

# Informe Laboratorio 5

## Sección 1

Francisco Gálvez Plaza  
e-mail: francisco.galvez\_p@mail\_udp.cl

Diciembre de 2024

# Índice

<b>Descripción de actividades</b>	<b>3</b>
<b>1. Desarrollo (Parte 1)</b>	<b>5</b>
1.1. Códigos de cada Dockerfile . . . . .	5
1.1.1. C1 . . . . .	6
1.1.2. C2 . . . . .	6
1.1.3. C3 . . . . .	7
1.1.4. C4/S1 . . . . .	8
1.2. Creación de las credenciales para S1 . . . . .	9
1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .	9
1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .	11
1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .	13
1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .	14
1.7. Tipo de información contenida en cada uno de los paquetes generados en texto plano . . . . .	16
1.7.1. C1 . . . . .	16
1.7.2. C2 . . . . .	17
1.7.3. C3 . . . . .	18
1.7.4. C4/S1 . . . . .	19
1.8. Diferencia entre C1 y C2 . . . . .	20
1.9. Diferencia entre C2 y C3 . . . . .	20
1.10. Diferencia entre C3 y C4 . . . . .	21
<b>2. Desarrollo (Parte 2)</b>	<b>22</b>
2.1. Identificación del cliente SSH con versión “?” . . . . .	22
2.2. Replicación de tráfico al servidor (paso por paso) . . . . .	23
<b>3. Desarrollo (Parte 3)</b>	<b>25</b>
3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso) . . . . .	25
<b>4. Desarrollo (Parte 4)</b>	<b>27</b>
4.1. Explicación OpenSSH en general . . . . .	27
4.2. Capas de Seguridad en OpenSSH . . . . .	28
4.3. Identificación de que protocolos no se cumplen . . . . .	29

## 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 o Podman, donde cada uno tendrá el siguiente SO: Ubuntu 16.10, Ubuntu 18.10, Ubuntu 20.10 y Ubuntu 22.10 a los cuales se llamarán C1, C2, C3 y C4 respectivamente.  
El equipo con Ubuntu 22.10 también será utilizado como S1.
- Para cada uno de ellos, deberá instalar el cliente openSSH disponible en los repositorios de apt, y para el equipo S1 deberá también instalar el servidor openSSH.
- En S1 deberá crear el usuario “**prueba**” con contraseña “**prueba**”, para acceder a él desde los clientes por el protocolo SSH.
- 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, deberá capturar el tráfico generado por cada conexión con el server. A partir de cada Handshake, deberá analizar el patrón de tráfico generado por cada cliente y adicionalmente obtener el HASSH que lo identifique. De esta forma podrá obtener una huella digital para cada cliente a partir de su tráfico. Cada HASSH deberá compararlo con la base de datos HASSH disponible en el módulo de TLS, e identificar si el hash obtenido corresponde a la misma versión de su cliente.

Indique el tamaño de los paquetes del flujo generados por el cliente y el contenido asociado a cada uno de ellos. Indique qué información distinta contiene el escenario siguiente (diff incremental). El objetivo de este paso 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. Recuerde que toda la información generada es parte del sw, por lo tanto usted puede modificar toda la información.

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

4. Tomando en cuenta lo aprendido en este laboratorio, así como en los anteriores, explique el protocolo OpenSSH y las diferentes capas de seguridad que son parte del protocolo para garantizar los principios de seguridad de la información, integridad, confidencialidad, disponibilidad, autenticidad y no repudio. Es importante que sea muy específico en el objetivo del principio en el protocolo. En caso de considerar que alguno de los principios no se cumple, justifique su razonamiento. Es fundamental que su análisis se base en el tráfico SSH interceptado.

## 1. Desarrollo (Parte 1)

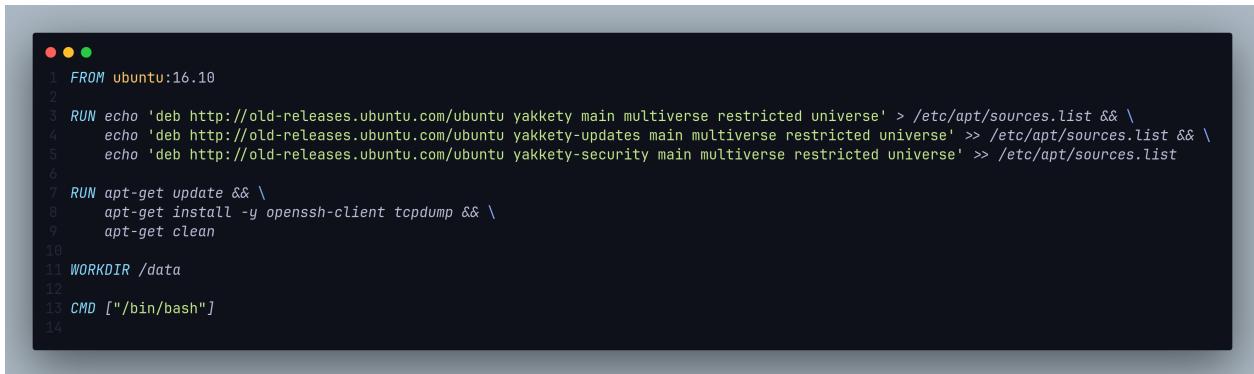
### 1.1. Códigos de cada Dockerfile

Para el desarrollo de la actividad, es crear cuatro imágenes de Ubuntu con distintas versiones utilizando Docker. Para esto se crean múltiples archivos del tipo “Dockerfile” los cuales son compilados en la computadora mediante el uso del comando `$docker build -t nombre-de-la-imagen .`, esto debe ser realizado dentro de la carpeta contenedora del archivo Dockerfile.

Una vez realizado aquello, para levantar cada uno de los contenedores, se utiliza el comando `$docker run -it --rm nombre-de-la-imagen`.

A continuación se muestran los códigos realizados para cada una de las versiones de Ubuntu utilizadas.

### 1.1.1. C1



```

1 FROM ubuntu:16.10
2
3 RUN echo 'deb http://old-releases.ubuntu.com/ubuntu yakkety main multiverse restricted universe' > /etc/apt/sources.list && \
4     echo 'deb http://old-releases.ubuntu.com/ubuntu yakkety-updates main multiverse restricted universe' >> /etc/apt/sources.list && \
5     echo 'deb http://old-releases.ubuntu.com/ubuntu yakkety-security main multiverse restricted universe' >> /etc/apt/sources.list
6
7 RUN apt-get update && \
8     apt-get install -y openssh-client tcpdump && \
9     apt-get clean
10
11 WORKDIR /data
12
13 CMD ["/bin/bash"]
14

```

Figura 3: Dockerfile para Ubuntu 16.10

Cómo se puede apreciar en la imagen, se utiliza Ubuntu en su versión 16.10, el uso de `old-releases.ubuntu.com` en el archivo indica que se está configurando el sistema basado en una versión de Ubuntu que ya no es soportada oficialmente por los repositorios principales de Ubuntu. Su implementación es necesaria para asegurar que los paquetes necesarios sigan estando disponibles y no haya problemas de compatibilidad con las herramientas a utilizar.

Además, se ejecuta una actualización y se instala el cliente de “OpenSSH”, junto con “TCPDump”. Al correr el contenedor, se utiliza la terminal de este para interactuar con el sistema.

### 1.1.2. C2



```

1 FROM ubuntu:18.10
2
3 RUN echo 'deb http://old-releases.ubuntu.com/ubuntu cosmic main multiverse restricted universe' > /etc/apt/sources.list && \
4     echo 'deb http://old-releases.ubuntu.com/ubuntu cosmic-updates main multiverse restricted universe' >> /etc/apt/sources.list && \
5     echo 'deb http://old-releases.ubuntu.com/ubuntu cosmic-security main multiverse restricted universe' >> /etc/apt/sources.list
6
7 RUN apt-get update && \
8     apt-get install -y openssh-client tcpdump && \
9     apt-get clean
10
11 WORKDIR /data
12
13 CMD ["/bin/bash"]
14

```

Figura 4: Dockerfile para Ubuntu 18.10

Al igual que para C1, en C2 se utiliza Ubuntu, pero en este caso la versión 18.10. También se hace uso de `old-releases.ubuntu.com` en el archivo. Se ejecuta una actualización y se instala el cliente de “OpenSSH”, junto con “TCPDump”. Al correr el contenedor, se utiliza la terminal de este para interactuar con el sistema.

### 1.1.3. C3

```

1  FROM ubuntu:20.10
2
3  RUN echo 'deb http://old-releases.ubuntu.com/ubuntu groovy main multiverse restricted universe' > /etc/apt/sources.list && \
4      echo 'deb http://old-releases.ubuntu.com/ubuntu groovy-updates main multiverse restricted universe' >> /etc/apt/sources.list && \
5      echo 'deb http://old-releases.ubuntu.com/ubuntu groovy-security main multiverse restricted universe' >> /etc/apt/sources.list
6
7  RUN apt-get update && \
8      apt-get install -y openssh-client tcpdump git vim autoconf libssl-dev zlib1g-dev make && \
9      apt-get clean
10
11 WORKDIR /data
12
13 CMD ["/bin/bash"]

```

Figura 5: Dockerfile para Ubuntu 20.10

Al igual que para C1 y C2, en C3 se utiliza Ubuntu, pero en este caso la versión 20.10. También se hace uso de `old-releases.ubuntu.com` en el archivo. Se ejecuta una actualización y se instala el cliente de “OpenSSH”, junto con “TCPDump”.

Sin embargo, también se requiere la instalación de:

- **Vim:** Utilizado para la edición de archivos más adelante en la actividad.
- **autoconf:** Es una herramienta para generar scripts que configuran automáticamente los paquetes de software para adaptarse a sistemas Unix-like.
- **libssl-dev:** Son bibliotecas de desarrollo para SSL, necesarias para compilar programas que usan SSL.
- **zlib1g-dev:** Bibliotecas de desarrollo para zlib, una biblioteca de compresión.
- **make:** Es una herramienta que controla la generación de “ejecutables” y otros componentes no fuente de un programa a partir de los archivos fuente.
- **git:** Permite interactuar con el entorno Git y así obtener acceso a archivos almacenados en repositorios públicos.

