

# Informe Laboratorio 5

## Sección 1

Jonathan A. Cuitiño Mendoza  
e-mail: jonathan.cuitino@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 . . . . .	5
2.1.2. C1 . . . . .	6
2.1.3. C2 . . . . .	6
2.1.4. C3 . . . . .	6
2.1.5. C4 . . . . .	7
2.2. Creación de las credenciales para S1 . . . . .	7
2.3. Tráfico generado por C1 (detallado) . . . . .	7
2.4. Tráfico generado por C2 (detallado) . . . . .	10
2.5. Tráfico generado por C3 (detallado) . . . . .	10
2.6. Tráfico generado por C4 (4 iface lo) (detallado) . . . . .	11
2.7. Diferencia entre C1 y C2 . . . . .	14
2.8. Diferencia entre C2 y C3 . . . . .	15
2.9. Diferencia entre C3 y C4 . . . . .	15
<b>3. Desarrollo (Parte 2)</b>	<b>15</b>
3.1. Identificación del cliente ssh . . . . .	15
3.2. Replicación de tráfico (paso por paso) . . . . .	16
<b>4. Desarrollo (Parte 3)</b>	<b>19</b>
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

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.

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-C			
TCP	66	22 → 42398	[ACK]	Seq=1	Ack=
SSHv2	107	Server: Protocol (SSH-2.0-C			
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	A

Figura 2: Captura del Key Exchange

## 2. Desarrollo (Parte 1)

En el presente (y ultimo) laboratorio, el informante logro establecer un medio seguro mediante el protocolo Diffie-Hellman, eliminando la necesidad de intercambiar contraseñas previamente. La tarea consiste en crear 4 contenedores en Docker con diferentes versiones de Ubuntu, cada uno equipado con las ultimas versiones de cliente y servidor OpenSSH. En el contenedor S1 (que mas adelante denotaremos también como C4) se establece un usuario llamado **test**, cuya contraseña es también el nombre de usuario, esto para permitir el acceso desde otros contenedores. De esta forma, se definen los clientes C1, C2, C3, C4 y el servidor C1.

### 2.1. Códigos de cada Dockerfile

Los contenedores C1, C2 y C3 fueron creados con exactamente el mismo código, solo cambiando la versión de ubuntu utilizada (14.10, 16.10, 18.10). El contenedor C4 (que también

actuara como S1), a diferencia de las demás versiones de sus símiles, contara con una versión de Ubuntu mas reciente, la versión 20.10, además de contar con una mayor instalación de paquetes. A continuacion, se detallan estos contenedores.

### 2.1.1. S1

El contenedor para el servidor S1 sera una instancia (al igual que C4) del mismo contenedor de docker, pues, este contenedor esta diseñado para ser cliente y servidor a la vez. A continuacion, se expone el código utilizado:

```
1 FROM ubuntu:20.10
2
3 RUN sed -i 's/http://archive.ubuntu.com/ubuntu/http://old-releases.ubuntu.com/ubuntu/' /etc/apt/sources.list
4 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/sources.list
5 RUN apt update && apt install -y sudo net-tools openssh-client openssh-server
6 RUN apt-get install -y wireshark
7 RUN apt-get install -y tshark
8 RUN useradd -m -s /bin/bash test && echo 'test:test' | chpasswd && adduser test sudo
9 RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
10
11 |
12 EXPOSE 22
13
14 CMD ["/usr/sbin/sshd", "-D"]
```

Figura 3: Dockerfile s1

Se presenta un breve detalle de los comandos utilizados:

- **FROM ubuntu:20.10:** Establece la imagen base para la construcción del contenedor. En este caso, se utiliza la imagen oficial de Ubuntu en la versión 20.10.
- **Línea 3:** Esta línea modifica el archivo `/etc/apt/release.list` para cambiar los repositorios de Ubuntu. Este paso se hace para utilizar repositorios de versiones anteriores, ya que la versión 20.10 podría haber llegado al final de su ciclo de vida y algunos paquetes pueden no estar disponibles en los repositorios estándar.
- **RUN sed -i '/deb.\*security.ubuntu.com/s/^/#/' /etc/apt/sources.list:** Comenta las líneas que se refieren a los repositorios de seguridad de Ubuntu en el archivo `/etc/apt/sources.list`. Esto se hace para evitar que se instalen actualizaciones de seguridad, lo cual puede ser útil en algunos casos específicos, aunque generalmente no se recomienda.
- **RUN apt update && apt install -y sudo net-tools openssh-client openssh-server:** Actualiza los repositorios de paquetes y luego instala una serie de paquetes.

- **RUN useradd -m -s /bin/bash test && echo 'test:test' — chpasswd && adduser test sudo:** Crea un usuario llamado "test" con contraseña "test" lo agrega al grupo sudo para otorgarle privilegios de superusuario.
- **RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd\_config:** Modifica la configuración del servidor SSH para permitir el inicio de sesión como root mediante autenticación de contraseña.
- **CMD ["/usr/sbin/sshd", "D"]:** Define el comando predeterminado que se ejecutará cuando se inicie el contenedor. En este caso, inicia el servidor SSH en modo daemon ("D")

### 2.1.2. C1

Los comandos utilizados desde acá en adelante son los explicados para el paso 2.2.1.

```
C1 > Dockerfile > ...
1 FROM ubuntu:14.10
2 RUN sed -i 's/http://archive.ubuntu.com/ubuntu/http://old-releases.ubuntu.com/ubuntu/' /etc/apt/sources.list
3 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/sources.list
4 RUN apt update && apt install -y sudo net-tools openssh-client
```

Figura 4: Dockerfile C1

### 2.1.3. C2

```
C2 > Dockerfile > ...
1 FROM ubuntu:16.10
2 RUN sed -i 's/http://archive.ubuntu.com/ubuntu/http://old-releases.ubuntu.com/ubuntu/' /etc/apt/sources.list
3 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/sources.list
4 RUN apt update && apt install -y sudo net-tools openssh-client
```

Figura 5: Dockerfile C2

### 2.1.4. C3

```
C3 > Dockerfile > ...
1 FROM ubuntu:18.10
2 RUN sed -i 's/http://archive.ubuntu.com/ubuntu/http://old-releases.ubuntu.com/ubuntu/' /etc/apt/sources.list
3 RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/sources.list
4 RUN apt update && apt install -y sudo net-tools openssh-client
```

