

# Informe Laboratorio 5

## Sección 2

David Silva

e-mail: davidesteb.silva@mail.udp.cl

Noviembre de 2023

## Índice

<b>1. Descripción de actividades</b>	<b>2</b>
<b>2. Desarrollo (Parte 1)</b>	<b>4</b>
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 . . . . .	7
2.3. Tráfico generado por C1 (detallado) . . . . .	7
2.4. Tráfico generado por C2 (detallado) . . . . .	8
2.5. Tráfico generado por C3 (detallado) . . . . .	9
2.6. Tráfico generado por C4 (4 iface lo) (detallado) . . . . .	9
2.7. Diferencia entre C1 y C2 . . . . .	10
2.8. Diferencia entre C2 y C3 . . . . .	11
2.9. Diferencia entre C3 y C4 . . . . .	11
<b>3. Desarrollo (Parte 2)</b>	<b>12</b>
3.1. Identificación del cliente ssh . . . . .	12
3.2. Replicación de tráfico (paso por paso) . . . . .	12
<b>4. Desarrollo (Parte 3)</b>	<b>13</b>
4.1. Replicación de tráfico (paso por paso) . . . . .	13

## 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.

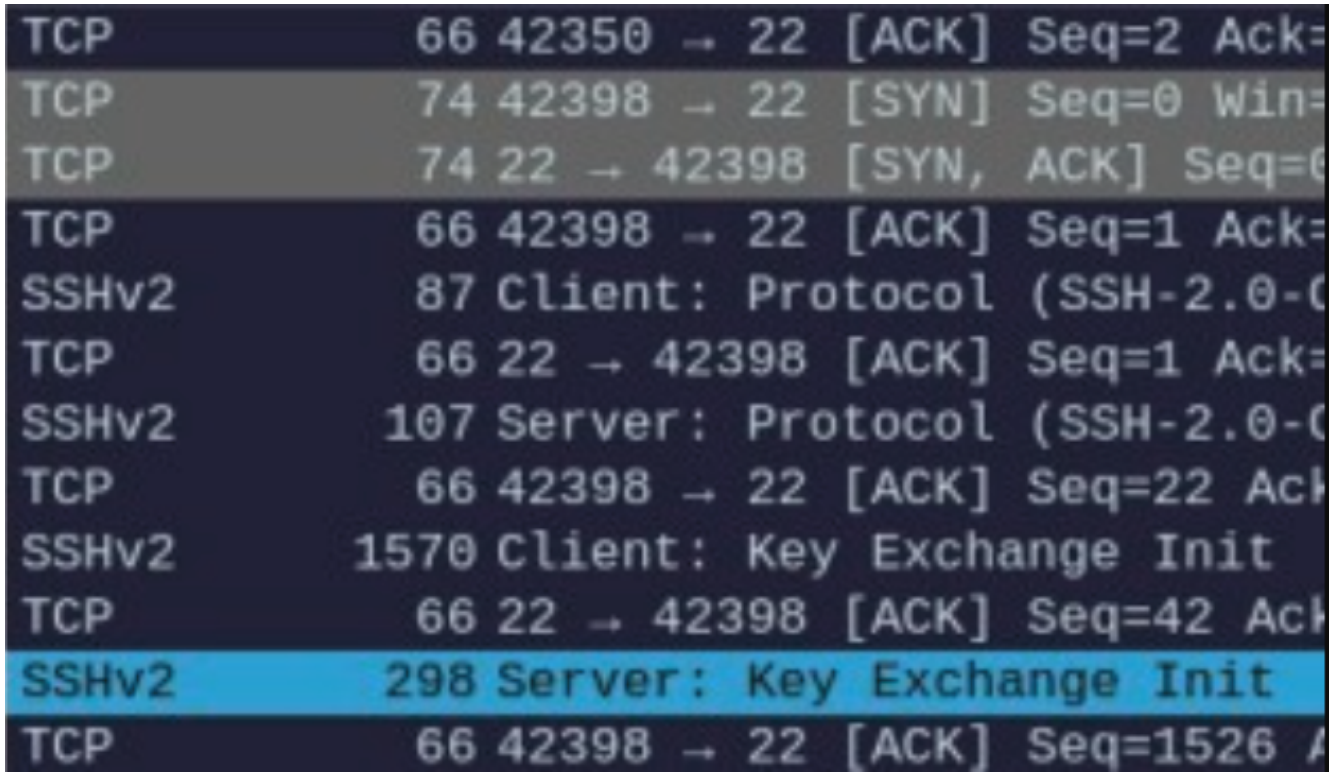


Figura 2: Captura del Key Exchange

## 2. Desarrollo (Parte 1)

### 2.1. Códigos de cada Dockerfile

#### 2.1.1. S1

Tanto S1 como C4 en base a la descripción del problema son lo mismo, solo que C4/S1 actúa tanto como cliente como servidor para las conexiones SSH. El código de C4/S1 Es el siguiente:

```
FROM ubuntu:20.10

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|' && sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu/|'
RUN apt-get update && apt-get install -y openssh-server
RUN useradd -m -d /home/test -s /bin/bash test && echo 'test:test' | chpasswd
```

```
RUN mkdir /var/run/sshd
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

### 2.1.2. C1

```
FROM ubuntu:14.10

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|'
    && sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|'
    && apt-get update && apt-get install -y openssh-server
RUN echo 'root:test' | chpasswd
RUN mkdir /var/run/sshd

CMD ["/usr/sbin/sshd", "-D"]
```

### 2.1.3. C2

```
FROM ubuntu:16.10

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|'
    && sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|'
    && apt-get update && apt-get install -y openssh-server
RUN echo 'root:test' | chpasswd
RUN mkdir /var/run/sshd
