

# Informe Laboratorio 5

**Sección x**

Alumno x

e-mail: alumno.contacto@mail\_udp.cl

Noviembre de 2025

# Índice

<b>Descripción de actividades</b>	<b>4</b>
<b>1. Desarrollo (Parte 1)</b>	<b>6</b>
1.1. Códigos de cada Dockerfile . . . . .	6
1.1.1. C1 . . . . .	7
1.1.2. C2 . . . . .	7
1.1.3. C3 . . . . .	8
1.1.4. C4/S1 . . . . .	8
1.2. Creación de las credenciales para S1 . . . . .	9
1.2.1. Creación de la red privada . . . . .	9
1.2.2. Verificación de credenciales de acceso . . . . .	9
1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .	10
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) . . . . .	12
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 . . . . .	15
1.7.1. C1 (Ubuntu 16.10) . . . . .	15
1.7.2. C2 (Ubuntu 18.10) . . . . .	15
1.7.3. C3 (Ubuntu 20.10) . . . . .	16
1.7.4. C4/S1 (Ubuntu 22.10) . . . . .	16
1.8. Diferencia entre C1 y C2 . . . . .	16
1.9. Diferencia entre C2 y C3 . . . . .	16
1.10. Diferencia entre C3 y C4 . . . . .	16
<b>2. Desarrollo (Parte 2)</b>	<b>17</b>
2.1. Identificación del cliente SSH con versión “?” . . . . .	17
2.2. Replicación de tráfico al servidor (paso por paso) . . . . .	17
<b>3. Desarrollo (Parte 3)</b>	<b>20</b>
3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso) . . . . .	20
<b>4. Desarrollo (Parte 4)</b>	<b>23</b>
4.1. Explicación OpenSSH en general . . . . .	23
4.2. Capas de Seguridad en OpenSSH . . . . .	25
4.2.1. Confidencialidad . . . . .	25
4.2.2. Integridad . . . . .	25

4.2.3. Autenticidad . . . . .	25
4.2.4. Disponibilidad y No Repudio . . . . .	25
4.3. Identificación de que protocolos no se cumplen . . . . .	26
4.3.1. Confidencialidad Parcial (Metadatos) . . . . .	26
4.3.2. No Repudio (Limitado) . . . . .	26
<b>5. Conclusiones y comentarios</b>	<b>26</b>

## Descripción de actividades

Para este último laboratorio, se solicita trabajar con Docker y el protocolo SSH, a fin de poder entender el concepto de criptografía asimétrica y firmas digitales.

Para lo anterior deberá:

- Crear 4 contenedores en Docker por medio de un DockerFile, 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, 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. 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=3
TCP	74 42398 → 22 [SYN] Seq=0 Win=65535
TCP	74 22 → 42398 [SYN, ACK] Seq=1 Ack=2
TCP	66 42398 → 22 [ACK] Seq=1 Ack=2
SSHv2	87 Client: Protocol (SSH-2.0-OpenSSH_8.2)
TCP	66 22 → 42398 [ACK] Seq=1 Ack=1
SSHv2	107 Server: Protocol (SSH-2.0-OpenSSH_8.2)
TCP	66 42398 → 22 [ACK] Seq=22 Ack=23
SSHv2	1570 Client: Key Exchange Init
TCP	66 22 → 42398 [ACK] Seq=42 Ack=43
SSHv2	298 Server: Key Exchange Init
TCP	66 42398 → 22 [ACK] Seq=1526 Ack=1527