Figura 6: Dockerfile C3

## 2.1.5. C4

```

1  FROM ubuntu:20.10
2
3  RUN sed -i 's/http://archive.ubuntu.com/ubuntu/http://old-releases.ubuntu.com/ubuntu/' /etc/apt/sources.list
4  RUN sed -i '/^deb.*security.ubuntu.com/s/^/#/' /etc/apt/sources.list
5  RUN apt update && apt install -y sudo net-tools openssh-client openssh-server
6  RUN apt-get install -y wireshark
7  RUN apt-get install -y tshark
8  RUN useradd -m -s /bin/bash test && echo 'test:test' | chpasswd && adduser test sudo
9  RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
10
11 |
12 EXPOSE 22
13
14 CMD ["/usr/sbin/sshd", "-D"]

```

Figura 7: Dockerfile C4

## 2.2. Creación de las credenciales para S1

Para la creación de las credenciales para S1, se utiliza la siguiente instrucción:

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

De esta forma, se crea un usuario llamado test, cuya contraseña también será test. Posteriormente se agrega al grupo sudo para otorgarle los privilegios de súper usuario.

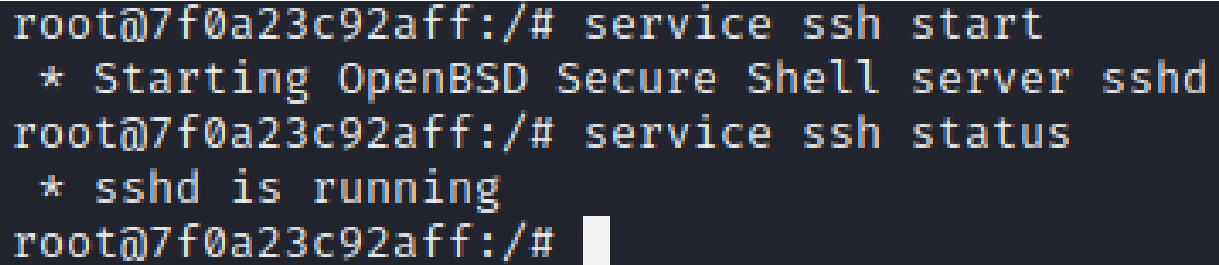
## 2.3. Tráfico generado por C1 (detallado)

Una vez listo los códigos, se debe construir la imagen para cada uno de los contenedores, los 4 clientes, y el servidor (que como se mencionó anteriormente, es el mismo que C4). Una vez que se construye la imagen, se puede acceder al contenedor. A continuación, se muestran los comandos utilizados para el C1:

```
sudo docker build -t c1 -f .dockerfile .
sudo docker run -it c1 /bin/bash
```

Posteriormente, se debe verificar que el servicio sshd esté activo para el contenedor C4S1 (Desde ahora así se denotará al contenedor C4 que también es S1). Para eso, se utilizan las siguientes instrucciones:

```
service ssh status
service ssh start
```

A terminal window with a dark background and light-colored text. The text shows a user at a root prompt starting and checking the status of the SSH service.

```
root@7f0a23c92aff:/# service ssh start
* Starting OpenBSD Secure Shell server sshd
root@7f0a23c92aff:/# service ssh status
* sshd is running
root@7f0a23c92aff:/#
```

Figura 8: Activando servicio SSH en C4S1

De esta forma, verificamos que del lado del servidor este habilitada esta configuracion. Posteriormente, y a través de wireshark se monitorean los paquetes enviados entre la consola C1 y el servidor C4S1. Una vez esta abierto wireshark y capturando los paquetes de la red *docker0*, desde la consola C1 se procede con la conexión, utilizando el siguiente comando:

```
ssh test@172.27.0.2
```

La contraseña del usuario **test**, como se vio anteriormente, es también test. De esta manera, se logra la conexión, tal como se aprecia en la imagen siguiente, y se capturan los paquetes en wireshark.



```

root@49cc36dff9d4:/# sudo ssh test@172.17.0.2
The authenticity of host '172.17.0.2 (172.17.0.2)' can't be established.
ECDSA key fingerprint is 90:90:84:df:85:69:db:6e:78:81:81:36:47:53:ad:81.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.17.0.2' (ECDSA) to the list of known hosts.
test@172.17.0.2's password:
Welcome to Ubuntu 20.10 (GNU/Linux 5.18.0-kali7-amd64 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

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

test@7f0a23c92aff:~$

```

Figura 9: Estableciendo conexión C1 con S1(C4)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.5	172.17.0.2	TCP	74	45346 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=372854714 TSecr=0 WS=128
2	0.000017057	172.17.0.2	172.17.0.5	TCP	74	22 → 45346 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2240015555 TSecr=372
3	0.000028358	172.17.0.5	172.17.0.2	TCP	66	45346 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=372854714 TSecr=2240015555
4	0.000205956	172.17.0.5	172.17.0.2	SSHv2	100	Client: Protocol (SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-8)
5	0.000212760	172.17.0.2	172.17.0.5	TCP	66	22 → 45346 [ACK] Seq=1 Ack=35 Win=65152 Len=0 TSval=2240015555 TSecr=372854714
6	0.000279729	172.17.0.2	172.17.0.5	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
7	0.000743075	172.17.0.5	172.17.0.2	TCP	66	45346 → 22 [ACK] Seq=35 Ack=42 Win=64256 Len=0 TSval=372854721 TSecr=2240015562
8	0.007098624	172.17.0.5	172.17.0.2	SSHv2	2034	Client: Key Exchange Init
9	0.007443811	172.17.0.2	172.17.0.5	SSHv2	1122	Server: Key Exchange Init
10	0.009138946	172.17.0.5	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11	0.012667270	172.17.0.2	172.17.0.5	SSHv2	346	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
12	0.015234202	172.17.0.5	172.17.0.2	SSHv2	82	Client: New Keys
13	0.057806253	172.17.0.2	172.17.0.5	TCP	66	22 → 45346 [ACK] Seq=1378 Ack=2067 Win=64128 Len=0 TSval=2240015613 TSecr=372854729
14	0.057851423	172.17.0.5	172.17.0.2	SSHv2	122	Client: Encrypted packet (len=56)
15	0.057874387	172.17.0.2	172.17.0.5	TCP	66	22 → 45346 [ACK] Seq=1378 Ack=2123 Win=64128 Len=0 TSval=2240015613 TSecr=372854772
16	0.058045191	172.17.0.2	172.17.0.5	SSHv2	122	Server: Encrypted packet (len=56)
17	0.058225754	172.17.0.5	172.17.0.2	SSHv2	138	Client: Encrypted packet (len=72)
18	0.063806681	172.17.0.2	172.17.0.5	SSHv2	122	Server: Encrypted packet (len=56)
19	0.109786669	172.17.0.5	172.17.0.2	TCP	66	45346 → 22 [ACK] Seq=2195 Ack=1490 Win=64128 Len=0 TSval=372854824 TSecr=2240015619