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

### 2.1.4. C3

```
FROM ubuntu:18.10

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|'
    && sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|'
    && apt-get update && apt-get install -y openssh-server
RUN echo 'root:test' | chpasswd
RUN mkdir /var/run/sshd
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

### 2.1.5. C4

```
FROM ubuntu:20.10
```

```
RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|'
    && sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|'
RUN apt-get update && apt-get install -y openssh-server
RUN useradd -m -d /home/test -s /bin/bash test && echo 'test:test' | chpasswd
RUN mkdir /var/run/sshd
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

También, utilicé un docker-compose para crear la network entre todos los contenedores y hacer un levantamiento limpio y rápido con un solo comando.

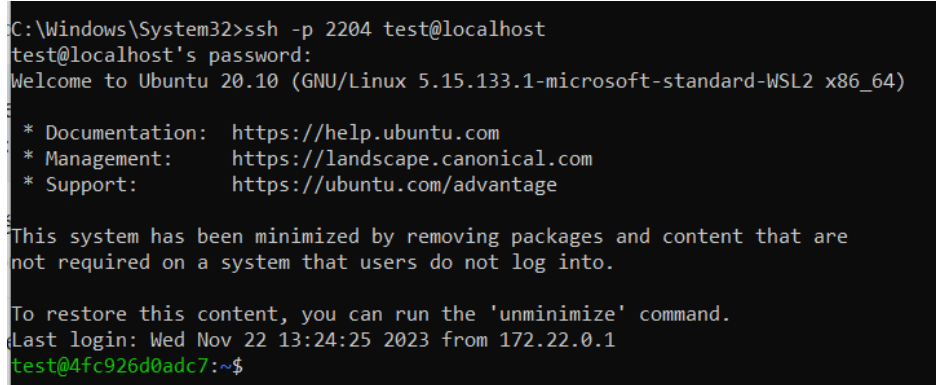
```
version: "3"

services:
  c1:
    build:
      context: .
      dockerfile: dockerfile-c1
    networks:
      - mi_red

  c2:
    build:
      context: .
      dockerfile: dockerfile-c2
    networks:
      - mi_red

  c3:
    build:
      context: .
      dockerfile: dockerfile-c3
    networks:
      - mi_red

  s1:
    build:
      context: .
      dockerfile: dockerfile-s1
    ports:
      - "2204:22"
    networks:
```



```
C:\Windows\System32>ssh -p 2204 test@localhost
test@localhost's password:
Welcome to Ubuntu 20.10 (GNU/Linux 5.15.133.1-microsoft-standard-WSL2 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 13:24:25 2023 from 172.22.0.1
test@4fc926d0adc7:~$
```

Figura 3: Captura de cómo ingresar a una máquina de docker.

```
- mi_red

networks:
  mi_red:
    driver: bridge
```

## 2.2. Creación de las credenciales para S1

Las credenciales para S1 fueron creadas en el mismo dockerfile con el siguiente comando:

```
RUN useradd -m -d /home/test -s /bin/bash test && echo 'test:test' | chpasswd
```

## 2.3. Tráfico generado por C1 (detallado)

Para realizar esto, utilizamos tshark en la máquina C4, que será a la que nos conectaremos. Esto, teniendo en escucha wireshark desde el host hacia los contenedores en docker.