Al correr el contenedor, se utiliza la terminal de este para interactuar con el sistema.

### 1.1.4. C4/S1



```

1 FROM ubuntu:22.10
2
3 RUN echo 'deb http://old-releases.ubuntu.com/ubuntu kinetic main multiverse restricted universe' > /etc/apt/sources.list && \
4     echo 'deb http://old-releases.ubuntu.com/ubuntu kinetic-backports main multiverse restricted universe' >> /etc/apt/sources.list && \
5     echo 'deb http://old-releases.ubuntu.com/ubuntu kinetic-proposed main multiverse restricted universe' >> /etc/apt/sources.list && \
6     echo 'deb http://old-releases.ubuntu.com/ubuntu kinetic-updates main multiverse restricted universe' >> /etc/apt/sources.list && \
7     echo 'deb http://old-releases.ubuntu.com/ubuntu kinetic-security main multiverse restricted universe' >> /etc/apt/sources.list
8
9 RUN apt-get update && \
10    apt-get install -y openssh-client tcpdump vim autoconf libssl-dev zlib1g-dev make openssh-server && \
11    apt-get clean
12
13 RUN echo 'root:root' | chpasswd && \
14    useradd -m -d /home/prueba -s /bin/bash prueba && \
15    echo 'prueba:prueba' | chpasswd && \
16    sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config && \
17    sed -i 's/#PermitRootLogin yes/PermitRootLogin yes/' /etc/ssh/sshd_config && \
18    mkdir -p /var/run/sshd
19
20
21 WORKDIR /data
22
23 EXPOSE 22
24
25 CMD ["/usr/sbin/sshd", "-D", "-d", "-e"]

```

Figura 6: Dockerfile para Ubuntu 22.10

Al igual que para C1 y C2, en C3 se utiliza Ubuntu, pero en este caso la versión 20.10. También se hace uso de `old-releases.ubuntu.com` en el archivo.

Se ejecuta una actualización y se instala el cliente de OpenSSH, junto con TCPDump. Sin embargo, también se requiere la instalación de:

- **Vim:** Utilizado para la edición de archivos más adelante en la actividad.
- **autoconf:** Es una herramienta para generar scripts que configuran automáticamente los paquetes de software para adaptarse a sistemas Unix-like.
- **libssl-dev:** Son bibliotecas de desarrollo para SSL, necesarias para compilar programas que usan SSL.
- **zlib1g-dev:** Bibliotecas de desarrollo para zlib, una biblioteca de compresión.
- **make:** Es una herramienta que controla la generación de “ejecutables” y otros componentes no fuente de un programa a partir de los archivos fuente.
- **openssh-server:** Corresponde al servidor SSH para permitir conexiones con los otros contenedores.

Al correr el contenedor, se ejecuta en la terminal de este contenedor el comando `con` el fin de configurar y ejecutar el servidor SSH (`sshd`) con opciones específicas para interactuar con el sistema. Donde “`-D`” indica que se ejecute el servidor en primer plano. Con “`-d`” se ejecuta el modo de depuración. Al utilizar esta opción, `sshd` imprime

mensajes de depuración detallados sobre su progreso. Finalmente “**-e**”, esta opción hace que **sshd** envíe la salida de los logs al **stderr** estándar, permitiendo que los logs se capturen más fácilmente

## 1.2. Creación de las credenciales para S1

Tal como se vió anteriormente, en el Dockerfile de Ubuntu 22.10, se encuentra la siguiente configuración.



```

1 RUN echo 'root:root' | chpasswd && \
2     useradd -m -d /home/prueba -s /bin/bash prueba && \
3     echo 'prueba:prueba' | chpasswd && \
4     sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config && \
5     sed -i 's/#PermitRootLogin yes/PermitRootLogin yes/' /etc/ssh/sshd_config && \
6     mkdir -p /var/run/sshd

```

Figura 7: Credenciales para S1.

Como se puede apreciar, en este proceso se crean y configuran las credenciales de usuario y los ajustes de seguridad en el servidor SSH. Primero, se establece la contraseña del usuario root como “root”. Después, se crea un nuevo usuario llamado ‘prueba’ y se configura un directorio de inicio para él. Se asigna la contraseña ‘prueba’ al usuario ‘prueba’. Finalmente, se modifican los archivos de configuración para permitir el inicio de sesión del usuario root y se habilita la autenticación por contraseña en el servidor SSH.

**Nota:** Cabe mencionar que durante la ejecución de los contenedores, esta no se realizó de manera simultanea, esto por comodidad a la hora de realizar el análisis y ejecución de las conexiones. Debido a esto, es que para los clientes generalmente se le asigna la misma IP a estos, mientras que el servidor posee una distinta.

## 1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

A continuación, se presenta una captura de tráfico realizada utilizando el software “Wireshark”, donde posible apreciar el “Handshake” generado al conectar el cliente C1 con el servidor S1.

### 1.3 Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

#### 1 DESARROLLO (PARTE 1)

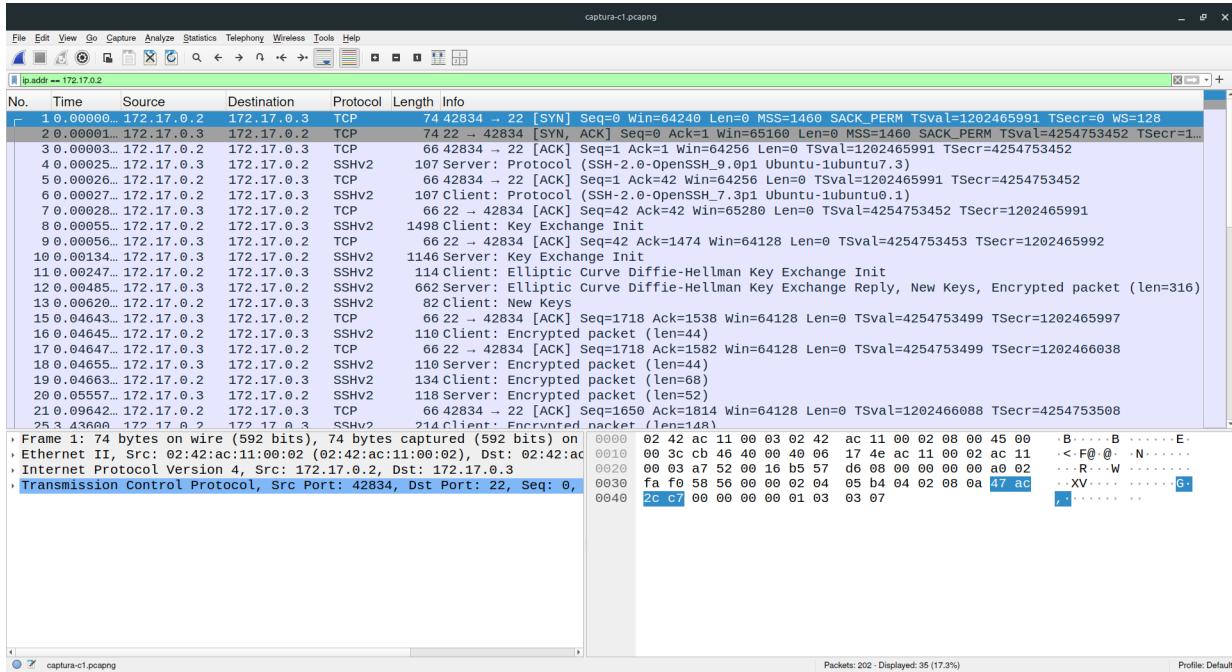


Figura 8: Captura de Wireshark para Hanshake entre C1 y S1.

A partir de esto se menciona la presencia de los siguientes paquetes:

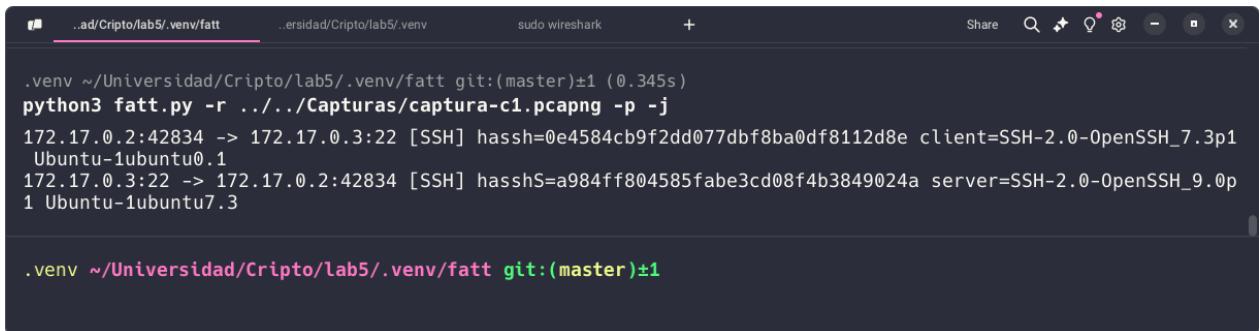
- Paquete 1: TCP [SYN] - Este paquete inicia la conexión TCP. El paquete tiene un tamaño de 74 bytes y establece los números de secuencia iniciales.
- Paquete 3: TCP [ACK] - Completa el handshake TCP, estableciendo la conexión. Este paquete es una respuesta al paquete SYN y tiene un tamaño de 66 bytes.
- Paquete 6: SSH-2.0-Cliente - Indica la versión del cliente SSH, en este caso OpenSSH 7.7p1 Ubuntu-4ubuntu0.3. El tamaño del paquete es de 107 bytes.
- Paquete 8: SSH Key Exchange Init. Inicia el intercambio de claves entre el cliente y el servidor. (1498 Bytes)
- Paquete 11: Key Exchange Init del servidor utilizando Elliptic Curve Diffie-Hellman (114 bytes).
- Paquete 13: New Keys. Este paquete confirma la instalación de las nuevas claves por parte del cliente. (82 Bytes)

Esta captura de paquetes muestra el proceso de establecimiento de la conexión SSH entre C1 y S1, incluyendo la negociación de los algoritmos de cifrado y la generación de nuevas claves.