Frame 4: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface docker0, id 0  
 Ethernet II, Src: 02:42:ac:11:00:05 (02:42:ac:11:00:05), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)  
 Internet Protocol Version 4, Src: 172.17.0.5, Dst: 172.17.0.2  
 Transmission Control Protocol, Src Port: 45346, Dst Port: 22, Seq: 1, Ack: 1, Len: 34  
 SSH Protocol

Figura 10: Paquetes capturados con wireshark para la conexión C1-S1

Es importante apreciar de esta captura que la llave *Key Exchange Init* pesa 2024 Bytes.

## 2.4. Tráfico generado por C2 (detallado)

Los pasos para c2 son exactamente los mismos que los hechos en C1, y también los que se harán en c3, de esta manera, no se entrara en detalles de los pasos, y se mostrara la captura de wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:42:ac:11:00:03	Broadcast	ARP	42	Who has 172.17.0.2? Tell 172.17.0.3
2	0.000027256	02:42:ac:11:00:02	02:42:ac:11:00:03	ARP	42	172.17.0.2 is at 02:42:ac:11:00:02
3	0.000049098	172.17.0.3	172.17.0.2	TCP	74	45610 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3188691358 TSecr=0 WS=128
4	0.000060828	172.17.0.2	172.17.0.3	TCP	74	22 → 45610 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1430342322 TSecr=
5	0.000070964	172.17.0.3	172.17.0.2	TCP	66	45610 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3188691358 TSecr=1430342322
6	0.000359313	172.17.0.3	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)
7	0.000357016	172.17.0.2	172.17.0.3	TCP	66	22 → 45610 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=1430342323 TSecr=3188691359
8	0.000224060	172.17.0.2	172.17.0.3	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
9	0.000234836	172.17.0.3	172.17.0.2	TCP	66	45610 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=3188691364 TSecr=1430342328
10	0.000755136	172.17.0.3	172.17.0.2	SSHv2	1498	Client: Key Exchange Init
11	0.000919863	172.17.0.2	172.17.0.3	SSHv2	1122	Server: Key Exchange Init
12	0.000445232	172.17.0.3	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
13	0.011533360	172.17.0.2	172.17.0.3	SSHv2	574	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=228)
14	0.054730724	172.17.0.3	172.17.0.2	TCP	66	45610 → 22 [ACK] Seq=1522 Ack=1606 Win=64128 Len=0 TSval=3188691413 TSecr=1430342334
15	4.390018470	172.17.0.3	172.17.0.2	SSHv2	82	Client: New Keys
16	4.430755985	172.17.0.2	172.17.0.3	TCP	66	22 → 45610 [ACK] Seq=1606 Ack=1538 Win=64128 Len=0 TSval=1430346753 TSecr=3188695748
17	4.430794961	172.17.0.3	172.17.0.2	SSHv2	110	Client: Encrypted packet (len=44)
18	4.430820061	172.17.0.2	172.17.0.3	TCP	66	22 → 45610 [ACK] Seq=1606 Ack=1582 Win=64128 Len=0 TSval=1430346753 TSecr=3188695789
19	4.430974942	172.17.0.2	172.17.0.3	SSHv2	110	Server: Encrypted packet (len=44)

\* Frame 10: 1498 bytes on wire (11984 bits), 1498 bytes captured (11984 bits) on interface docker0, id 0  
 \* Ethernet II, Src: 02:42:ac:11:00:03 (02:42:ac:11:00:03), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)  
 \* Internet Protocol Version 4, Src: 172.17.0.3, Dst: 172.17.0.2  
 \* Transmission Control Protocol, Src Port: 45610, Dst Port: 22, Seq: 42, Ack: 42, Len: 1432  
 \* SSH Protocol

Figura 11: Paquetes capturados con wireshark para la conexión C2-S1

Al igual que en la conexión anterior, se aprecia acá el intercambio de llaves entre ambas partes de la comunicación. Se ve de esta manera como funciona el protocolo de intercambio de claves **Diffie-Hellman**, en el que no existe un intercambio explícito de la clave secreta, si no que esta es calculada a través de logaritmos discretos en campos finitos. Ambas partes generan aleatoriamente sus propios números privados y realizan operaciones matemáticas en un campo finito para calcular un número público. Estos números públicos se intercambian entre las partes y se combinan con sus propios números privados para calcular una clave secreta compartida.

Lo notable es que, aunque los números públicos se transmiten sin cifrar, calcular la clave secreta sin conocer los números privados resulta computacionalmente impracticable. De esta manera, el algoritmo de Diffie-Hellman proporciona un método seguro para que dos partes acuerden una clave secreta a través de un canal no seguro.

Se destaca, de esta manera, que en esta ocasión, el tamaño de la llave es 1498 bytes.