El tráfico lo podemos encontrar en 4. En donde podemos ver primero cómo se genera el three-way handshake. Donde se envía un paquete TCP con bandera SYN donde el cliente responde con un TCP [SYN,ACK] reconociendo la solicitud y solicitando completar el handshake. A lo que el cliente responde con un ACK confirmando y completando el proceso para la conexión TCP.

Luego, y lo más importante es que los paquetes 78 y 89 muestran el inicio del protocolo SSH versión 2 con los mensajes Client:Protocolz "Server:Protocol". Estos mensajes contienen la identificación de las versiones de SSH que están usando el cliente y el servidor, aún no nos encontramos en el intercambio de información sensible.

En los paquetes 82 y 84 vemos el inicio de intercambio de claves Diffie-Hellman (KEX). El cliente inicia la negociación de la criptografía para la conexión SSH y luego el servidor responde con la lista de algoritmos criptográficos que cada parte está dispuesta a utilizar.

75	302.577277496	172.22.0.4	172.22.0.5	TCP	74 52588 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=22555
76	302.578558994	172.22.0.5	172.22.0.4	TCP	74 22 → 52588 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM
77	302.578734111	172.22.0.4	172.22.0.5	TCP	66 52588 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2255545806 TSecr=1
78	302.592170202	172.22.0.4	172.22.0.5	SSHv2	100 Client: Protocol (SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-8)
79	302.592196725	172.22.0.5	172.22.0.4	TCP	66 22 → 52588 [ACK] Seq=1 Ack=35 Win=65152 Len=0 TSval=1972847921 TSecr=
80	302.749261268	172.22.0.5	172.22.0.4	SSHv2	107 Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
81	302.749650861	172.22.0.4	172.22.0.5	TCP	66 52588 → 22 [ACK] Seq=35 Ack=42 Win=64256 Len=0 TSval=2255545977 TSecr=
82	302.751195483	172.22.0.4	172.22.0.5	SSHv2	2034 Client: Key Exchange Init
83	302.751251847	172.22.0.5	172.22.0.4	TCP	66 22 → 52588 [ACK] Seq=42 Ack=2003 Win=64000 Len=0 TSval=1972848080 TSe
84	302.765703905	172.22.0.5	172.22.0.4	SSHv2	1122 Server: Key Exchange Init
85	302.767359860	172.22.0.4	172.22.0.5	SSHv2	114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
86	302.777771200	172.22.0.5	172.22.0.4	SSHv2	346 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
87	302.826661482	172.22.0.4	172.22.0.5	TCP	66 52588 → 22 [ACK] Seq=2051 Ack=1378 Win=64128 Len=0 TSval=2255546054 T

Figura 4: Tráfico generado al realizar una conexión ssh desde C1 hasta C4/S1

Luego, en los paquetes 85 y 87 tenemos el intercambio de claves utilizando criptografía curva elíptica.

Respecto a la longitud:

- Handshake: 74 bits ambos.
- Inicio del protocolo SSH: 100 bits el cliente y 107 el servidor. Es más largo el del servidor porque la versión de su SO es más larga.
- Negociación de claves: El cliente 2034 bits y el servidor responde con 1122. La larga longitud de estos paquetes se debe a que en ellos yacen la lista de algoritmos con los que puede intercambiar el cliente al cual el servidor le responde con los que el le puede responder que coinciden con los del cliente.
- Intercambio de claves: Para el cliente 114 bits y para el servidor 346 bits.

Desde ahora solo mostraré las longitudes de los mensajes sin la explicación ya que en los siguientes casos es similar.

## 2.4. Tráfico generado por C2 (detallado)

Respecto a la longitud:

- Handshake: 74 bits ambos.
- Inicio del protocolo SSH: 107 bits el cliente y 107 el servidor. Es más largo el del servidor porque la versión de su SO es más larga.
- Negociación de claves: El cliente 1498 bits y el servidor responde con 1122. La larga longitud de estos paquetes se debe a que en ellos yacen la lista de algoritmos con los que puede intercambiar el cliente al cual el servidor le responde con los que el le puede responder que coinciden con los del cliente.
- Intercambio de claves: Para el cliente 114 bits y para el servidor 574 bits.