Luego, a forma de obtener el HASSH correspondiente generado, se hace uso de “FATT” (Fingerprint All The Things), el cual es un script basado en Pyshark que se utiliza para extraer metadatos de red y huellas dactilares de archivos pcap y tráfico de red en vivo. Para esto es necesario copiar el repositorio en el equipo y utilizar el siguiente comando en la consola: `python3 fatt.py -r <ruta-del-archivo.pcapng>-p -j`

Lo que hace dicha linea, es ejecutar el script FATT en Python, donde `-r <ruta-del-archivo.pcapng>` especifica la captura de Wireshark que el script debe procesar. `-p` indica al script que imprima los resultados de la ejecución en la pantalla del terminal desde donde se ejecuta el comando y por último, `-j` le indica al script que registre la salida en formato JSON.

De esta forma se obtiene el siguiente registro:



```
..ad/Cripto/lab5/.venv/fatt ..ersidad/Cripto/lab5/.venv sudo wireshark + Share Q - X
.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1 (0.345s)
python3 fatt.py -r ../../Capturas/captura-c1.pcapng -p -j
172.17.0.2:42834 -> 172.17.0.3:22 [SSH] hassh=0e4584cb9f2dd077dbf8ba0df8112d8e client=SSH-2.0-OpenSSH_7.3p1
Ubuntu-1ubuntu0.1
172.17.0.3:22 -> 172.17.0.2:42834 [SSH] hasshS=a984ff804585fabe3cd08f4b3849024a server=SSH-2.0-OpenSSH_9.0p
1 Ubuntu-1ubuntu7.3

.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1
```

Figura 9: HASSH generado para Handshake entre C1 y S1.

“HASSH” se refiere a un hash de la configuración del protocolo SSH, utilizado tanto por clientes como servidores. El primer hash (hassh) proviene de la configuración del cliente SSH, indicando que el cliente es ”SSH-2.0-OpenSSH\_7.3p1 Ubuntu-1ubuntu0.1”. El segundo hash (hasshS) proviene de la configuración del servidor SSH, que en este caso es ”SSH-2.0-OpenSSH\_9.0p1 Ubuntu-1ubuntu7.3”.

Como es posible ver en la figura 9, el HASSH generado para C1 es `0e4584cb9f2dd077dbf8ba0df8112d8e`

#### 1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

Para la captura de tráfico obtenida, donde posible apreciar el “Handshake” generado al conectar el cliente C2 con el servidor S1, se obtiene:

## 1.4 Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

### 1 DESARROLLO (PARTE 1)

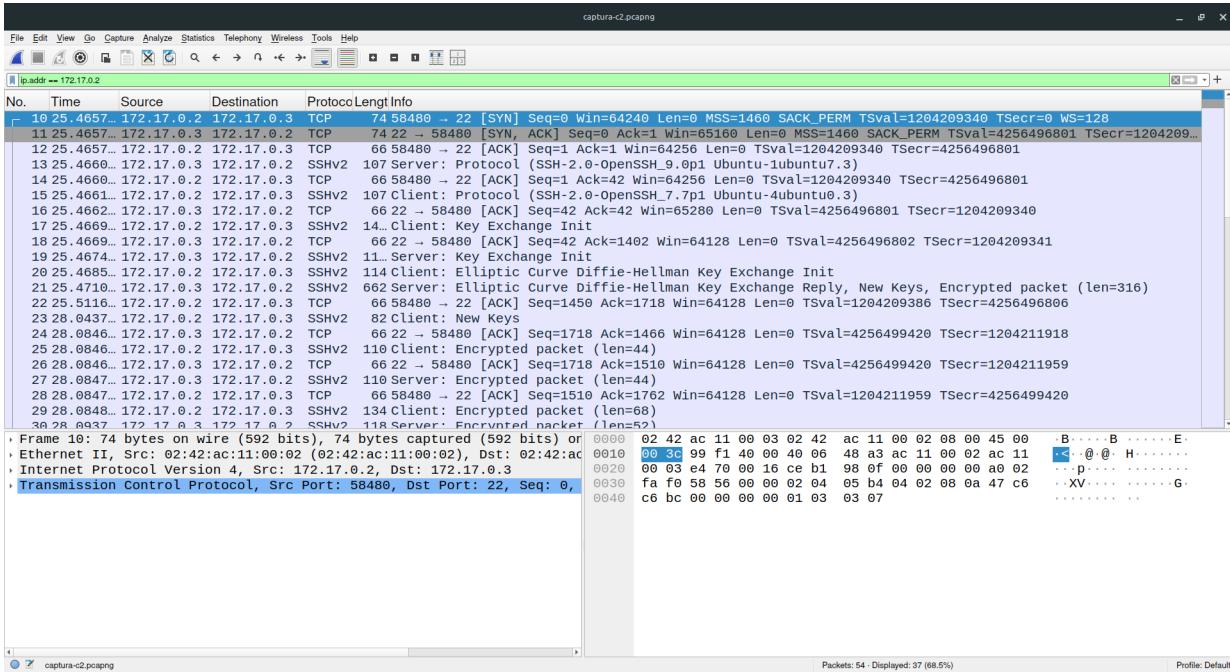


Figura 10: Captura de Wireshark para Handshake entre C2 y S1.

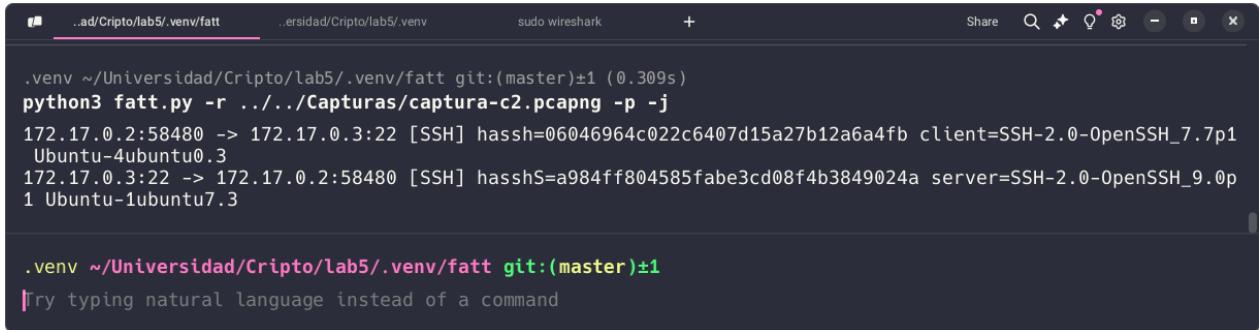
- Paquete 10: TCP [SYN] - Este paquete inicia la conexión TCP. El paquete tiene un tamaño de 74 bytes y establece los números de secuencia iniciales.
- Paquete 12: TCP [ACK] - Completa el handshake TCP, estableciendo la conexión. Este paquete es una respuesta al paquete SYN y tiene un tamaño de 66 bytes.
- Paquete 15: SSH-2.0-Cliente - Indica la versión del cliente SSH, en este caso OpenSSH 7.7p1 Ubuntu-4ubuntu0.3. El tamaño del paquete es de 107 bytes.
- Paquete 17: SSH Key Exchange Init. Inicia el intercambio de claves entre el cliente y el servidor. Tiene un tamaño de 1426 bytes.
- Paquete 20: Key Exchange Init del servidor utilizando Elliptic Curve Diffie-Hellman. Tiene un tamaño de 114 Bytes.
- Paquete 23: New Keys - Este paquete confirma la instalación de las nuevas claves por parte del cliente. Tiene un tamaño de 82 bytes.
- Paquete 35: Encrypted Packet - Este paquete contiene datos cifrados, lo que indica que la comunicación entre cliente y servidor está ahora segura. El tamaño del paquete es de 214 bytes.

Al igual que para C1, se utiliza FATT para analizar el HASSH generado en la conexión. De esta manera se obtiene:

Como es posible ver en la figura 11, el HASSH generado para C2 es 06046964c022c6407d15a27b12a6a4fb

## 1.5 Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

### 1 DESARROLLO (PARTE 1)



```

..ad/Cripto/lab5/.venv/fatt ..ersidad/Cripto/lab5/.venv sudo wireshark +
.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1 (0.309s)
python3 fatt.py -r ../../Capturas/captura-c2.pcapng -p -j
172.17.0.2:58480 -> 172.17.0.3:22 [SSH] hassh=06046964c022c6407d15a27b12a6a4fb client=SSH-2.0-OpenSSH_7.7p1
Ubuntu-4ubuntu0.3
172.17.0.3:22 -> 172.17.0.2:58480 [SSH] hasshS=a984ff804585fabe3cd08f4b3849024a server=SSH-2.0-OpenSSH_9.0p1
Ubuntu-1ubuntu7.3

.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1
Try typing natural language instead of a command

```

Figura 11: HASSH generado para Handshake entre C2 y S1.