Figura 2: Captura del Key Exchange

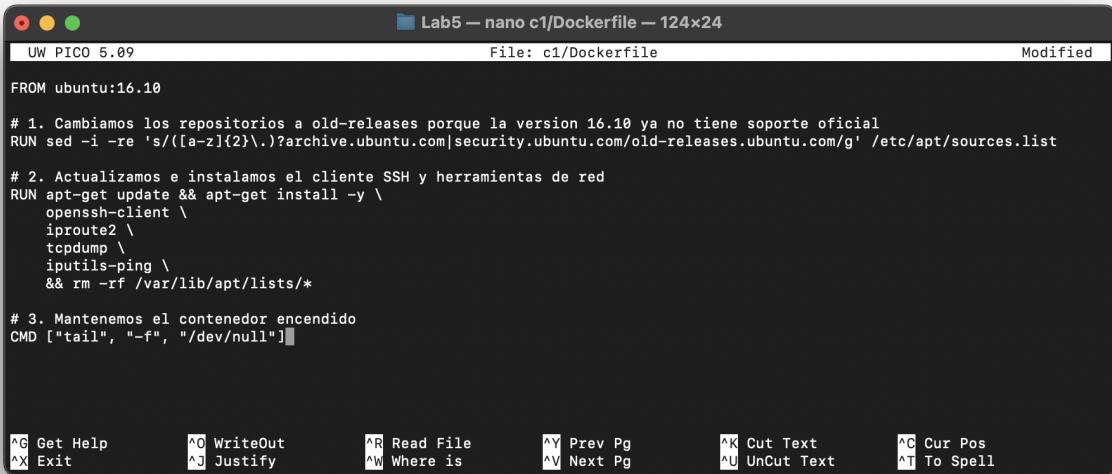
4. Tomando en cuenta lo aprendido en este laboratorio, así como en los anteriores, explique el protocolo OpenSSH y las diferentes capas de seguridad que son parte del protocolo para garantizar los principios de seguridad de la información, integridad, confidencialidad, disponibilidad, autenticidad y 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

Se requiere la creación de cuatro contenedores Docker utilizando imágenes base de Ubuntu con versiones específicas (16.10, 18.10, 20.10 y 22.10). Dado que estas distribuciones se encuentran fuera del ciclo de soporte oficial (EOL - End of Life), la instalación de paquetes de OpenSSH se posibilita mediante la modificación del archivo sources.list, este ajuste se ejecuta utilizando el comando sed para redireccionar los repositorios de descarga a old-releases.ubuntu.com, garantizando así la instalación exitosa del software cliente y servidor requerido para la experimentación, los comandos usando docker se muestran a continuacion.

## 1.1.1. C1



```

UW PICO 5.09                               File: c1/Dockerfile                         Modified
FROM ubuntu:16.10

# 1. Cambiamos los repositorios a old-releases porque la version 16.10 ya no tiene soporte oficial
RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

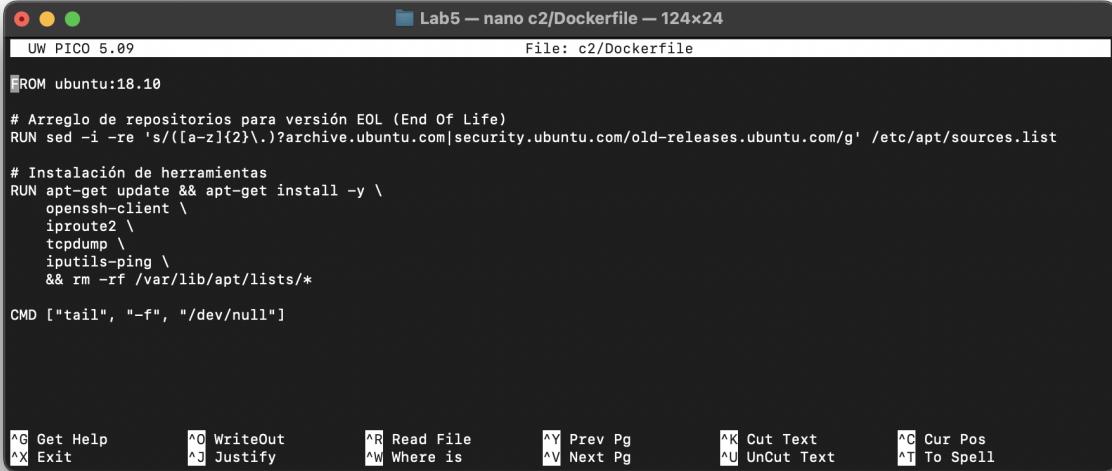
# 2. Actualizamos e instalamos el cliente SSH y herramientas de red
RUN apt-get update && apt-get install -y \
    openssh-client \
    iproute2 \
    tcpdump \
    iputils-ping \
    && rm -rf /var/lib/apt/lists/*
# 3. Mantenemos el contenedor encendido
CMD ["tail", "-f", "/dev/null"]