166 592.671055132 172.22.0.2	172.22.0.5	TCP	74 53578 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=16180
167 592.673382258 172.22.0.5	172.22.0.2	TCP	74 22 → 53578 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM
168 592.673666858 172.22.0.2	172.22.0.5	TCP	66 53578 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=161809399 TSecr=
169 592.680909913 172.22.0.2	172.22.0.5	SSHv2	107 Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-lubuntu0.1)
170 592.689724780 172.22.0.5	172.22.0.2	TCP	66 22 → 53578 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3690734489 TSecr=
171 592.795997751 172.22.0.5	172.22.0.2	SSHv2	107 Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-lubuntu0.1)
172 592.798626410 172.22.0.2	172.22.0.5	TCP	66 53578 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=16180939489 TSecr=
173 592.793779857 172.22.0.2	172.22.0.5	SSHv2	1498 Client: Key Exchange Init
174 592.761799966 172.22.0.5	172.22.0.2	TCP	66 22 → 53578 [ACK] Seq=42 Ack=1474 Win=64128 Len=0 TSval=3690734561 TSe
175 592.769892858 172.22.0.5	172.22.0.2	SSHv2	1122 Server: Key Exchange Init
176 592.770647078 172.22.0.2	172.22.0.5	SSHv2	114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
177 592.777365643 172.22.0.5	172.22.0.2	SSHv2	574 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
178 592.82910739 172.22.0.2	172.22.0.5	TCP	66 53578 → 22 [ACK] Seq=1522 Ack=1606 Win=64128 Len=0 TSval=1618093953 T

Figura 5: Tráfico generado al realizar una conexión ssh desde C2 hasta C4/S1

203 676.881240903 172.22.0.3	172.22.0.5	TCP	74 56180 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=10125
204 676.882174926 172.22.0.5	172.22.0.3	TCP	74 22 → 56180 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM
205 676.882878131 172.22.0.3	172.22.0.5	TCP	66 56180 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1012516942 TSecr=3
206 676.907447474 172.22.0.3	172.22.0.5	SSHv2	107 Client: Protocol (SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3)
207 676.907703467 172.22.0.5	172.22.0.3	TCP	66 22 → 56180 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3819717512 TSecr=
208 677.003063634 172.22.0.5	172.22.0.3	SSHv2	107 Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-lubuntu0.1)
209 677.003141566 172.22.0.3	172.22.0.5	TCP	66 56180 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=1012517062 TSecr=
210 677.024116585 172.22.0.3	172.22.0.5	SSHv2	1426 Client: Key Exchange Init
211 677.026090172 172.22.0.5	172.22.0.3	TCP	66 22 → 56180 [ACK] Seq=42 Ack=1402 Win=64128 Len=0 TSval=3819717628 TSe
212 677.070653755 172.22.0.5	172.22.0.3	SSHv2	1122 Server: Key Exchange Init
213 677.073667521 172.22.0.3	172.22.0.5	SSHv2	114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
214 677.073864395 172.22.0.5	172.22.0.3	TCP	66 22 → 56180 [ACK] Seq=1098 Ack=1450 Win=64128 Len=0 TSval=3819717678 T
215 677.091119815 172.22.0.5	172.22.0.3	SSHv2	574 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
216 677.138345134 172.22.0.3	172.22.0.5	TCP	66 56180 → 22 [ACK] Seq=1450 Ack=1606 Win=64128 Len=0 TSval=1012517198 T

Figura 6: Tráfico generado al realizar una conexión ssh desde C3 hasta C4/S1

## 2.5. Tráfico generado por C3 (detallado)

Respecto a la longitud:

- Handshake: 74 bits ambos.
- Inicio del protocolo SSH: 107 bits el cliente y 107 el servidor.
- Negociación de claves: El cliente 1426 bits y el servidor responde con 1122.
- Intercambio de claves: Para el cliente 114 bits y para el servidor 574 bits.

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

Para hacer esto, la máquina host no es capaz de escuchar esta solicitud, debido a que como el contenedor se está comunicando consigo mismo, la solicitud no sale a la red. En este caso, se instaló t-shark para ver qué estaba sucediendo. Respecto a la longitud:

- Handshake: 76 bits ambos.
- Inicio del protocolo SSH: 109 bits el cliente y 109 el servidor.