## 1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

Para la captura de tráfico obtenida, donde posible apreciar el “Handshake” generado al conectar el cliente C3 con el servidor S1, se obtiene:

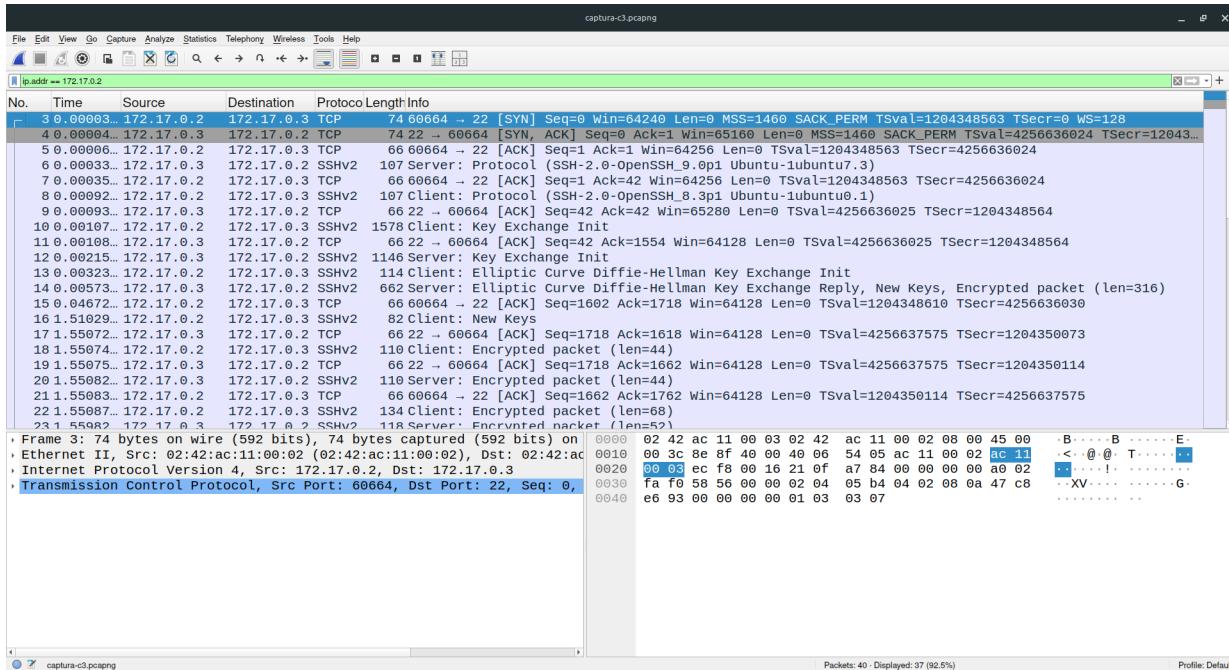
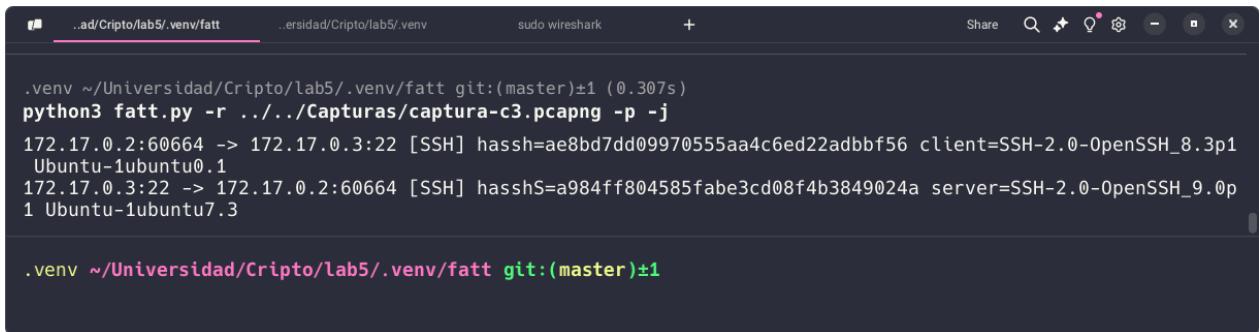


Figura 12: Captura de Wireshark para Handshake entre C3 y S1.

- Paquete 3: TCP [SYN] - Este paquete inicia la conexión TCP. El paquete tiene un tamaño de 74 bytes y establece los números de secuencia iniciales.
- Paquete 5: TCP [ACK] - Completa el handshake TCP, estableciendo la conexión. Este paquete es una respuesta al paquete SYN y tiene un tamaño de 66 bytes.

- Paquete 8: SSH-2.0-Cliente - Indica la versión del cliente SSH, en este caso OpenSSH 7.7p1 Ubuntu-4ubuntu0.3. El tamaño del paquete es de 107 bytes.
- Paquete 10: SSH Key Exchange Init. Inicia el intercambio de claves entre el cliente y el servidor. Tiene un tamaño de 1578 bytes.
- Paquete 13: Key Exchange Init del servidor utilizando Elliptic Curve Diffie-Hellman. Tiene un tamaño de 114 Bytes.
- Paquete 16: New Keys - Este paquete confirma la instalación de las nuevas claves por parte del cliente. Tiene un tamaño de 82 bytes.
- Paquete 18: Encrypted Packet - Este paquete contiene datos cifrados, lo que indica que la comunicación entre cliente y servidor está ahora segura. El tamaño del paquete es de 110 bytes.

Al igual que para C1 y C2, se utiliza FATT para analizar el HASSH generado en la conexión entre C3 y S1. De esta manera se obtiene:



```
.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1 (0.307s)
python3 fatt.py -r ../../Capturas/captura-c3.pcapng -p -j
172.17.0.2:60664 -> 172.17.0.3:22 [SSH] hassh=ae8bd7dd09970555aa4c6ed22adbbf56 client=SSH-2.0-OpenSSH_8.3p1
Ubuntu-1ubuntu0.1
172.17.0.3:22 -> 172.17.0.2:60664 [SSH] hasshS=a984ff804585fabe3cd08f4b3849024a server=SSH-2.0-OpenSSH_9.0p
1 Ubuntu-1ubuntu7.3

.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1
```

Figura 13: HASSH generado para Handshake entre C3 y S1.

Como es posible ver en la figura 13, el HASSH generado para C3 es ae8bd7dd09970555aa4c6ed22adbbf56

## 1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

Para la captura de tráfico obtenida, donde posible apreciar el “Handshake” generado al conectar el cliente C4 con el servidor S1, se obtiene:

## 1.6 Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

### 1 DESARROLLO (PARTE 1)

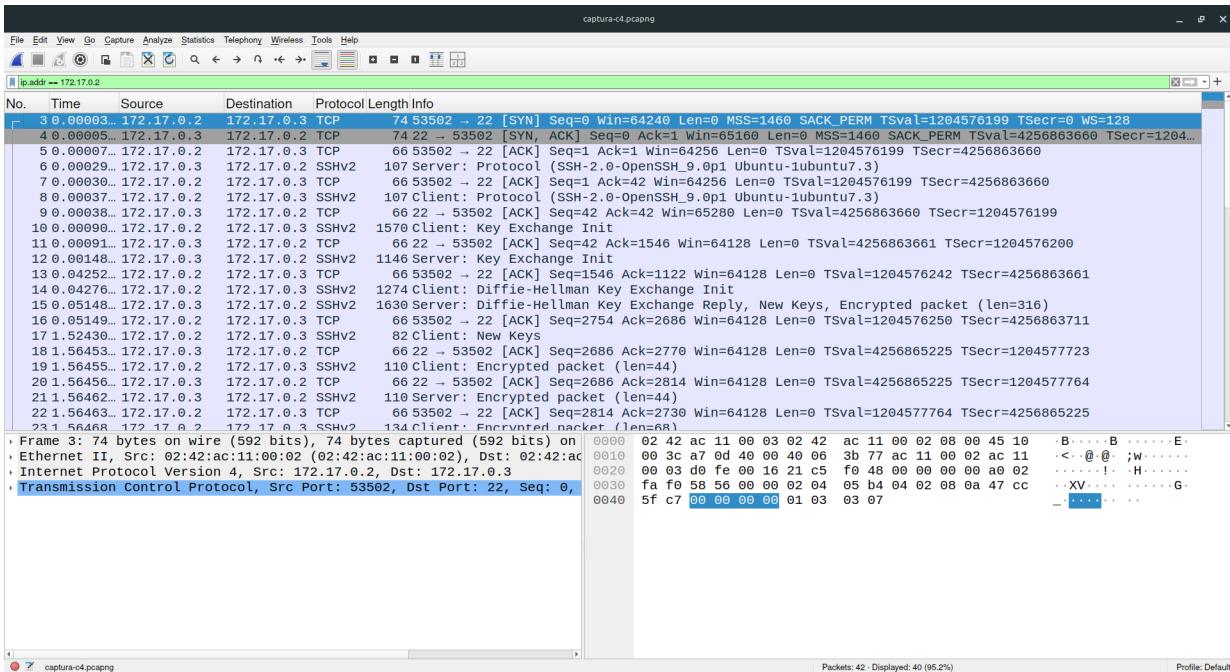
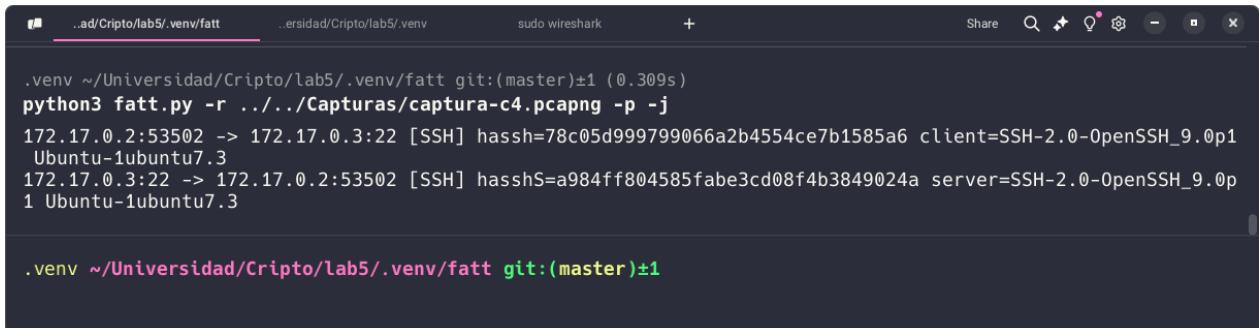


Figura 14: Captura de Wireshark para Handshake entre C4 y S1.