## 2.5. Tráfico generado por C3 (detallado)

De la misma manera que en los puntos explicados anteriormente, se establece la conexión con el intercambio de llaves pertinente, consiguiendo la siguiente captura.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:42:ac:11:00:04	Broadcast	ARP	42	Who has 172.17.0.2? Tell 172.17.0.4
2	0.000025607	02:42:ac:11:00:02	02:42:ac:11:00:04	ARP	42	172.17.0.2 is at 02:42:ac:11:00:02
3	0.000048369	172.17.0.4	172.17.0.2	TCP	74	40026 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2
4	0.000059978	172.17.0.2	172.17.0.4	TCP	74	22 → 40026 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PER
5	0.000076566	172.17.0.4	172.17.0.2	TCP	66	40026 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2130733890 TSecr
6	0.000358508	172.17.0.4	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3)
7	0.000365350	172.17.0.2	172.17.0.4	TCP	66	22 → 40026 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3234804339 TSec
8	0.000445624	172.17.0.2	172.17.0.4	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
9	0.000457351	172.17.0.4	172.17.0.2	TCP	66	40026 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=2130733896 TSe
10	0.007189634	172.17.0.2	172.17.0.4	SSHv2	1122	Server: Key Exchange Init
11	0.007196206	172.17.0.4	172.17.0.2	TCP	66	40026 → 22 [ACK] Seq=42 Ack=1098 Win=64128 Len=0 TSval=2130733897 T
12	0.007336408	172.17.0.4	172.17.0.2	SSHv2	1426	Client: Key Exchange Init
13	0.048328226	172.17.0.2	172.17.0.4	TCP	66	22 → 40026 [ACK] Seq=1098 Ack=1402 Win=64128 Len=0 TSval=3234804387
14	0.048376300	172.17.0.4	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
15	0.048397767	172.17.0.2	172.17.0.4	TCP	66	22 → 40026 [ACK] Seq=1098 Ack=1450 Win=64128 Len=0 TSval=3234804387
16	0.060130329	172.17.0.2	172.17.0.4	SSHv2	574	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys,
17	0.100306958	172.17.0.4	172.17.0.2	TCP	66	40026 → 22 [ACK] Seq=1450 Ack=1606 Win=64128 Len=0 TSval=2130733996
18	2.130790469	172.17.0.4	172.17.0.2	SSHv2	82	Client: New Keys
19	2.172307608	172.17.0.2	172.17.0.4	TCP	66	22 → 40026 [ACK] Seq=1606 Ack=1466 Win=64128 Len=0 TSval=3234806511

```

> Frame 12: 1426 bytes on wire (11408 bits), 1426 bytes captured (11408 bits) on interface docker0, id 0
> Ethernet II, Src: 02:42:ac:11:00:04 (02:42:ac:11:00:04), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
> Internet Protocol Version 4, Src: 172.17.0.4, Dst: 172.17.0.2
> Transmission Control Protocol, Src Port: 40026, Dst Port: 22, Seq: 42, Ack: 1098, Len: 1360
> SSH Protocol

```

Figura 12: Paquetes capturados con wireshark para la conexión C3-S1

En esta conexión, la llave generada es de 1426 bytes.

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

En esta ocasión, la conexión y posterior captura de paquetes debe ser realizada de distinta manera, pues cabe recordar que C4 y S1 son el mismo contenedor instanciado 2 veces, con lo que la captura de paquetes debe ser efectuada como se detallara a continuación.

En primer lugar, la manera de acceder al bash del contenedor se realiza de distinta manera a los casos anteriores: mediante el siguiente comando:

```
sudo docker run --cap-add=ALL-it c4 bash
```

En este caso, se agregan todos los privilegios disponibles al contenedor. Entre estos privilegios están los privilegios para realizar operaciones de administración de red. Esto incluye configurar interfaces de red, cambiar direcciones IP, y realizar otras tareas de administración de red.

Una vez dentro del contenedor, se hace uso de la siguiente instrucción:

```
sudo dpkg-reconfigure wireshark-common
```

Este comando se utiliza comúnmente después de realizar cambios en la configuración del paquete o si se necesitan ajustar algunas opciones específicas. En nuestro caso, deseamos capturar paquetes recursivamente”, es decir, dentro de la misma interfaz.

```

root@7f0a23c92aff:/# sudo dpkg-reconfigure wireshark-common
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC contains: /etc/perl /usr/local/lib/x86_64-linux-gnu/perl/5.30.3 /usr/local/share/perl/5.30.3 /usr/lib/x86_64-linux-gnu/perl5/5.30 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl-base /usr/lib/x86_64-linux-gnu/perl/5.30 /usr/share/perl/5.30 /usr/local/lib/site_perl) at /usr/share/perl5/Debconf/FrontEnd/Readline.pm line 7.)
debconf: falling back to frontend: Teletype
Configuring wireshark-common

Dumpcap can be installed in a way that allows members of the "wireshark" system group to capture packets. This is recommended over the alternative of running Wireshark/Tshark directly as root, because less of the code will run with elevated privileges.

For more detailed information please see /usr/share/doc/wireshark-common/README.Debian.gz once the package is installed.

Enabling this feature may be a security risk, so it is disabled by default. If in doubt, it is suggested to leave it disabled.
Should non-superusers be able to capture packets? [yes/no] yes
root@7f0a23c92aff:/# usermod -a -G wireshark test

```

Figura 13: dpkg-reconfigure

Para poder capturar los paquetes necesarios, es necesario añadir a wireshark el usuario test, de la siguiente manera:

```
usermod -a -G wireshark test
```

una vez hecho esto, estará añadido a wireshark el usuario test, y podrá capturar paquetes vía tshark.

Para listar las interfaces de red disponibles que tshark puede utilizar para la captura de tráfico se utiliza el siguiente comando:

```
root@7f0a23c92aff:/# tshark -D
Running as user "root" and group "root". This could be dangerous.
1. ciscodump (Cisco remote capture)
2. dpauxmon (DisplayPort AUX channel monitor capture)
3. randpkt (Random packet generator)
4. sdjournal (systemd Journal Export)
5. sshdump (SSH remote capture)
6. udpdump (UDP Listener remote capture)
root@7f0a23c92aff:/# su test
test@7f0a23c92aff:/$ tshark -D
1. ciscodump (Cisco remote capture)
2. dpauxmon (DisplayPort AUX channel monitor capture)
3. randpkt (Random packet generator)
4. sdjournal (systemd Journal Export)
5. sshdump (SSH remote capture)
6. udpdump (UDP Listener remote capture)
```

Figura 14: tshark -D

De esta forma se identifica la red en donde se va a capturar (su test), luego de acceder a ella, se comienza la captura de paquetes de la siguiente manera:

```
root@16f16361e8a0:/# tshark -i lo -w captura.pcapng
Running as user "root" and group "root". This could be dangerous.
Capturing on 'Loopback: lo'
```

Figura 15: captura tshark C4-S1

Una vez se esta escuchando con tshark, y se capturaran los paquetes relacionados con esta interfaz, se procede a hacer la conexión con C4 a S1 de la siguiente manera:

```
ssh test@localhost
```