77 15.862824186 172.22.0.5	172.22.0.5	TCP	76 43006 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=
78 15.862828330 172.22.0.5	172.22.0.5	TCP	76 22 → 43006 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK
79 15.862869345 172.22.0.5	172.22.0.5	TCP	66 43006 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3690176389 TS
80 15.873521438 172.22.0.5	172.22.0.5	SSHv2	109 Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-lubuntu0.1)
81 15.873521438 172.22.0.5	172.22.0.5	TCP	66 22 → 43006 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=3690176400 TS
82 15.885048290 172.22.0.5	172.22.0.5	SSHv2	109 Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-lubuntu0.1)
83 15.885097345 172.22.0.5	172.22.0.5	TCP	66 43006 → 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSval=3690176411 TS
84 15.886980380 172.22.0.5	172.22.0.5	SSHv2	1124 Server: Key Exchange Init
85 15.886938956 172.22.0.5	172.22.0.5	TCP	66 43006 → 22 [ACK] Seq=42 Ack=1098 Win=64512 Len=0 TSval=369017641
86 15.887384106 172.22.0.5	172.22.0.5	SSHv2	1580 Client: Key Exchange Init
87 15.939658686 172.22.0.5	172.22.0.5	TCP	66 22 → 43006 [ACK] Seq=1098 Ack=1554 Win=65536 Len=0 TSval=3690176
88 15.939692894 172.22.0.5	172.22.0.5	SSHv2	116 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
89 15.939728400 172.22.0.5	172.22.0.5	TCP	66 22 → 43006 [ACK] Seq=1098 Ack=1082 Win=65536 Len=0 TSval=3690176
90 15.953058371 172.22.0.5	172.22.0.5	SSHv2	576 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Ke

Figura 7: Enter Caption

Figura 8: Payload del cliente C1. Muestra el paquete donde aparecen los algoritmos que acepta el cliente

Figura 9: Payload del cliente C2. Muestra el paquete donde aparecen los algoritmos que acepta el cliente

- Negociación de claves: El cliente 1124 bits y el servidor responde con 1580.
- Intercambio de claves: Para el cliente 116 bits y para el servidor 576 bits.

Por un lado, las diferencias en el handshake no son relevantes. Lo que llama la atención es la diferencia entre la negociación de claves. Por un lado en C1 el cliente envía un paquete de 2034 bits al servidor y el servidor responde con 1122. Por otro en el C2, el cliente envía 1498 bits y el servidor responde también con 1122 bits (Véase 8 y 9). Como podemos ver en las imágenes 9 respecto al payload de C2 y 8 respecto al payload de c1, la diferencia radica en que la cantidad de algoritmos que el cliente ofrece en c1 es mucho mayor a la del cliente C2. Sin embargo, como la cantidad de bits que devuelve el servidor para ambos clientes, podríamos decir que la cantidad de algoritmos con los que se llega a el acuerdo es la misma. Algo importante también a señalar respecto al tamaño del intercambio de claves tiene sentido que no sean iguales tanto para C1 como para C2 porque al tener un gran repertorio

```

Algorithms
Cookie: d73a4531fed7c22e722f167a83e95ab9
kex_algorithms length: 304
kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp25
server_host_key_algorithms length: 290
server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-
encryption_algorithms_client_to_server length: 108
encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,
encryption_algorithms_server_to_client length: 108
encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,
mac_algorithms_client_to_server length: 213
mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.co
mac_algorithms_server_to_client length: 213
mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.co
compression_algorithms_client_to_server length: 26
compression_algorithms_client_to_server string: none,zlib@openssh.com,zlib
compression_algorithms_server_to_client length: 26
compression_algorithms_server_to_client string: none,zlib@openssh.com,zlib

```

Figura 10: Payload del cliente C3. Muestra el paquete donde aparecen los algoritmos que acepta el cliente

No.	Time	Source	Destination	Protocol	Length	Info
83	15.885097340	172.22.0.5	172.22.0.3	TCP	68	43006 → 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSval=3698176411
84	15.886098300	172.22.0.3	172.22.0.5	SSHv2	1324	Server: Key Exchange Init
85	15.886098300	172.22.0.5	172.22.0.3	TCP	68	43006 → 22 [ACK] Seq=42 Ack=1099 Win=64512 Len=0 TSval=3698176411
86	15.887364100	172.22.0.3	172.22.0.5	SSHv2	1590	Client: Key Exchange Init

Figura 11: Comparación entre el tráfico de C3(arriba) y C4(abajo) cuando se realiza la negociación de claves.