- Paquete 3: TCP [SYN] - Este paquete inicia la conexión TCP. El paquete tiene un tamaño de 74 bytes y establece los números de secuencia iniciales.
- Paquete 5: TCP [ACK] - Completa el handshake TCP, estableciendo la conexión. Este paquete es una respuesta al paquete SYN y tiene un tamaño de 66 bytes.
- Paquete 8: SSH-2.0-Cliente - Indica la versión del cliente SSH, en este caso OpenSSH 7.7p1 Ubuntu-4ubuntu0.3. El tamaño del paquete es de 107 bytes.
- Paquete 10: SSH Key Exchange Init. Inicia el intercambio de claves entre el cliente y el servidor. Tiene un tamaño de 1570 bytes.
- Paquete 14: Key Exchange Init del servidor utilizando Diffie-Hellman. Tiene un tamaño de 1274 Bytes.
- Paquete 17: New Keys - Este paquete confirma la instalación de las nuevas claves por parte del cliente. Tiene un tamaño de 82 bytes.
- Paquete 19: Encrypted Packet - Este paquete contiene datos cifrados, lo que indica que la comunicación entre cliente y servidor está ahora segura. El tamaño del paquete es de 110 bytes.

Al igual que para los otros 3 clientes, se utiliza FATT para analizar el HASSH generado en la conexión entre C4 y S1. De esta manera se obtiene:

## 1.7 Tipo de información contenida en cada uno de los paquetes generados en texto plano



The screenshot shows a terminal window running a script to analyze an SSH handshake. The command is:

```
.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1 (0.309s)
python3 fatt.py -r ../../Capturas/captura-c4.pcapng -p -j
```

The output shows two SSH packets exchanged between 172.17.0.2 and 172.17.0.3. The first packet is from the client (Ubuntu 14.04) to the server (Ubuntu 14.04), containing the identifier 'hash=78c05d999799066a2b4554ce7b1585a6'. The second packet is from the server to the client, containing the identifier 'hashS=a984ff804585fabe3cd08f4b3849024a'. The terminal prompt at the bottom is:

```
.venv ~/Universidad/Cripto/lab5/.venv/fatt git:(master)±1
```

Figura 15: HASSH generado para Handshake entre C4 y S1.

Como es posible ver en la figura 15, el HASSH generado para C4 es 78c05d999799066a2b4554ce7b1585a6

### 1.7. Tipo de información contenida en cada uno de los paquetes generados en texto plano

#### 1.7.1. C1

En base a lo obtenido de la captura realizada presente en la Figura 8, se destaca la presencia de los siguientes paquetes y la información de cada uno.

- **Paquetes de Configuración de Conexión (SYN, SYN-ACK, ACK):**
  - Paquetes de sincronización y reconocimiento entre el cliente C1 (172.17.0.2) y el servidor S1 (172.17.0.3) usando TCP para establecer la conexión. Estos incluyen paquetes con las etiquetas SYN y ACK, que son típicos del *handshake* inicial de TCP para sincronizar y establecer una sesión. Los paquetes relevantes son los números 1, 2 y 3.
- **Especificación de la Versión SSH:**
  - **Cliente:** Anuncia y utiliza la versión SSH-2.0-OpenSSH\_7.3p1 Ubuntu-14.04.1.
  - **Servidor:** Responde y confirma utilizando la versión SSH-2.0-OpenSSH\_9.0p1 Ubuntu-14.04.3, asegurando compatibilidad y seguridad.
- **Intercambio de Protocolo SSH:**
  - **Paquete de Protocolo:** Intercambio inicial de protocolo indicando la versión SSH utilizada por el cliente y el servidor. Los paquetes son los números 4 y 6.
  - **Intercambio de Claves:** Se transmiten paquetes relacionados con el intercambio de claves para la negociación de la encriptación, incluyendo la inicialización del intercambio de claves y la notificación de nuevas claves utilizadas en la sesión.

## 1.7 Tipo de información contenida en cada uno de los paquetes

Los paquetes importantes son los números 8 y 10 para el cliente y el servidor, respectivamente.

- **Paquetes Encriptados:** Varios paquetes etiquetados como “Client: Encrypted packet” y “Server: Encrypted packet”, representando datos encriptados enviados a través de la sesión SSH. Los paquetes correspondientes son el 16, 18, 19, etc., para el cliente y el servidor.

- **Confirmaciones de Recepción (ACK):**

- Se observan múltiples paquetes TCP con el flag ACK utilizados para confirmar la recepción de los paquetes anteriores. Son cruciales para el control de flujo y la integridad de la transmisión de datos en TCP. Paquetes relevantes incluyen los números 3, 5, 7, entre otros.

- **Detalles Técnicos:**

- Los paquetes incluyen información sobre longitud, secuencia y números de acuse de recibo, así como detalles sobre las ventanas de transmisión, todos importantes para el manejo eficiente de la red y el control de congestión.

### **1.7.2. C2**

En base a lo obtenido de la captura realizada presente en la Figura 10, se destaca la presencia de los siguientes paquetes y la información de cada uno.

- **Paquetes de Configuración de Conexión (SYN, SYN-ACK, ACK):**

- Paquetes de sincronización y reconocimiento entre el cliente C2 (172.17.0.2) y el servidor S1 (172.17.0.3) usando TCP para establecer la conexión. Estos incluyen paquetes con las etiquetas SYN y ACK, que son típicos del *handshake* inicial de TCP para sincronizar y establecer una sesión. Los paquetes relevantes son los números 10, 11 y 12.

- **Especificación de la Versión SSH:**

- **Cliente:** Anuncia y utiliza la versión SSH-2.0-OpenSSH\_7.7p1 Ubuntu-4ubuntu0.3.
- **Servidor:** Responde y confirma utilizando la versión SSH-2.0-OpenSSH\_9.0p1 Ubuntu-1ubuntu7.3, asegurando compatibilidad y seguridad.

- **Intercambio de Protocolo SSH:**

- **Paquete de Protocolo:** Intercambio inicial de protocolo indicando la versión SSH utilizada por el cliente y el servidor. Los paquetes son los números 13 y 15.

## 1.7 Tipo de información contenida en cada uno de los paquetes ~~DESARROLLO o PARTE 1~~

---

- **Intercambio de Claves:** Se transmiten paquetes relacionados con el intercambio de claves para la negociación de la encriptación, incluyendo la inicialización del intercambio de claves y la notificación de nuevas claves utilizadas en la sesión. Los paquetes importantes son los números 17 y 19 para el cliente y el servidor, respectivamente.
- **Paquetes Encriptados:** Varios paquetes etiquetados como “Client: Encrypted packet” y “Server: Encrypted packet”, representando datos encriptados enviados a través de la sesión SSH. Los paquetes correspondientes son el 25, 27, 29, etc., para el cliente y el servidor.
- **Confirmaciones de Recepción (ACK):**
  - Se observan múltiples paquetes TCP con el flag ACK utilizados para confirmar la recepción de los paquetes anteriores. Son cruciales para el control de flujo y la integridad de la transmisión de datos en TCP. Paquetes relevantes incluyen los números 12, 14, 16, entre otros.
- **Detalles Técnicos:**
  - Los paquetes incluyen información sobre longitud, secuencia y números de acuse de recibo, así como detalles sobre las ventanas de transmisión, todos importantes para el manejo eficiente de la red y el control de congestión.

### 1.7.3. C3

En base a lo obtenido de la captura realizada presente en la Figura 12, se destaca la presencia de los siguientes paquetes y la información de cada uno.

- **Paquetes de Configuración de Conexión (SYN, SYN-ACK, ACK):**
  - Se utilizan paquetes TCP con las etiquetas SYN y ACK para establecer y confirmar la conexión entre el cliente C3 (172.17.0.2) y el servidor S1 (172.17.0.3). Estos paquetes son típicos del *handshake* inicial de TCP para sincronizar y establecer una sesión. Los paquetes relevantes son los números 3 y 4.
- **Especificación de la Versión SSH:**
  - **Cliente:** Anuncia y utiliza la versión SSH-2.0-OpenSSH\_8.3p1 Ubuntu-1ubuntu0.1.
  - **Servidor:** Responde y confirma utilizando la versión SSH-2.0-OpenSSH\_9.0p1 Ubuntu-1ubuntu7.3.
- **Intercambio de Protocolo SSH:**
  - **Paquete de Protocolo:** Intercambio inicial de protocolo indicando la versión SSH utilizada por el cliente y el servidor. Los paquetes son los números 6 y 8.

## **1.7 Tipo de información contenida en cada uno de los paquetes**

---

- **Intercambio de Claves:** Se transmiten paquetes relacionados con el intercambio de claves para la negociación de la encriptación, incluyendo la inicialización del intercambio de claves y la notificación de nuevas claves utilizadas en la sesión. Los paquetes son los números 10 y 12 para el cliente y el servidor, respectivamente.
- **Paquetes Encriptados:** Varios paquetes etiquetados como “Client: Encrypted packet” y “Server: Encrypted packet”, representando datos encriptados enviados a través de la sesión SSH. Los paquetes correspondientes son el 18, 20, 22, etc., para el cliente y el servidor.
- **Confirmaciones de Recepción (ACK):**
  - Se observan múltiples paquetes TCP con el flag ACK utilizados para confirmar la recepción de los paquetes anteriores. Son necesarios para el control de flujo y la integridad de la transmisión de datos en TCP. Paquetes relevantes incluyen los números 5, 7, 9, entre otros.
- **Detalles Técnicos:**
  - Cada paquete incluye información sobre el tamaño (Length), secuencia y acuse de recibo (Seq y Ack), así como las ventanas de transmisión (Win). Cada uno de ellos es importante para el manejo eficiente de la red y el control de congestión.

### **1.7.4. C4/S1**

En base a lo obtenido de la captura realizada presente en la Figura 14, se destaca la presencia de los siguientes paquetes y la información de cada uno.