EL archivo *.pcapng* guardado en el contenedor de docker debe ser extraído para poder ser analizado con wireshark. Para esto, nos ayudamos de la siguiente funcionalidad de docker:

```
sudo docker cp 57d87c6066db:captura.pcapng .
```

Finalmente, la captura realizada vista a través de wireshark es la siguiente:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:42:ac:11:00:06	Broadcast	ARP	42	Who has 172.17.0.2? Tell 172.17.0.6
2	0.000039673	02:42:ac:11:00:02	02:42:ac:11:00:06	ARP	42	172.17.0.2 is at 02:42:ac:11:00:02
3	0.000057273	172.17.0.6	172.17.0.2	TCP	74	51134 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1225156886 TSecr=0 WS=128
4	0.000070635	172.17.0.2	172.17.0.6	TCP	74	22 → 51134 [SYN, ACK] Seq=0 Ack=1 Win=65152 Len=0 MSS=1460 SACK_PERM=1 TSval=3503422675 TSecr=1225156886
5	0.000082577	172.17.0.6	172.17.0.2	TCP	66	51134 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1225156886 TSecr=3503422675
6	0.000411477	172.17.0.6	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
7	0.000449164	172.17.0.2	172.17.0.6	TCP	66	22 → 51134 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3503422675 TSecr=1225156886
8	0.007358714	172.17.0.2	172.17.0.6	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
9	0.007370093	172.17.0.6	172.17.0.2	TCP	66	51134 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=1225156893 TSecr=3503422682
10	0.007511478	172.17.0.6	172.17.0.2	SSHv2	1578	Client: Key Exchange Init
11	0.008123819	172.17.0.2	172.17.0.6	SSHv2	1122	Server: Key Exchange Init
12	0.009840503	172.17.0.6	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
13	0.013170771	172.17.0.2	172.17.0.6	SSHv2	574	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=228)
14	0.055976742	172.17.0.6	172.17.0.2	TCP	66	51134 → 22 [ACK] Seq=1602 Ack=1606 Win=64128 Len=0 TSval=1225156942 TSecr=3503422688
15	5.207817625	02:42:ac:11:00:02	02:42:ac:11:00:06	ARP	42	Who has 172.17.0.6? Tell 172.17.0.2
16	5.207844393	02:42:ac:11:00:06	02:42:ac:11:00:02	ARP	42	172.17.0.6 is at 02:42:ac:11:00:06
17	16.367944739	172.17.0.6	172.17.0.2	SSHv2	82	Client: New Keys
18	16.411964617	172.17.0.2	172.17.0.6	TCP	66	22 → 51134 [ACK] Seq=1606 Ack=1618 Win=64128 Len=0 TSval=3503439087 TSecr=1225173254
19	16.411974680	172.17.0.6	172.17.0.2	SSHv2	110	Client: Encrypted packet (len=44)

Frame 10: 1578 bytes on wire (12624 bits), 1578 bytes captured (12624 bits) on interface docker0, id 0  
 Ethernet II, Src: 02:42:ac:11:00:06 (02:42:ac:11:00:06), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)  
 Internet Protocol Version 4, Src: 172.17.0.6, Dst: 172.17.0.2  
 Transmission Control Protocol, Src Port: 51134, Dst Port: 22, Seq: 42, Ack: 42, Len: 1512  
 SSH Protocol

Figura 16: Paquetes capturados con wireshark para la conexión C4-S1

Se aprecia en esta imagen que, luego de establecer la comunicación segura vía Diffie-Hellman, la llave del cliente pesa 1578 bytes.

## 2.7. Diferencia entre C1 y C2

Las diferencias entre los tráficos generados por c1 y c2 con S1, respectivamente, pueden ser encontradas al analizar los paquetes de wireshark (figura 10 y 11). Se puede apreciar que para el tráfico generado por C1, la versión de SSH es la 6.6, y el tamaño de la key del cliente es 2034 bytes, como se muestra a continuación:

4	0.000205956	172.17.0.5	172.17.0.2	SSHv2	100	Client: Protocol (SSH-2.0-OpenSSH_6.6.ip1 Ubuntu-8)
5	0.000212760	172.17.0.2	172.17.0.5	TCP	66	22 → 45346 [ACK] Seq=1 Ack=35 Win=65152 Len=0 TSval=2240015555 TSecr=372854714
6	0.0006729729	172.17.0.2	172.17.0.5	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
7	0.0006743075	172.17.0.5	172.17.0.2	TCP	66	45346 → 22 [ACK] Seq=35 Ack=42 Win=64256 Len=0 TSval=2240015562 TSecr=2240015562
8	0.0006798624	172.17.0.5	172.17.0.2	SSHv2	2034	Client: Key Exchange Init
9	0.007443811	172.17.0.2	172.17.0.5	SSHv2	1122	Server: Key Exchange Init

Figura 17: Tráfico C1

Sin embargo, para el tráfico C2-S1, la versión de OpenSSH es 7.3, y el tamaño de la llave también cambia, en este caso a 1498 bytes, tal como se aprecia a continuación.

5	0.000070964	172.17.0.3	172.17.0.2	TCP	66	45610 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3188691358 TSecr=1430342322
6	0.000350313	172.17.0.3	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)
7	0.000357016	172.17.0.2	172.17.0.3	TCP	66	22 → 45610 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=1430342323 TSecr=3188691359
8	0.0006224060	172.17.0.2	172.17.0.3	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
9	0.0006234836	172.17.0.3	172.17.0.2	TCP	66	45610 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=3188691364 TSecr=1430342328
10	0.006755136	172.17.0.3	172.17.0.2	SSHv2	1498	Client: Key Exchange Init
11	0.006919863	172.17.0.2	172.17.0.3	SSHv2	1122	Server: Key Exchange Init

Figura 18: Tráfico C2

De esta forma, las principales diferencias están en la versión de SSH utilizada para cada contenedor, y el tamaño de la key. Entre otros factores, esto se debe a la versión de ubuntu utilizada.

## 2.8. Diferencia entre C2 y C3

Las diferencias entre c2 y c3 son en esencia las mismas detalladas en el apartado anterior, por lo que se describirán a continuación:

### 1. C2:

- *Version de SSH:* Client: Protocol (SSH-2.0-OpenSSH\_7.3p1 Ubuntu-1ubuntu0.1)
- *Tamaño de la key:* 1498 bytes.