```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg ^K Cut Text ^C Cur Pos  
 ^X Exit ^J Justify ^W Where is ^V Next Pg ^U UnCut Text ^T To Spell

Figura 3: comandos zsh para el cliente 1

## 1.1.2. C2



```

UW PICO 5.09                               File: c2/Dockerfile                         Modified
FROM ubuntu:18.10

# Arreglo de repositorios para versión EOL (End Of Life)
RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

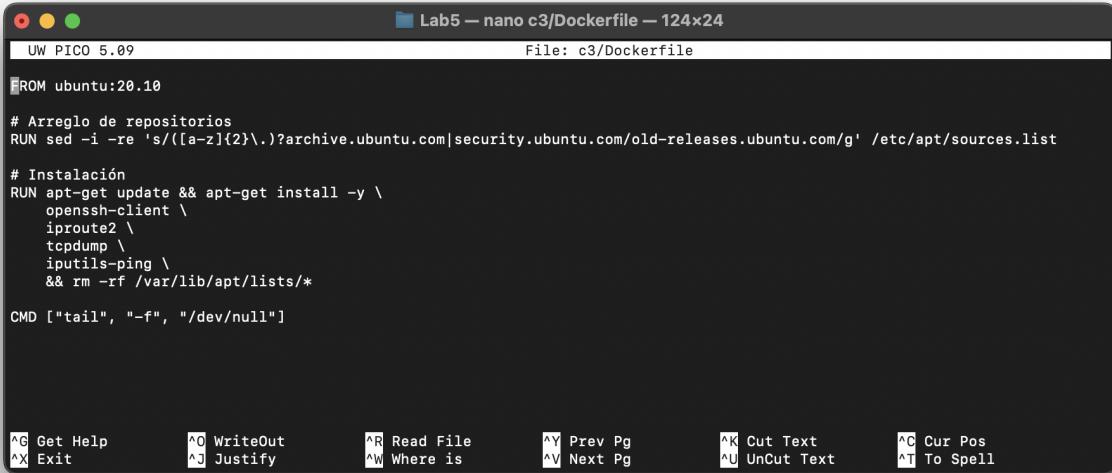
# Instalación de herramientas
RUN apt-get update && apt-get install -y \
    openssh-client \
    iproute2 \
    tcpdump \
    iputils-ping \
    && rm -rf /var/lib/apt/lists/*
CMD ["tail", "-f", "/dev/null"]

```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg ^K Cut Text ^C Cur Pos  
 ^X Exit ^J Justify ^W Where is ^V Next Pg ^U UnCut Text ^T To Spell

Figura 4: comandos zsh para el cliente 2

### 1.1.3. C3



```

FROM ubuntu:20.10

# Arreglo de repositorios
RUN sed -i -re 's/([a-z]{2}.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

# Instalación
RUN apt-get update && apt-get install -y \
    openssh-client \
    iproute2 \
    tcpdump \
    iputils-ping \
    && rm -rf /var/lib/apt/lists/*

CMD ["tail", "-f", "/dev/null"]

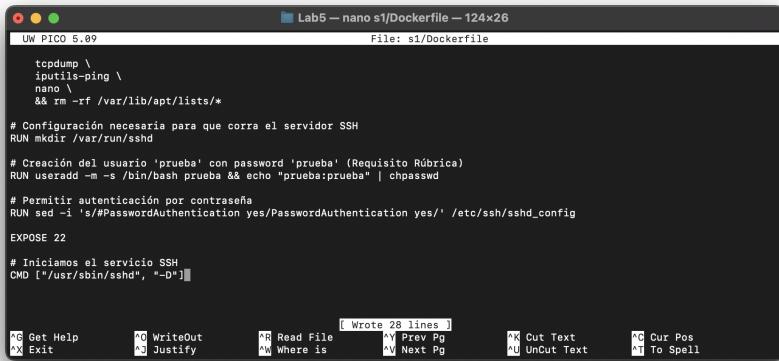
```

The terminal window has a title bar "Lab5 — nano c3/Dockerfile — 124x24" and a status bar at the bottom with various keyboard shortcuts for nano editor.