- **Paquetes de Configuración de Conexión (SYN, SYN-ACK, ACK):**
  - Paquetes de sincronización y reconocimiento entre el cliente C3 (172.17.0.2) y el servidor S1 (172.17.0.3) usando TCP para establecer la conexión.
  - Intercambio inicial que prepara la sesión TCP para la comunicación segura a través de SSH.
- **Especificación de la Versión SSH:**
  - **Cliente:** Anuncia y utiliza la versión SSH-2.0-OpenSSH\_9.0p1 Ubuntu-1ubuntu7.3 para la conexión.
  - **Servidor:** Responde y confirma utilizando la misma versión SSH-2.0-OpenSSH\_9.0p1 Ubuntu-1ubuntu7.3, asegurando compatibilidad y seguridad.
- **Intercambio de Protocolo SSH:**
  - **Intercambio de Claves:** Se observan paquetes de inicio del intercambio de claves, esenciales para establecer una sesión cifrada.

- **Paquetes Encriptados:** Varios paquetes etiquetados como “Client: Encrypted packet” y “Server: Encrypted packet”, representando datos encriptados enviados a través de la sesión SSH, lo cual es un comportamiento esperado en cualquier conexión segura SSH para proteger los datos transmitidos. Corresponde al paquete 18, 22, 25, etc.
- **Confirmaciones de Recepción (ACK):** Estos paquetes son fundamentales para confirmar la recepción de datos en la sesión TCP, asegurando una comunicación fluida y sin errores.
- **Detalles Técnicos:**
  - Los paquetes incluyen información sobre longitud, secuencia y números de acuse de recibo, así como detalles sobre las ventanas de transmisión, todos importantes para el manejo eficiente de la red y el control de congestión.

## 1.8. Diferencia entre C1 y C2

Al comparar las capturas de paquetes de C1 y C2, se observan varias diferencias en como están constituidos los paquetes y la información que estos traen.

- En C1 el cliente usa SSH-2.0-OpenSSH\_7.3p1 y el servidor usa SSH-2.0-OpenSSH\_9.0p1, mientras que en C2, el cliente usa SSH-2.0-OpenSSH\_7.7p1 y el servidor usa SSH-2.0-OpenSSH\_9.0p1. Estas diferencias en las versiones del cliente pueden afectar las características de seguridad disponibles y la compatibilidad.
- La longitud de los paquetes varía entre las dos capturas. Por ejemplo, las longitudes de los paquetes de Key Exchange Init son 1498 bytes en C1 y 1426 bytes en C2. Esto puede deberse a diferencias en la configuración de las sesiones SSH o en los algoritmos de cifrado utilizados.
- Que los tiempos entre paquetes también varíen, aunque de forma mínima, se puede deber a la carga de la red, la configuración del servidor o las propiedades del tráfico de red en dichos momentos.

## 1.9. Diferencia entre C2 y C3

- En C1 el cliente usa la versión SSH-2.0-OpenSSH\_7.7p1 y el servidor SSH-2.0-OpenSSH\_9.0p1, mientras que en C2, el cliente usa SSH-2.0-OpenSSH\_8.3p1 y la del servidor también es SSH-2.0-OpenSSH\_9.0p1. Habiendo una diferencia entre los clientes, al igual que en el caso comparativo entre C1 y C2.
- La longitud de los paquetes varía entre las dos capturas. Por ejemplo, las longitudes de los paquetes de Key Exchange Init son 1426 bytes para C2, mientras que en C3 es de 1578 bytes. Esto puede deberse a diferencias en la configuración de las sesiones SSH o en los algoritmos de cifrado utilizados.

- AL igual que en la comparación anterior, que los tiempos entre paquetes también varíen, aunque de forma mínima, se puede deber a la carga de la red, la configuración del servidor o las propiedades del tráfico de red en dichos momentos.

## 1.10. Diferencia entre C3 y C4

- Ambas capturas utilizan la misma versión del servidor SSH (SSH-2.0-OpenSSH\_9.0p1), pero el cliente en C4 también usa esta versión, a diferencia de C3, donde usa SSH-2.0-OpenSSH\_8.3p1. Esto indica que en C4, tanto el cliente como el servidor están sincronizados en la versión más reciente de OpenSSH.
- La longitud de los paquetes varía entre las dos capturas. Por ejemplo, para C3 la longitud del paquete Key Exchange Init son 1578 bytes, mientras que en C4 es de 1570 bytes. Aunque las longitudes son muy similares, existe una pequeña diferencia que puede estar relacionada con variaciones en los datos o headers adicionales.
- AL igual que en las comparaciones anteriores, que los tiempos entre paquetes también varíen, aunque de forma mínima, se puede deber a la carga de la red, la configuración del servidor o las propiedades del tráfico de red en dichos momentos.

## 2. Desarrollo (Parte 2)

### 2.1. Identificación del cliente SSH con versión “?”

Para poder identificar el cliente buscado, es necesario considerar las principales diferencias entre los clientes utilizados, como las mencionadas anteriormente.

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 16: Tráfico generado del informante

captura-c3.pcapng						
No.	Time	Source	Destination	Protocol	Length	Info
3 0.00003..	172.17.0.2	172.17.0.3	TCP	74	60664 → 22	[SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TStamp=1204348563 TSecr=1204348563
4 0.00004..	172.17.0.3	172.17.0.2	TCP	74	22 → 60664	[SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TStamp=1204348563 TSecr=1204348563
5 0.00006..	172.17.0.2	172.17.0.3	TCP	66	60664 → 22	[ACK] Seq=1 Ack=1 Win=64256 Len=0 TStamp=1204348563 TSecr=1204348563
6 0.000033..	172.17.0.3	172.17.0.2	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)	
7 0.000035..	172.17.0.2	172.17.0.3	TCP	66	60664 → 22	[ACK] Seq=1 Ack=42 Win=64256 Len=0 TStamp=1204348563 TSecr=1204348563
8 0.000092..	172.17.0.2	172.17.0.3	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)	
9 0.000093..	172.17.0.2	172.17.0.3	TCP	66	22 → 60664	[ACK] Seq=42 Ack=42 Win=65280 Len=0 TStamp=1204348564 TSecr=1204348564
10 0.00107..	172.17.0.2	172.17.0.3	SSHv2	1578	Client: Key Exchange Init	
11 0.00108..	172.17.0.3	172.17.0.2	TCP	66	22 → 60664	[ACK] Seq=42 Ack=1554 Win=64128 Len=0 TStamp=1204348564 TSecr=1204348564

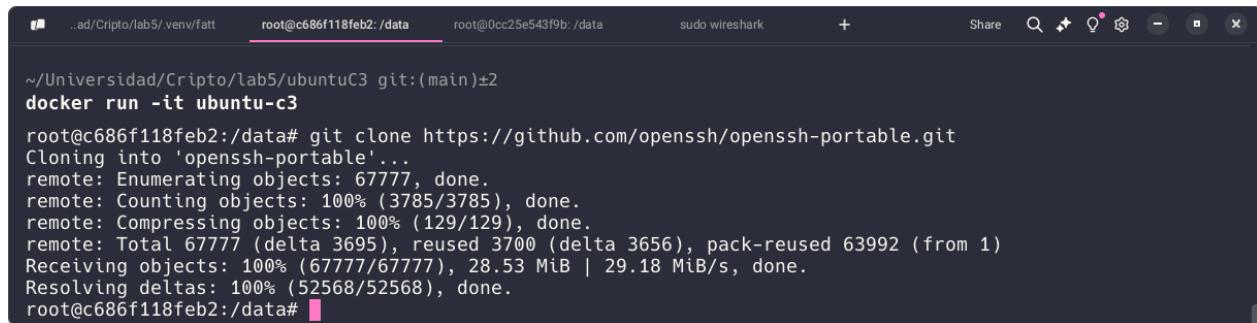
Figura 17: KEI de C3

En base a lo mencionado, se puede apreciar que el tamaño del KEI perteneciente al cliente buscado (Figura 16), es idéntico al presente en la captura de C3 (Figura 17),

y como esta era una de las principales diferencias entre las capturas realizadas, se asume que este es el cliente con versión “?” con el cual el informante realizo la captura de tráfico.

## 2.2. Replicación de tráfico al servidor (paso por paso)

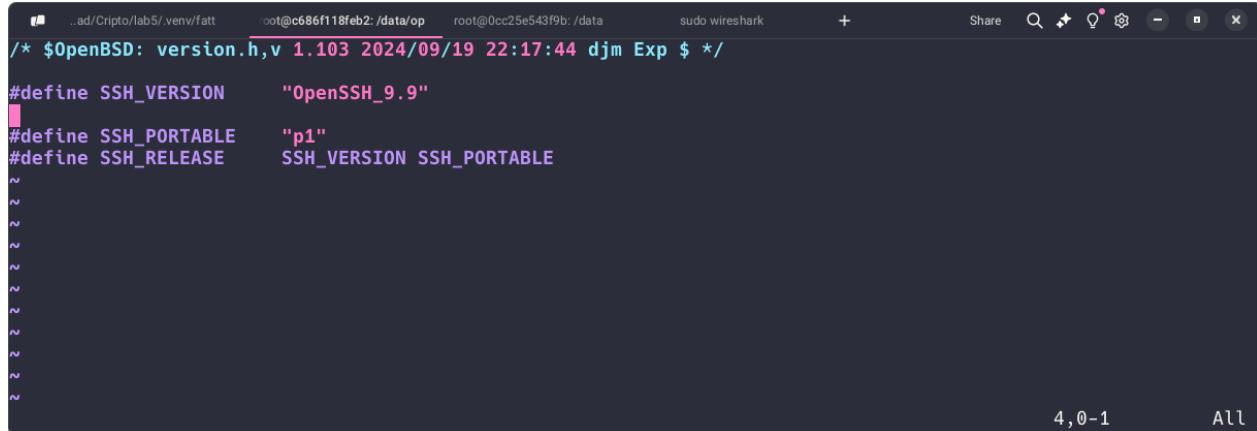
Para replicar el tráfico mostrado en la Figura 16, se hace uso de la versión portable de SSH, la cual es obtenida desde su repositorio. Este repositorio es clonado en la carpeta “data”, la cual es la utilizada para trabajar dentro de la imagen del cliente C3.



```
~/Universidad/Cripto/lab5/ubuntuC3 git:(main)±2
docker run -it ubuntu-c3
root@c686f118feb2:/data# git clone https://github.com/openssh/openssh-portable.git
Cloning into 'openssh-portable'...
remote: Enumerating objects: 67777, done.
remote: Counting objects: 100% (3785/3785), done.
remote: Compressing objects: 100% (129/129), done.
remote: Total 67777 (delta 3695), reused 3700 (delta 3656), pack-reused 63992 (from 1)
Receiving objects: 100% (67777/67777), 28.53 MiB | 29.18 MiB/s, done.
Resolving deltas: 100% (52568/52568), done.
root@c686f118feb2:/data#
```

Figura 18: Clonación del repositorio

Luego, en la carpeta contenedora del repositorio se debe modificar el archivo “version.h” el cual almacena información respecto a la versión utilizada por SSh. Para esto se utiliza VIM (editor de texto), escribiendo en la terminal `vim version.h`.



```
/* $OpenBSD: version.h,v 1.103 2024/09/19 22:17:44 djm Exp $ */
#define SSH_VERSION      "OpenSSH_9.9"
#define SSH_PORTABLE     "p1"
#define SSH_RELEASE      SSH_VERSION SSH_PORTABLE
~
```

Figura 19: Archivo version.h original

Figura 20: Archivo version.h modificado

Como es posible apreciar en la Figura 19, se la versión empleada de SSH es “OpenSSh\_9.9”, por lo que esta es cambiada por “OpenSSh\_?”. Luego se aplican los cambios con la combinación de teclas :wq .