### 2. C3:

- *Version de SSH:* Client: Protocol (SSH-2.0-OpenSSH\_7.7p1 Ubuntu-4ubuntu0.3)
- *Tamaño de la key:* 1426 bytes.

## 2.9. Diferencia entre C3 y C4

### 1. C4:

- *Version de SSH:* Client: Protocol (SSH-2.0-OpenSSH\_8.3p1 Ubuntu-1ubuntu0.1)
- *Tamaño de la key:* 1498 bytes.

### 2. C3:

- *Version de SSH:* 6 0.000358508 172.17.0.4 172.17.0.2 SSHv2 107 Client: Protocol (SSH-2.0-OpenSSH\_7.7p1 Ubuntu-4ubuntu0.3)
- *Tamaño de la key:* 1578 bytes.

## 3. Desarrollo (Parte 2)

### 3.1. Identificación del cliente ssh

En el enunciado, se puede apreciar en la figura 1 que la llave *Key Exchange Init* del cliente pesa **1578 bytes**. Al analizar el tráfico generado por cada consola (y analizando también el punto previo al actual), se puede notar que la única Key Exchange Init del cliente con un tamaño de 1578 fue generada por el tráfico capturado de la consola c4. De este modo, identificamos que c4 es el cliente en cuestión, basados en ubuntu 20.10 y con una versión SSH-2.0 OpenSSH 7.7.

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

Para replicar el tráfico y modificarlo, se debe seguir paso por paso las instrucciones del contenedor de git: <https://github.com/openssh/openssh-portable>. Desde aca, se pueden identificar los siguientes pasos:

1. **Ingresar al contenedor:** Para ingresar al contenedors utiliza el siguiente comando:

```
sudo docker run --cap-add=NET_RAW --cap-add=NET_ADMIN -it c4 bash
```

2. **Instalar los paquetes necesarios:** Esto se hace dentro del contenedor, y se sigue el siguiente comando:

```
apt update
apt install autoconf libssl-dev zlib1g-dev gcc make git vim
```

3. **Clonacion del repositorio:** En tercer lugar, se debe clonar el repositorio de git:

```
git clone https://github.com/openssh/openssh-portable
```

4. **Hacer los pasos que indica el repositorio:** En este momento, se deben seguir cuidadosamente los pasos que indica el repositorio de git:

```
cd openssh-portable
vim version.h
```

Una vez en el abierto el archivo, se deben hacer 2 modificaciones fundamentales, resultando en las nuevas 2 lineas::

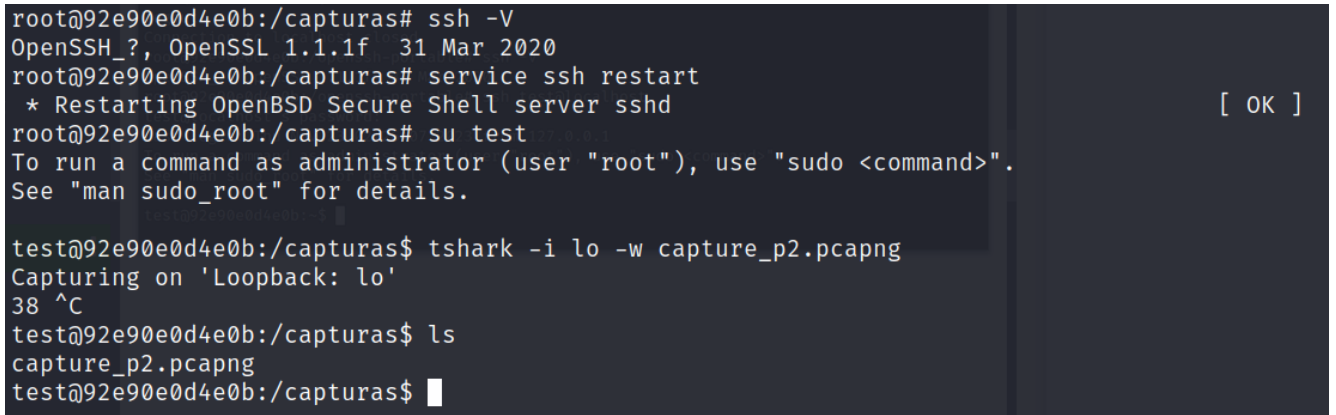
```
#define SSH_VERSION "OpenSSH.?"
#define SSH_RELEASE SSH_VERSION
```

Una vez hecha estas modificaciones se deben realizar los siguientes pasos para compilar esta nueva modificacion:

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



5. **Realizar la conexión:** Una vez compilados los cambios hechos anteriormente, se debe establecer la conexión entre c4 y s1. El proceso es el siguiente:

A terminal window with a dark background and light-colored text. The prompt is root@92e90e0d4e0b:/capturas#. The user enters 'ssh -V', showing OpenSSH and OpenSSL versions. Then 'service ssh restart' is entered, showing a restart message and '[ OK ]'. Then 'su test' is entered, showing a switch to the test user. Then 'tshark -i lo -w capture\_p2.pcapng' is entered, showing 'Capturing on 'Loopback: lo''. After a Ctrl-C interrupt, 'ls' is entered, showing 'capture\_p2.pcapng'.

```
root@92e90e0d4e0b:/capturas# ssh -V
OpenSSH_?, OpenSSL 1.1.1f 31 Mar 2020
root@92e90e0d4e0b:/capturas# service ssh restart
 * Restarting OpenBSD Secure Shell server sshd
root@92e90e0d4e0b:/capturas# su test
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

test@92e90e0d4e0b:/capturas$ tshark -i lo -w capture_p2.pcapng
Capturing on 'Loopback: lo'
38 ^C
test@92e90e0d4e0b:/capturas$ ls
capture_p2.pcapng
test@92e90e0d4e0b:/capturas$
```

Figura 19: tshark activado y escuchando en s1

En la imagen anterior, se aprecia como una de las consolas esta monitoreando la conexión, y capturando los paquetes con tshark. De esta forma, se almacenan los paquetes capturados en el archivo capture\_p2.pcapng, mediante el comando:

```
tshark -i lo -w capture_p2.pcapng
```