Figura 5: comandos zsh para el cliente 3

### 1.1.4. C4/S1

El contenedor S1 se configura para actuar simultáneamente como Cliente 4 y como Servidor SSH. Por esta razón, se instalan tanto los paquetes de cliente como los de servidor sobre la imagen base de Ubuntu 22.10, adicionalmente se incluyen las directivas para la inicialización correcta del demonio SSH.



```

tcpdump \
iputils-ping \
nano \
&& rm -rf /var/lib/apt/lists/*

# Configuración necesaria para que corra el servidor SSH
RUN mkdir /var/run/sshd

# Creación del usuario 'prueba' con password 'prueba' (Requisito Rúbrica)
RUN useradd -m -s /bin/bash prueba && echo "prueba:prueba" | chpasswd

# Permitir autenticación por contraseña
RUN sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd_config

EXPOSE 22

# Iniciamos el servicio SSH
CMD ["/usr/sbin/sshd", "-D"]

```

The terminal window has a title bar "Lab5 — nano s1/Dockerfile — 124x24" and a status bar at the bottom with various keyboard shortcuts for nano editor, including a note "[ Wrote 28 lines ]".

Figura 6: comandos zsh del servidor, s1

## 1.2. Creación de las credenciales para S1

### 1.2.1. Creación de la red privada

Se procede a la creación de una red **bridge** dedicada dentro del entorno Docker, denominada **red\_lab5**. Esta red se utiliza para interconectar de forma privada los contenedores Cliente (C1, C2, C3) con el Servidor (S1). Al servidor S1 se le asigna una dirección IP estática (172.20.0.10) para asegurar que la conexión SSH de los clientes se establezca de forma consistente. La puesta en marcha de esta infraestructura es esencial para el posterior análisis del tráfico.

```

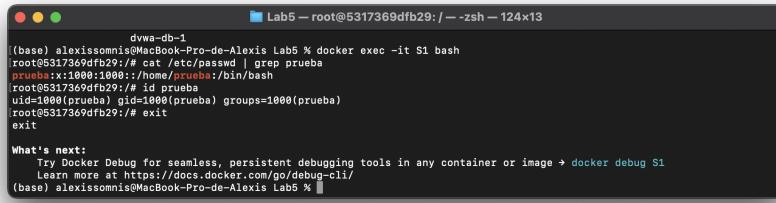
[base] alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker network create --subnet=172.20.0.0/16 red_lab5
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker run -d --name S1 --net red_lab5 --ip 172.20.0.10 lab5-s1
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker run -d --name C1 --net red_lab5 lab5-c1
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker run -d --name C2 --net red_lab5 lab5-c2
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker run -d --name C3 --net red_lab5 lab5-c3
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
e974032d7ea5 lab5-c3 "tail -f /dev/null" 9 seconds ago Up 8 seconds
d3009dbd7664 lab5-c2 "tail -f /dev/null" 9 seconds ago Up 8 seconds
d3efef3242cc lab5-c1 "tail -f /dev/null" 10 seconds ago Up 9 seconds
5317369dfb29 lab5-s1 "/usr/sbin/sshd -D" 18 seconds ago Up 17 seconds 22/tcp
33cf54bad698 ghar.io/digininja/dvwa:latest "docker-php-entrypoint" 7 weeks ago Up 10 hours 0.0.0.0:80->80/tcp,
[::]:80->80/tcp dvwa-dvwa-1
c93468e41348 mariadb:10 "docker-entrypoint.s..." 7 weeks ago Up 10 hours 3306/tcp
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 %

```

Figura 7: red simulada

### 1.2.2. Verificación de credenciales de acceso

La creación de las credenciales de acceso para el servidor S1 es realizada directamente durante la fase de construcción de la imagen Docker **lab5-s1** (ver sección 1.1.4). El usuario **prueba**, con contraseña **prueba**, es generado mediante la combinación de los comandos **useradd** y **chpasswd** insertada en el Dockerfile. La correcta implementación de las credenciales es verificada accediendo al contenedor S1 y confirmando la existencia del usuario **prueba** con el UID y GID 1000, lo cual valida que el usuario se encuentra listo para la autenticación SSH .