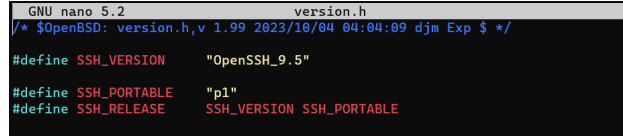
de posibles algoritmos no siempre escogerán el mismo. Lo cual puede diferenciar entre los tamaños de claves que se están intercambiando (Véase 9 y 10).

## 2.8. Diferencia entre C2 y C3

Respecto a C2 y C3 pasa algo parecido. El cliente C2 envía una cantidad ligeramente mayor de bits que C3. Enviando C2 1498 bits y C3 1426. Esto probablemente porque C3 envía menos algoritmos posibles para hacer el intercambio que C2. Respecto a lo que devuelven C2 y C3 podemos decir nuevamente que los algoritmos a los que llegan de acuerdo son los mismos. ya que el tamaño del payload es exactamente igual.

## 2.9. Diferencia entre C3 y C4

Respecto a C3 y C4 vemos algo totalmente distinto. Es el servidor quien comienza la el intercambio de claves y no el servidor cuando miramos el C4. Algo interesante también es que el tamaño del paquete de algoritmos que acepta el servidor es el mismo del del de C3, cuando este responde al cliente. Sin embargo, nuevamente vemos un cambio entre los tamaños de paquetes en donde van los algoritmos que acepta el cliente de C3 y C4 (Véase 10 y 11).



```

GNU nano 5.2                                version.h
/* $OpenBSD: version.h,v 1.99 2023/10/04 04:04:09 djm Exp $ */

#define SSH_VERSION      "OpenSSH_9.5"
#define SSH_PORTABLE     "p1"
#define SSH_RELEASE      SSH_VERSION SSH_PORTABLE

```

Figura 12: Imágen de cómo se ve el version.h originalmente.

## 3. Desarrollo (Parte 2)

### 3.1. Identificación del cliente ssh

El cliente SSH que generó ese paquete es seguramente el C4. Debido a que como se ve en la captura, el length del paquete de negociación de claves por parte del cliente es solo dos bits mayor al del que simulamos localmente. Por un lado en la captura ese paquete "Key Exchange INIT" posee 1578 bits y el de la captura que hicimos en el laboratorio tiene 1580 bits. Esto me dice que muy probablemente los algoritmos que se están compartiendo son los mismos. Por lo que tendría sentido pensar que es la misma versión del cliente SSH.

### 3.2. Replicación de tráfico (paso por paso)

La replicación es sencilla. Primero, debemos clonar un repositorio de openssh dentro de nuestro cliente. Las instrucciones para instalar en base al repositorio son las siguientes:

```

git clone https://github.com/openssh/openssh-portable # or https://anongit.mindrot
cd openssh-portable
autoreconf
./configure
make install
/usr/local/sbin/sshd

```

Sin embargo, antes de hacer el autoreconf, modificaremos el archivo version.h para replicar lo que aparece en las capturas.

Primero que todo, para clonar el repositorio necesitamos instalar git en nuestro cliente, ya que no viene por defecto. Eso lo hacemos mediante el siguiente comando:

```
apt-get install git
```

Una vez instalado, clonamos el repositorio de openssh. Continuamos entrando a la carpeta de openSSH y abrimos el archivo version.h (Véase 12). Ahora, para replicar el tráfico, lo que debemos hacer es eliminar el SSH\_PORTABLE en la última línea de código y modificamos el SSH\_VERSION agregando un 'i' a la versión. Guardamos (Véase 13).

Ahora, el cliente necesita tener instalado algunas dependencias y se deben crear unas carpetas, ya que por defecto, openssh almacena archivos en diferentes carpetas que se utilizan para la separación de privilegios. Para ello haremos:

```

/* $OpenBSD: version.h,v 1.99 2023/10/04 04:04:09 djm Exp $ */

#define SSH_VERSION "OpenSSH_9.5"
#define SSH_PORTABLE "p1"
#define SSH_RELEASE SSH_VERSION

```

Figura 13: Version.h modificado.