Luego, se deben ejecutar las siguientes combinaciones de comandos en la terminal para aplicar los cambios:

- **autoreconf** Este comando genera los scripts de configuración necesarios a partir de los archivos configure.ac y Makefile.am.
  - **./configure** es un script que configura el paquete de software para que se compile en el sistema. Detecta las características del sistema y ajustan los archivos de configuración.
  - **make** compila el paquete de software, construyendo los ejecutables y otros archivos necesarios a partir del código fuente.
  - **make install**, como dice su nombre, se encarga de instalar los archivos compilados en el sistema, colocándolos en los directorios adecuados.

Finalmente, se vuelve a realizar una conexión entre C3 y S1, sobre la cual se realiza una captura en Wireshark, obteniendo así los cambios reflejados en los paquetes.

4	8.98519...	172.17.0.2	172.17.0.3	TCP	74	34244 → 22	[SYN]	Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSeq=1
5	8.98520...	172.17.0.3	172.17.0.2	TCP	74	22 → 34244	[SYN, ACK]	Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSeq=2
6	8.98521...	172.17.0.2	172.17.0.3	TCP	66	34244 → 22	[ACK]	Seq=1 Ack=1 Win=64256 Len=0 TSeq=1 MSS=1169583808
7	8.98539...	172.17.0.2	172.17.0.3	SSHv2	85	Client: Protocol (SSH-2.0-OpenSSH_?)		
8	8.98540...	172.17.0.3	172.17.0.2	TCP	66	22 → 34244	[ACK]	Seq=1 Ack=20 Win=65152 Len=0 TSeq=1 MSS=134887504
9	8.98592...	172.17.0.3	172.17.0.2	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0.0 Ubuntu-1ubuntu7.3)		

Figura 21: Archivo version.h modificado

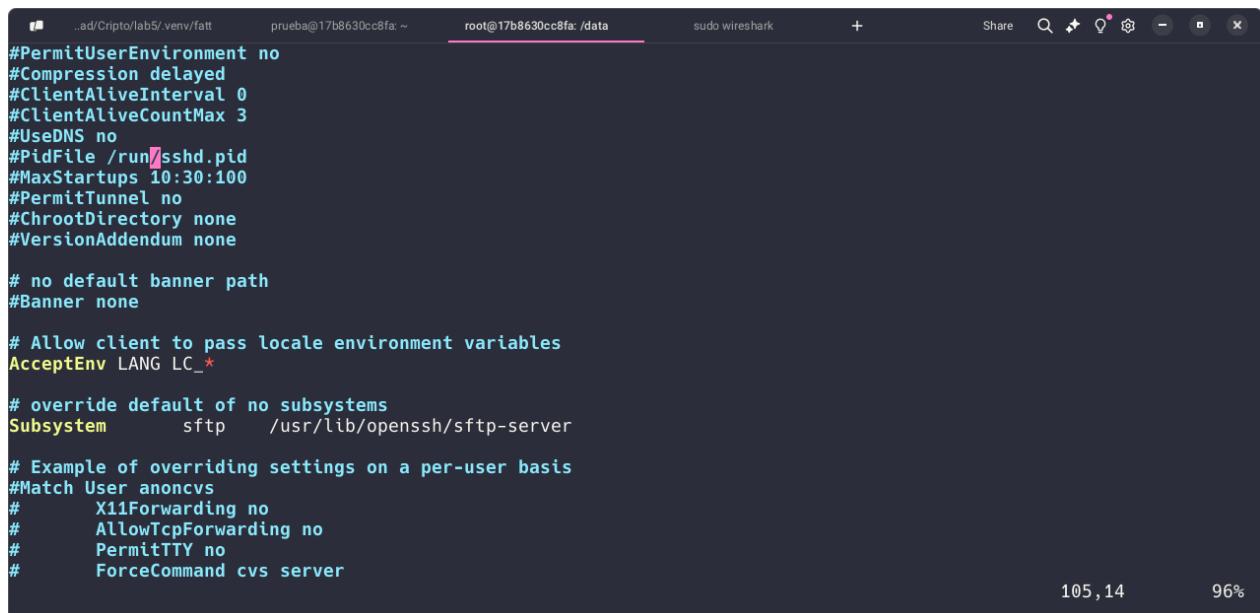
Como se aprecia en la Figura 21, se observa que el paquete numero 7, muestra la versión modificada para OpenSSH.

## 3. Desarrollo (Parte 3)

### 3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso)

Para lograr obtener un KEI con un tamaño menor a 300 bytes, se pueden hacer modificaciones en archivos de configuración de SSH. Para esto primero se debe ingresar al archivo `sshd_config` mediante el comando en consola `vim /etc/ssh/sshd_config`.

Dentro del archivo se pueden modificar algunas configuraciones de algoritmos empleados.



```
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS no
#PidFile /run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none

# no default banner path
#Banner none

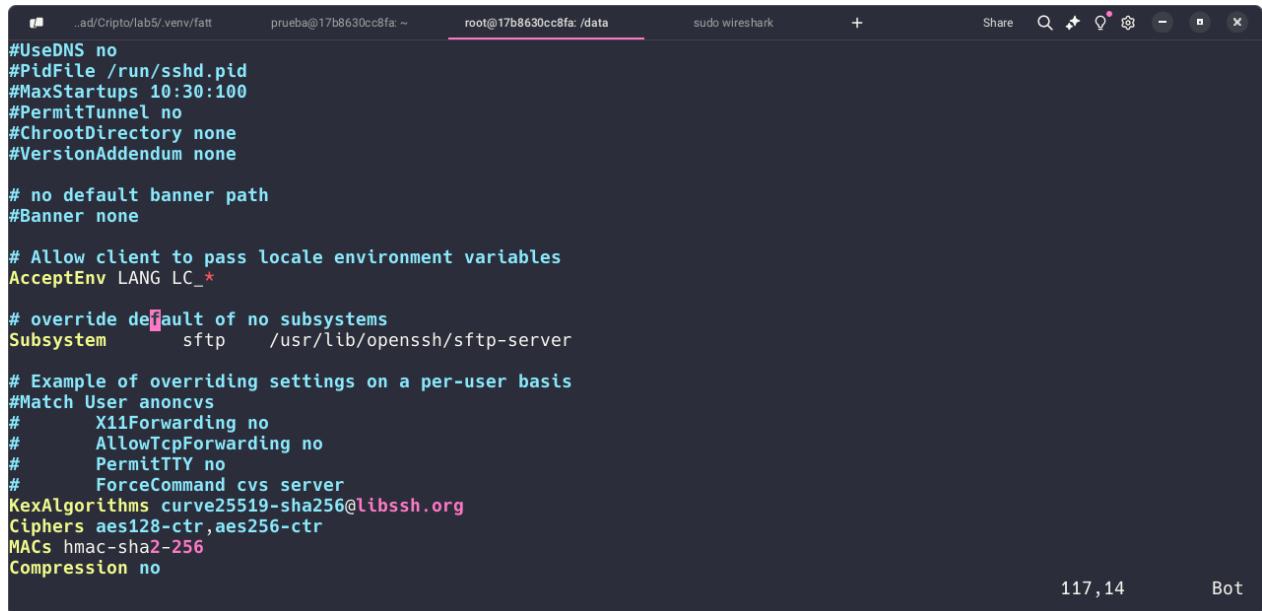
# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

# override default of no subsystems
Subsystem      sftp    /usr/lib/openssh/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#      X11Forwarding no
#      AllowTcpForwarding no
#      PermitTTY no
#      ForceCommand cvs server
```

Figura 22: Archivo `sshd_config` original.

### 3.1 Replicación del KEI con tamaño menor a 300 bytes (paso DESARROLLO (PARTE 3))



The screenshot shows a terminal window with the following content:

```
#UseDNS no
#PidFile /run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none

# no default banner path
#Banner none

# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

# override default of no subsystems
Subsystem      sftp    /usr/lib/openssh/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server
KexAlgorithms curve25519-sha256@libssh.org
Ciphers aes128-ctr,aes256-ctr
MACs hmac-sha2-256
Compression no
```

The terminal window has a dark background and light-colored text. It includes standard Linux terminal elements like tabs, a status bar at the bottom, and a title bar.

Figura 23: Archivo sshd\_config modificado.

En la Figura 23, se pueden apreciar los cambios respectivos, dentro de los cuales encontramos:

- **KexAlgorithms:** Especifica los algoritmos de intercambio de claves permitidos. (curve25519-sha256@libssh.org)
- **Ciphers:** Especifica los cifrados permitidos. Se puede limitar a dos de los más seguros y eficientes en cuanto al tamaño del mensaje (aes128-ctr,aes256-ctr).
- **MACs:** Especifica los algoritmos MAC permitidos. Se elige hmac-sha2-256 el cuál es seguro pero no agrega mucho “overhead”.
- **Compression:** desactiva la compresión para reducir la complejidad.