The screenshot shows a terminal window titled 'Lab5 — root@5317369dfb29: / ~zsh — 124x13'. The command 'cat /etc/passwd | grep prueba' is run, showing the user 'prueba' with uid 1000 and gid 1000. The window also displays a message about Docker Debug and a prompt '(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 %'.

```
dvwa-db-1
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker exec -it S1 bash
root@5317369dfb29:/# cat /etc/passwd | grep prueba
prueba:x:1000:1000::/home/prueba:/bin/bash
root@5317369dfb29:/# id prueba
uid=1000(prueba) gid=1000(prueba) groups=1000(prueba)
root@5317369dfb29:/# exit
exit

What's next:
Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug S1
Learn more at https://docs.docker.com/go/debug-cli/
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 %
```

Figura 8: credenciales para s1

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

El análisis del tráfico capturado entre el cliente C1 (Ubuntu 16.10) y el servidor S1 revela un patrón de handshake característico de la versión OpenSSH 7.2p2, inicialmente, se observa el intercambio de banners en texto plano, donde el cliente envía un paquete de 107 bytes identificando su versión. Posteriormente, la negociación de algoritmos comienza con el mensaje Client Key Exchange Init, el cual posee una longitud de 1498 bytes, seguido por la respuesta del servidor (Server Key Exchange Init) con un tamaño de 1146 bytes. El intercambio de claves efímeras se concreta mediante un paquete de 114 bytes por parte del cliente (ECDH Init) y una respuesta de 662 bytes del servidor (ECDH Reply), la cual incluye las nuevas llaves (New Keys). Finalmente, a partir de la lista de algoritmos propuesta en la fase inicial, se obtiene la huella digital HASSH 0e4584cb9f2dd077dbf8ba0df8112d8e, lo que permite identificar únicamente la configuración criptográfica utilizada por este cliente antiguo. Se adjunta en github la captura de wireshark para comprobaciones

A partir de la lista de algoritmos negociados en texto plano, se obtuvo la huella digital HASSH del cliente:

0e4584cb9f2dd077dbf8ba0df8112d8e

Este identificador es único para la configuración criptográfica de la versión OpenSSH 7.2p2 utilizada en el contenedor.

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

### 1 DESARROLLO (PARTE 1)

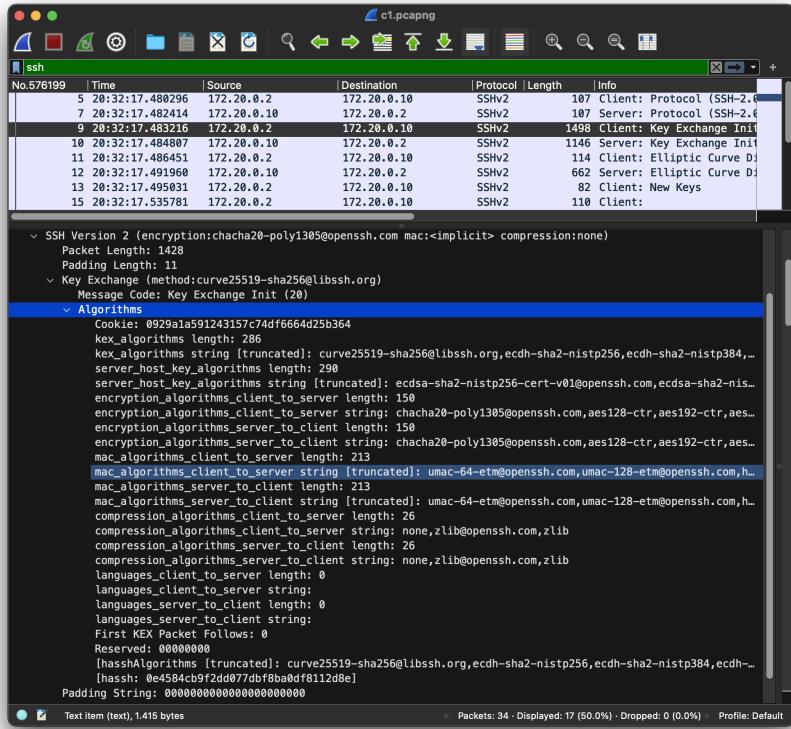


Figura 9: captura de paquetes c1 a s1

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

Para la segunda captura, se analiza el tráfico proveniente del cliente C2 (Ubuntu 18.10) hacia el servidor S1. El análisis de la captura revela una actualización en la versión del software cliente respecto al escenario anterior, identificándose ahora como OpenSSH 7.7p1. A pesar de esta actualización, se observa que los tamaños de los paquetes del *handshake* se mantienen consistentes con los de C1, registrándose los siguientes valores:

- **Client Protocol:** El cliente se identifica mediante el banner SSH-2.0-OpenSSH\_7.7p1 Ubuntu-4ubuntu0.3, enviado en un paquete de **107 bytes**.
- **Client Key Exchange Init:** El paquete de negociación mantiene un tamaño de **1498 bytes**. No obstante, la inspección interna confirma un cambio en la prioridad y selección de los algoritmos criptográficos.
- **Server Key Exchange Init:** La respuesta del servidor conserva su tamaño de **1146 bytes**.

- **Intercambio de Claves (ECDH):** La inicialización por parte del cliente (*ECDH Init*) ocupa **114 bytes**, y la respuesta del servidor con las nuevas llaves (*ECDH Reply + New Keys*) ocupa **662 bytes**.

Debido a la modificación en el orden y tipo de algoritmos soportados por la nueva versión, la huella digital cambia. El HASSH calculado para este cliente es:

**06046964c022c6407d15a27b12a6a4fb**

Este valor confirma que, aunque la volumetría del tráfico es similar a la versión anterior, la firma criptográfica permite distinguir que se trata de un cliente actualizado en un entorno Ubuntu 18.10

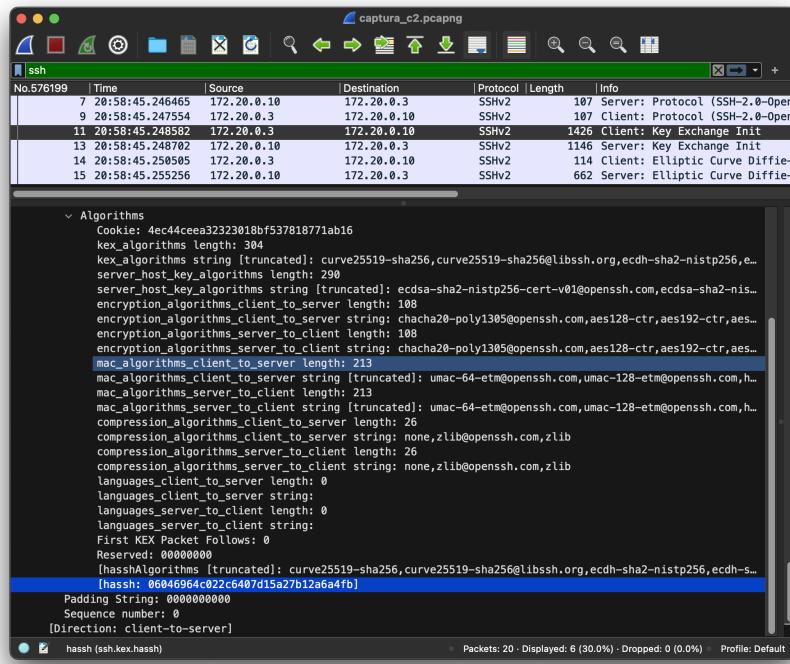


Figura 10: captura de paquetes c2 a s1

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

En el tercer escenario, correspondiente al cliente C3 (Ubuntu 20.10), se observa una evolución significativa en el protocolo. El análisis de la captura `captura_c3.pcapng` revela que la versión del cliente se ha actualizado a OpenSSH 8.3p1. Este cambio introduce nuevos algoritmos de intercambio de claves post-cuánticos (como `sntrup761x25519`), lo que impacta directamente en el tamaño de los paquetes de negociación:

- **Client Protocol:** El cliente anuncia su versión SSH-2.0-OpenSSH\_8.3p1 Ubuntu-1ubuntu0.1 en un paquete de **107 bytes**.
- **Client Key Exchange Init:** Este paquete aumenta su tamaño a **1578 bytes** (comparado con los 1498 bytes de C1 y C2), reflejando la inclusión de una lista más extensa y moderna de algoritmos criptográficos.
- **Server Key Exchange Init:** La respuesta del servidor se mantiene en **1146 bytes**.
- **Intercambio de Claves (ECDH):** El paquete de inicialización del cliente mantiene los **114 bytes**, y la respuesta del servidor ocupa **662 bytes**.

