13	1.781346949	127.0.0.1	127.0.0.1	SSH2	82	Client: New Keys	14	1.824679391	127.0.0.1	127.0.0.1	TCP	66	22 + 37846 [ACK] Seq=992 Ack=1822 Win=5536 Len=0 Tsoval=2224489731 Tscurr=2224489737	15	1.824679391	127.0.0.1	127.0.0.1	TCP	138	Client:	16	1.824746748	127.0.0.1	127.0.0.1	TCP	66	22 + 37846 [ACK] Seq=992 Ack=1868 Win=5536 Len=0 Tsoval=2224489731 Tscurr=2224489731	17	1.824746748	127.0.0.1	127.0.0.1	TCP	138	Client:	18	1.825952493	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=1686 Ack=966 Win=5536 Len=0 Tsoval=2224489731 Tscurr=2224489731	19	1.825952493	127.0.0.1	127.0.0.1	SSH2	146	Client:	20	1.826472353	127.0.0.1	127.0.0.1	SSH2	150	Client:	21	1.826472353	127.0.0.1	127.0.0.1	SSH2	178	Client:	22	1.832664616	127.0.0.1	127.0.0.1	SSH2	162	Server:	23	1.832664616	127.0.0.1	127.0.0.1	TCP	66	22 + 37846 + 22 [ACK] Seq=1878 Ack=1158 Win=5536 Len=0 Tsoval=2224489880 Tscurr=2224489759	24	3.35987626	127.0.0.1	127.0.0.1	SSH2	226	Client:	25	3.35987626	127.0.0.1	127.0.0.1	SSH2	224	Client:	26	3.361506250	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=2038 Ack=1206 Win=5536 Len=0 Tsoval=2224491287 Tscurr=2224491287	27	3.361506250	127.0.0.1	127.0.0.1	SSH2	226	Client:	28	3.361506250	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=2038 Ack=1206 Win=5536 Len=0 Tsoval=2224491287 Tscurr=2224491287	29	3.361879398	127.0.0.1	127.0.0.1	SSH2	732	Client:	30	3.361879398	127.0.0.1	127.0.0.1	SSH2	610	Client:	31	3.364125566	127.0.0.1	127.0.0.1	SSH2	482	Client:	32	3.364125566	127.0.0.1	127.0.0.1	SSH2	610	Server:	33	3.369790800	127.0.0.1	127.0.0.1	TCP	220	Client:	34	3.374655121	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=3158 Ack=2630 Win=5536 Len=0 Tsoval=2224491388 Tscurr=2224491296	35	3.374655121	127.0.0.1	127.0.0.1	SSH2	138	Server:	36	3.424636309	127.0.0.1	127.0.0.1	TCP	66	37846 + 22 [ACK] Seq=3158 Ack=2694 Win=5536 Len=0 Tsoval=2224491351 Tscurr=2224491383

Figura 30: Captura de datos replicados.

Como se observa en la figura, el objetivo se cumple. El servidor obtiene un largo menor a 300 bytes en el intercambio de llaves con el cliente.

Conclusiones y comentarios

En esta experiencia se comprende la importancia crítica de la protección de la conexión SSH entre dos usuarios. Mediante todo el laboratorio se ponen a prueba varias versiones del software con distintas versiones del sistema operativo, y se comprende empíricamente el funcionamiento de estas. Además, se comprende la funcionalidad que puede entregar una herramienta como Docker al momento de querer realizar tareas expuestas a la red, como conexiones mediante el puerto 22. Se comprende además el funcionamiento de la conexión SSH profundamente, además de poner en práctica el algoritmo Diffie-Hellman al momento de traspasar la llave entre cliente y servidor.