Luego al realizar reiniciar el servicio de S1 mediante el uso del comando `service ssh restart`, se procede a hacer una captura de paquetes en una nueva conexión entre uno de los clientes (C2 para este caso) y el servidor.

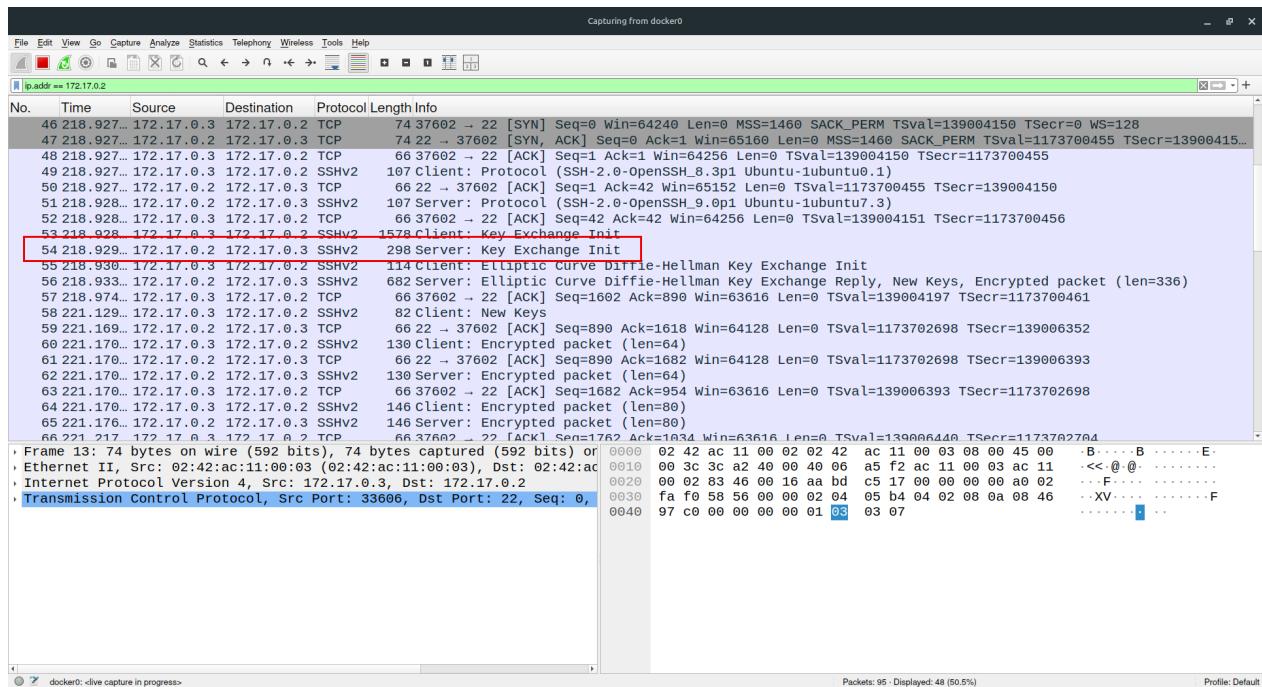


Figura 24: Nueva captura con KEI de servidor menor a 300 bytes.

Como es posible ver en la Figura 24, el paquete respectivo a la KEI del servidor, se puede apreciar que posee un largo de 298 bytes, cumpliendo con solicitado.

## 4. Desarrollo (Parte 4)

### 4.1. Explicación OpenSSH en general

OpenSSH es un conjunto herramientas que pueden ser utilizadas para establecer comunicaciones seguras en redes no seguras mediante el uso de protocolos criptográficos. Esta incluye el cliente ssh para conectar a servidores que ejecutan el demonio sshd, herramientas scp y sftp para la transferencia segura de archivos, y ssh-keygen para la gestión de claves de autenticación.

OpenSSH utiliza cifrado para proteger la transmisión de datos y autenticación basada en claves o contraseñas para verificar la identidad del usuario, asegurando así la integridad y confidencialidad de la información transferida. Además, puede crear vías seguras para “montar” sistemas de archivos remotos, con el fin de poder administrar de manera remota un sistemas y realizar la transferencia de datos en entornos inseguros.

## 4.2. Capas de Seguridad en OpenSSH

OpenSSH utiliza múltiples capas que trabajan en conjunto para proteger la comunicación entre el cliente y el servidor. En estas, podemos encontrar:

### 1. Autenticación de Usuario

- **Autenticación basada en contraseña:** Es el método más básico donde los usuarios ingresan una contraseña para acceder a un servidor remoto. Las contraseñas nunca se envían en texto plano y están protegidas durante la transmisión mediante cifrado.
- **Autenticación basada en llaves:** Se considera más segura que la autenticación por contraseña. Involucra un par de llaves, una pública y una privada. La clave privada está almacenada de forma segura por el usuario y la pública se coloca en el servidor. Durante la autenticación, el servidor desafía al cliente a demostrar que posee la clave privada correspondiente a la pública sin transmitir la clave en sí.
- **Otras formas de autenticación:** Incluyen el uso de certificados, métodos de autenticación de un solo uso, y autenticación de múltiples factores.

### 2. Cifrado de la Sesión

- **Negociación de Protocolo:** Al inicio de una sesión SSH, el cliente y el servidor acuerdan detalles sobre la versión del protocolo y los algoritmos de cifrado que se usarán.
- **Intercambio de Claves:** Utiliza algoritmos seguros para el intercambio de claves (como Diffie-Hellman) para establecer una clave de sesión secreta. Esta clave se utiliza para cifrar toda la comunicación de la sesión.
- **Cifrado Simétrico:** Una vez establecida, la clave de sesión se utiliza para cifrar los datos transferidos utilizando algoritmos simétricos como AES.

### 3. Integridad de los Datos

- **Algoritmos MAC (Message Authentication Code):** Despues de cifrar los datos, se agrega un código de autenticación de mensaje para asegurar la integridad de los datos. Esto ayuda a detectar cualquier alteración de los datos durante la transmisión. Este es uno de los parametros modificados durante la relaización de la actividad.

### 4. Configuración del Servidor y Políticas de Seguridad

- **Configuración de sshd\_config:** Los administradores pueden ajustar la seguridad del servidor SSH configurando “sshd\_config” para limitar o permitir ciertos algoritmos, configurar tiempos de espera, limitar los intentos de conexión, requerir autenticación de múltiples factores, etc. Esto también fue realizado en la actividad recién descrita.

## 5. Seguridad en el lado del cliente

- **ssh-agent:** Un agente de seguridad que mantiene las claves privadas cargadas en memoria, listas para ser usadas sin exponerlas directamente en el sistema de archivos. Esto reduce el riesgo de robo de claves.
- **Passphrase de la clave privada:** Añade una capa adicional de seguridad al requerir que se ingrese una frase de contraseña para desbloquear el uso de la clave privada.

### 4.3. Identificación de que protocolos no se cumplen

Si bien se puede apreciar el como se aplican distintos protocolos de seguridad, al realizar un análisis sobre el cumplimiento de los principios de seguridad de la información, es posible afirmar que el principio de no repudio en el contexto de OpenSSH es limitado porque, aunque el protocolo proporciona los mecanismos necesarios de autenticación y cifrado, no incluye una firma digital completa de todos los datos intercambiados entre las partes comunicantes (cliente y servidor). Esto puede significar que, aunque los usuarios pueden estar seguros de la confidencialidad e integridad de sus datos durante la transmisión, OpenSSH por sí solo no garantiza la capacidad de probar su participación en todas las acciones o transacciones. La falta de registros detallados y firmas digitales para cada acción ejecutada bajo la sesión SSH dificulta la atribución de acciones a individuos específicos, lo cual es de suma importancia para el no repudio. Este principio requiere que cada operación pueda ser vinculada a una parte responsable, característica que OpenSSH no proporciona de manera nativa.

## Conclusiones y comentarios

En este laboratorio, se ha logrado realizar un análisis completo sobre el tráfico SSH generado por diferentes versiones de clientes SSH, junto con un servidor OpenSSH. Para esto se utilizaron una variedad de configuraciones controladas en contenedores de Docker para simular y replicar diversas condiciones de red. Se emplearon herramientas como Wireshark para un correcto análisis del tráfico generado en la red por las conexiones simuladas, permitiendo la identificación precisa de las versiones de los clientes SSH y la diferenciación entre dichos paquetes. También se hizo uso de FATT para obtener los HASSH de las capturas. Además, se logró un ajuste en la configuración del servidor SSH para reducir el tamaño del paquete Key Exchange Init a menos de 300 bytes, demostrando la flexibilidad y la capacidad de configuración de OpenSSH.

Los resultados muestran la importancia de comprender las configuraciones y opciones disponibles en los protocolos de seguridad para garantizar la integridad y la seguridad de las comunicaciones. Este estudio también muestra la relevancia de la autenticación y el cifrado en la protección de datos sensibles transmitidos a través de redes potencialmente inseguras, asegurando la confidencialidad y la integridad de la información. Sin embargo, se logra identificar una limitación en el principio de no repudio, ya que OpenSSH no ofrece de forma nativa

un mecanismo que permita la firma completa de todos los datos intercambiados, lo que podría dificultar la atribución de acciones a individuos específicos sin registros adicionales o medidas de seguridad complementarias. Este laboratorio no solo ha fortalecido la comprensión sobre las capas de seguridad en comunicaciones cifradas, sino también ha puesto en evidencia la necesidad de continuar mejorando y adaptando las herramientas de seguridad para enfrentar los distintos desafíos que surgen actualmente en la protección de datos.

## Anexo

- Repositorio FATT.
- Repositorio openssh-portable.
- SSH: qué es y cómo funciona este protocolo.
- Repositorio Laboratorio.