La incorporación de estos nuevos algoritmos genera una huella digital única y diferente a las anteriores. El HASSH calculado para C3 es:

`ae8bd7dd09970555aa4c6ed22adbf56e`

Este valor confirma que el cliente C3 está utilizando una configuración de seguridad moderna, diferenciándose claramente de las versiones más antiguas observadas en C1 y C2.

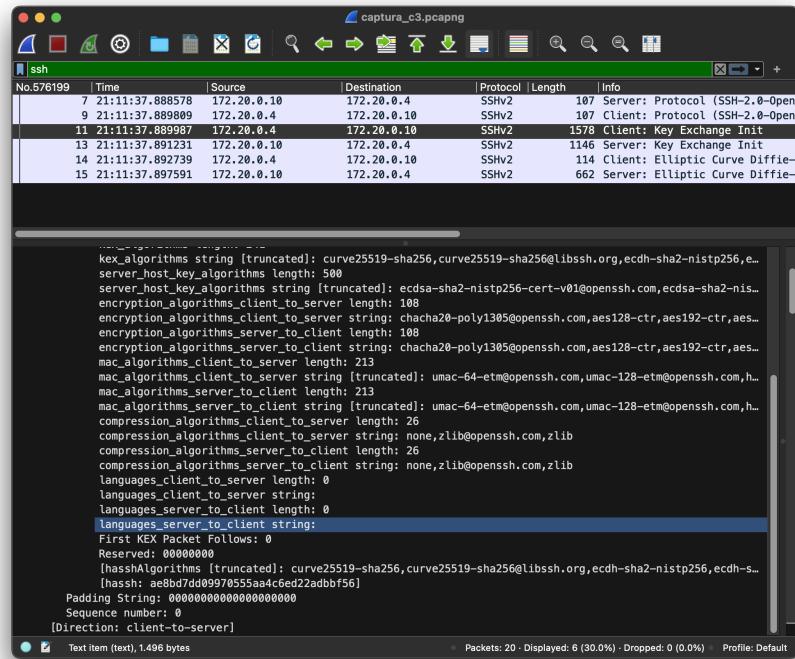


Figura 11: captura de paquetes c3 a s1

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

En el cuarto y último escenario, el cliente C4 se ejecuta en el mismo contenedor que el servidor (Ubuntu 22.10), estableciendo una conexión local a través de la interfaz de *loopback* (`lo`). El análisis de la captura `captura_c4.pcapng` muestra que el cliente utiliza la versión más reciente observada en el laboratorio: OpenSSH 9.0p1. Los paquetes capturados presentan las siguientes características:

- **Client Protocol:** El cliente envía su banner `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` en un paquete de **107 bytes**.
- **Client Key Exchange Init:** Este paquete tiene una longitud de **1572 bytes**. Aunque muy similar al de C3, la diferencia de tamaño y versión indica ajustes menores en la lista de algoritmos o en el manejo del *padding* en esta versión más reciente.
- **Server Key Exchange Init:** La respuesta del servidor mantiene un tamaño consistente de **1146 bytes**.
- **Intercambio de Claves (ECDH):** Debido a la actualización del protocolo, el tamaño del paquete de inicialización del cliente aumenta a **1276 bytes** (comparado con los 114 bytes de los casos anteriores), lo que sugiere el uso de un algoritmo de intercambio de claves post-cuántico híbrido (probablemente `sntrup761x25519`) por defecto. La respuesta del servidor es de **1632 bytes**.

El uso de algoritmos híbridos post-cuánticos en OpenSSH 9.0 genera una huella digital HASSH completamente nueva y distinta a las anteriores:

`78c05d999799066a2b4554ce7b1585a6`

Este resultado evidencia cómo las versiones más modernas de OpenSSH (9.x) introducen cambios significativos en la criptografía por defecto para prepararse ante amenazas futuras. (Nota: recuerda tomar la foto de C4 si no la tienes).