Posteriormente, y desde otra consola también instanciada en c4s1, se establece la conexión:

```

root@92e90e0d4e0b:/openssh-portable# ssh -V
OpenSSH_?, OpenSSL 1.1.1f 31 Mar 2020
root@92e90e0d4e0b:/openssh-portable# ssh test@localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ED25519 key fingerprint is SHA256:eIQCRLVtZU4b4e5asZHXN3yeaAbgM1m9thnURAPFTNo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
test@localhost's password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

test@92e90e0d4e0b:~$ exit
logout
Connection to localhost closed.
root@92e90e0d4e0b:/openssh-portable# ssh -V
OpenSSH_?, OpenSSL 1.1.1f 31 Mar 2020
root@92e90e0d4e0b:/openssh-portable# ssh test@localhost
test@localhost's password:
Last login: Thu Nov 23 02:26:07 2023 from 127.0.0.1
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

test@92e90e0d4e0b:~$

```

Figura 20: Estableciendo conexión con c4s1

Finalmente, los paquetes capturados en este procedimiento se muestran a continuación:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	60522 -> 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2951426355 TSecr=0 WS=128
2	0.000027253	127.0.0.1	127.0.0.1	TCP	74	22 -> 60522 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2951426355 TSecr=2951426355 WS=128
3	0.000048259	127.0.0.1	127.0.0.1	TCP	66	60522 -> 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2951426355 TSecr=2951426355
4	0.000713397	127.0.0.1	127.0.0.1	SSHv2	85	Client: Protocol (SSH-2.0-OpenSSH_?)
5	0.000724388	127.0.0.1	127.0.0.1	TCP	66	22 -> 60522 [ACK] Seq=1 Ack=20 Win=65536 Len=0 TSval=2951426356 TSecr=2951426356
6	0.010007770	127.0.0.1	127.0.0.1	SSHv2	85	Server: Protocol (SSH-2.0-OpenSSH_?)
7	0.010695959	127.0.0.1	127.0.0.1	TCP	66	60522 -> 22 [ACK] Seq=20 Ack=20 Win=65536 Len=0 TSval=2951426372 TSecr=2951426372
8	0.017593295	127.0.0.1	127.0.0.1	SSHv2	1578	Client: Key Exchange Init
9	0.018510535	127.0.0.1	127.0.0.1	SSHv2	1146	Server: Key Exchange Init
10	0.059058293	127.0.0.1	127.0.0.1	TCP	66	60522 -> 22 [ACK] Seq=1524 Ack=1100 Win=65536 Len=0 TSval=2951426414 TSecr=2951426374
11	0.123775162	127.0.0.1	127.0.0.1	SSHv2	1274	Client: Diffie-Hellman Key Exchange Init
12	0.137298826	127.0.0.1	127.0.0.1	SSHv2	1654	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=340)
13	0.137298864	127.0.0.1	127.0.0.1	TCP	66	60522 -> 22 [ACK] Seq=2732 Ack=2688 Win=64128 Len=0 TSval=2951426492 TSecr=2951426492
14	0.168582351	127.0.0.1	127.0.0.1	SSHv2	82	Client: New Keys
15	0.215102646	127.0.0.1	127.0.0.1	TCP	66	22 -> 60522 [ACK] Seq=2688 Ack=2748 Win=65536 Len=0 TSval=2951426570 TSecr=2951426524
16	0.215111021	127.0.0.1	127.0.0.1	SSHv2	110	Client: Encrypted packet (len=44)
17	0.215117374	127.0.0.1	127.0.0.1	TCP	66	22 -> 60522 [ACK] Seq=2688 Ack=2792 Win=65536 Len=0 TSval=2951426570 TSecr=2951426570
18	0.215167098	127.0.0.1	127.0.0.1	SSHv2	110	Server: Encrypted packet (len=44)
19	0.215216192	127.0.0.1	127.0.0.1	SSHv2	126	Client: Encrypted packet (len=60)

Frame 6: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface lo, id 0  
 Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
 Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 Transmission Control Protocol, Src Port: 22, Dst Port: 60522, Seq: 1, Ack: 20, Len: 19  
 SSH Protocol

Figura 21: Replicación

Se aprecia, de esta manera, como es que las modificaciones fueron correctas, y ahora la versión de SSH no esta disponible en wireshark.

## 4. Desarrollo (Parte 3)

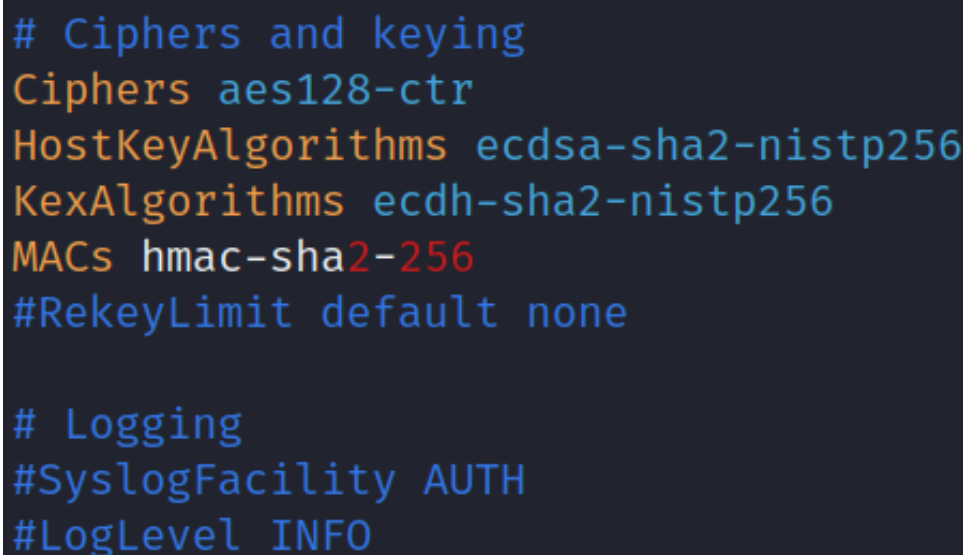
La realización de este paso es muy similar al punto anterior, solo que el archivo a modificar es distinto, detallado a continuación:

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

Al igual que en el paso anterior, se debe seguir una secuencia de pasos similar. EN primer lugar acceder al contenedor C4S1 mediante el comando:

```
sudo docker run --cap-add=ALL -it c4 bash
```