13 0.018689212	172.22.0.5	172.22.0.5	TCP	76 56488 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=
14 0.019155375	172.22.0.5	172.22.0.5	TCP	76 22 → 56488 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK
15 0.019240917	172.22.0.5	172.22.0.5	TCP	68 56488 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3699970975 TS
16 0.020023991	172.22.0.5	172.22.0.5	SSHv2	87 Client: Protocol (SSH-2.0-OpenSSH_?)
17 0.020028475	172.22.0.5	172.22.0.5	TCP	68 22 → 56488 [ACK] Seq=1 Ack=20 Win=65536 Len=0 TSval=3699970976 T

Figura 14: Captura de paquetes con la versión de OpenSSH con ?.

```

sudo apt update
sudo apt install git gcc make autoconf automake libtool zlib1g-dev libssl-dev
sudo mkdir -p /var/empty
sudo chown root:sys /var/empty
sudo chmod 755 /var/empty

```

Luego, podemos ejecutar los comandos para compilar la dependencia. que señalé al principio y probamos. Como podemos ver en la imagen 14 se pudo replicar la captura.

## 4. Desarrollo (Parte 3)

### 4.1. Replicación de tráfico (paso por paso)

Este caso es un poco más corto ya que como gran parte de la longitud del paquete en la negociación de la clave es ocupado por los algoritmos. Lo que haremos es dejar solo un algoritmo disponible. Para ello, vamos nuevamente el código fuente y modificamos el archivo `sshd config` (Véase ??). Se continua recompilando dentro del repositorio:

```

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

```

```

#RekeyLimit default none

Ciphers aes128-ctr
HostKeyAlgorithms ecdsa-sha2-nistp256
KexAlgorithms ecdh-sha2-
nistp256
MACs hmac-sha2-256

# Logging
#SyslogFacility AUTH

```

Figura 15: Líneas modificadas el el archivo `sshd.config`

11	0.067363766	172.22.0.5	172.22.0.5	SSHv2	1572 Client: Key Exchange Init
12	0.070394922	172.22.0.5	172.22.0.5	SSHv2	1124 Server: Key Exchange Init
13	0.072862264	172.22.0.5	172.22.0.5	SSHv2	116 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
14	0.077884447	172.22.0.5	172.22.0.5	SSHv2	976 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New

Figura 16: Captura de datos donde la longitud del paquete no fué disminuida.

Luego hacemos la captura: La cual no fué exitosa (Véase 16). Ya que la longitud del paquete del servidor sigue siendo mayor a 300. Por lo que seguramente no fué editado bien el archivo y recompilado.

## Conclusiones y comentarios

En este laboratorio, hemos abordado y demostrado exitosamente la configuración y el análisis de la seguridad en comunicaciones SSH mediante la creación y manejo de contenedores Docker con distintas versiones de Ubuntu. A través de la implementación del protocolo Diffie-Hellman, hemos establecido una base para intercambios de claves seguros.

A lo largo de las diversas pruebas, hemos capturado y analizado el tráfico de red entre los contenedores, permitiéndonos identificar las huellas digitales únicas asociadas con cada uno. Este análisis ha revelado que, a pesar de las diferencias en las versiones del sistema operativo y del software SSH, el proceso de establecimiento de la conexión SSH permanece coherente en su estructura, aunque varía en detalles como la longitud y el contenido de los paquetes intercambiados tales como los protocolos posibles que vienen y seguramente otros parámetros también.

La personalización del cliente SSH, realizada modificando el código fuente de OpenSSH y compilando una versión con una cadena de versión única, nos permitió replicar un escenario en el que podríamos identificar al informante con certeza. Esto demuestra que pequeñas alteraciones en la configuración del software pueden tener aplicaciones significativas en escenarios de seguridad informática.

En el futuro, sería prudente explorar el impacto de diferentes configuraciones y versiones de algoritmos en la seguridad y el rendimiento de la conexión SSH. Asimismo, sería beneficioso investigar cómo las mismas técnicas podrían aplicarse a otros protocolos de seguridad y autenticación.

En términos de conocimiento práctico y aplicado, este laboratorio ha sido una oportunidad importante para entender la complejidad y los desafíos de la seguridad en redes. Nos ha proporcionado una base sólida para considerar la importancia de las configuraciones seguras y la vigilancia constante en la gestión de la seguridad de la información.