Posteriormente, se instalan los paquetes necesarios (los mismos que en el paso anterior), se hace el git clone del mismo repositorio de github, y se busca el archivo llamado **sshd\_config**. Una vez abierto este archivo, se deben insertar las siguientes 4 líneas de código:



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

# Logging
#SyslogFacility AUTH
#LogLevel INFO
```

Figura 22: Inserción de 4 especificaciones de cifrado.

Una vez modificado este archivo de configuración sshd, se procede con los mismos pasos de compilación usados anteriormente:

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

Finalmente, desde una consola se capturan los paquetes, y la otra consola realiza la conexion:

```
(john@kali)-[~/Documentos/cripto/lab_05]
$ sudo docker exec -it 57d87c6066db /bin/bash
root@57d87c6066db:/# tshark -i lo -w /tmp/capturas_p3.pcapng
Running as user "root" and group "root". This could be dangerous.
Capturing on 'Loopback: lo'
37 ^C
```

Figura 23: Capturando con tshark

Es entonces cuando los paquetes son capturados, permitiendo su posterior analisis en wireshark (luego, claro esta, de copiarlo desde el contenedor al local):

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	48660 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2952411546 TSecr=0 WS=128
2	0.000010058	127.0.0.1	127.0.0.1	TCP	74	22 → 48660 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2952411546 TSecr=2952411546 WS=128
3	0.000017603	127.0.0.1	127.0.0.1	TCP	66	48660 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2952411546 TSecr=2952411546
4	0.000176454	127.0.0.1	127.0.0.1	SSHv2	87	Client: Protocol (SSH-2.0-OpenSSH_9.5)
5	0.000179938	127.0.0.1	127.0.0.1	TCP	66	22 → 48660 [ACK] Seq=1 Ack=22 Win=65536 Len=0 TSval=2952411546 TSecr=2952411546
6	0.004723566	127.0.0.1	127.0.0.1	SSHv2	87	Server: Protocol (SSH-2.0-OpenSSH_9.5)
7	0.004729793	127.0.0.1	127.0.0.1	TCP	66	48660 → 22 [ACK] Seq=22 Ack=22 Win=65536 Len=0 TSval=2952411551 TSecr=2952411551
8	0.004886000	127.0.0.1	127.0.0.1	SSHv2	1570	Client: Key Exchange Init
9	0.005218715	127.0.0.1	127.0.0.1	SSHv2	266	Server: Key Exchange Init
10	0.005408441	127.0.0.1	127.0.0.1	SSHv2	146	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11	0.006056151	127.0.0.1	127.0.0.1	SSHv2	746	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=368)
12	0.052113941	127.0.0.1	127.0.0.1	TCP	66	48660 → 22 [ACK] Seq=1696 Ack=902 Win=65536 Len=0 TSval=2952411598 TSecr=2952411552
13	1.888280075	127.0.0.1	127.0.0.1	SSHv2	82	Client: New Keys
14	1.932185233	127.0.0.1	127.0.0.1	TCP	66	22 → 48660 [ACK] Seq=902 Ack=1622 Win=65536 Len=0 TSval=2952413478 TSecr=2952413434
15	1.932221544	127.0.0.1	127.0.0.1	SSHv2	130	Client: Encrypted packet (len=64)
16	1.932226101	127.0.0.1	127.0.0.1	TCP	66	22 → 48660 [ACK] Seq=902 Ack=1680 Win=65536 Len=0 TSval=2952413478 TSecr=2952413478
17	1.932261908	127.0.0.1	127.0.0.1	SSHv2	130	Server: Encrypted packet (len=64)
18	1.932267405	127.0.0.1	127.0.0.1	TCP	66	48660 → 22 [ACK] Seq=1686 Ack=966 Win=65536 Len=0 TSval=2952413478 TSecr=2952413478
19	1.932299295	127.0.0.1	127.0.0.1	SSHv2	146	Client: Encrypted packet (len=80)

Frame 9: 266 bytes on wire (2128 bits), 266 bytes captured (2128 bits) on interface lo, id 0  
 Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
 Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 Transmission Control Protocol, Src Port: 22, Dst Port: 48660, Seq: 22, Ack: 1526, Len: 200  
 SSH Protocol

Figura 24: Captura de wireshar ssh.config modificado

Se puede apreciar aca, entre otras diferencias, que añadiendo estas especificaciones de cifrado la llave disminuyo considerablemente su tamaño, ahora pesando **266 bytes**.

## Conclusiones y comentarios

En este laboratorio, se llevaron a cabo una serie de actividades para explorar y comprender el funcionamiento del protocolo de seguridad Diffie-Hellman en conexiones SSH. A través de la creación de contenedores Docker con diferentes versiones de Ubuntu y configuraciones específicas, se establecieron conexiones seguras entre clientes y un servidor, capturando y analizando el tráfico de red generado durante estos intercambios.

La alumna demostró habilidades para configurar entornos de red virtualizados, instalar y configurar software SSH en diferentes sistemas operativos, y utilizar herramientas como Wireshark y Tshark para analizar el tráfico de red. Además, pudo identificar patrones de tráfico y diferencias entre las versiones de SSH utilizadas en los clientes.

En la segunda parte del laboratorio, se replicó y modificó el tráfico SSH, identificando un cliente específico y realizando cambios en la versión de OpenSSH. Este ejercicio demostró su comprensión práctica de los aspectos de seguridad y la capacidad de manipular el tráfico de red.

En la tercera parte, se realizaron modificaciones en la configuración del servidor SSH (sshd.config) para cambiar las especificaciones de cifrado. Este paso mostró la capacidad de la estudiante para realizar ajustes de seguridad en la configuración del servidor y entender cómo estos cambios afectan el tráfico y la negociación de claves.

En general, la alumna adquirió experiencia práctica en la implementación y modificación de configuraciones de seguridad en entornos SSH. Estas habilidades son fundamentales para la comprensión de protocolos de seguridad, en donde los profesionales de la ciberseguridad y administradores de estos sistemas en general deben garantizar la integridad y confidencialidad de las comunicaciones en entornos de red no seguros.

Los códigos utilizados en el presente informe, además del mismo pueden ser encontrados en el siguiente enlace:

<https://github.com/iJass21/Criptografia>