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

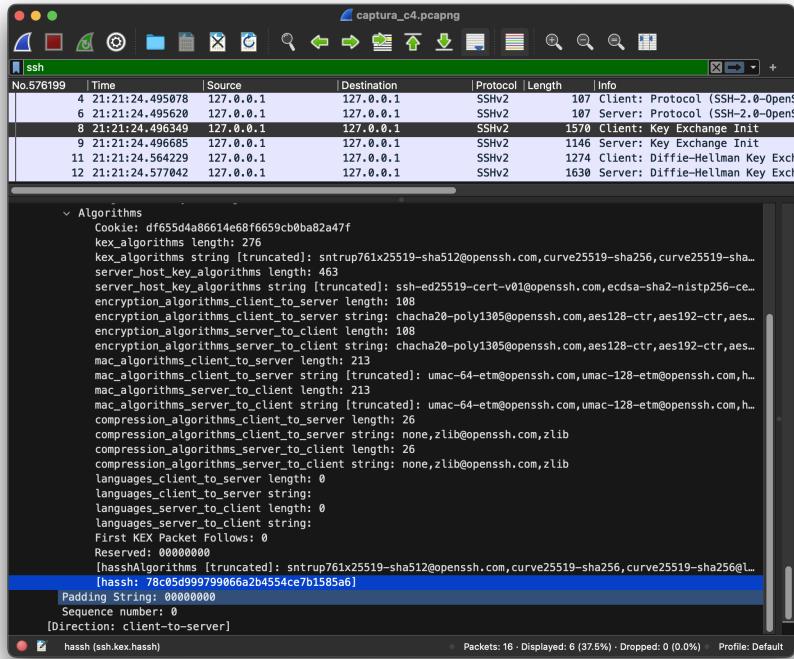


Figura 12: captura de paquetes c4 a s1, siendo el mismo

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

El análisis transversal de las capturas realizadas en los cuatro escenarios permite identificar que, previo al establecimiento del canal cifrado, el protocolo SSH expone información crítica en texto plano. A continuación se detalla el contenido visible para cada cliente:

#### **1.7.1. C1 (Ubuntu 16.10)**

En los primeros paquetes del flujo, se visualiza claramente el banner de versión (SSH-2.0-OpenSSH\_7.2p2...). El paquete *Key Exchange Init* expone la lista completa de algoritmos soportados para *kex* (intercambio de claves), *server\_host\_key*, *encryption* (cifrado simétrico), *mac* (integridad) y *compression*. Esta exposición permite enumerar las capacidades criptográficas del cliente sin necesidad de desencriptar el tráfico.

#### **1.7.2. C2 (Ubuntu 18.10)**

De manera similar, el cliente C2 revela su versión (OpenSSH 7.7p1) y su propuesta de algoritmos. Aunque la estructura es idéntica a C1, el contenido de las listas varía ligeramente en el orden de preferencia, priorizando algoritmos de curva elíptica más modernos, información que es legible directamente desde la captura.

### 1.7.3. C3 (Ubuntu 20.10)

El cliente C3 expone una lista de algoritmos significativamente más extensa en su mensaje *Key Exchange Init*, incluyendo propuestas para algoritmos post-cuánticos híbridos. Toda esta negociación, junto con el banner de versión OpenSSH 8.3p1, se transmite sin cifrar, permitiendo la identificación precisa del sistema operativo y software subyacente.

### 1.7.4. C4/S1 (Ubuntu 22.10)

En el escenario local, la captura evidencia la misma exposición de metadatos. Se observan en texto plano el banner OpenSSH 9.0p1 y la negociación de parámetros. A pesar de ser la versión más reciente, el protocolo obliga a que esta fase inicial sea legible para acordar los términos de seguridad, lo que constituye una ventana de reconocimiento pasivo para cualquier observador en la red.

## 1.8. Diferencia entre C1 y C2

La principal diferencia técnica entre C1 y C2 radica en la versión del protocolo y la priorización de algoritmos. Aunque ambos generan un paquete de negociación de tamaño idéntico (1498 bytes), el análisis del HASSH revela firmas distintas ('0e45...' vs '0604...'). Esto indica que, si bien la cantidad de datos es la misma, C2 (OpenSSH 7.7p1) ha modificado el orden de preferencia o sustituido algoritmos obsoletos por variantes más seguras dentro del mismo espacio de bytes.

## 1.9. Diferencia entre C2 y C3

Entre C2 y C3 se observa el salto evolutivo más notable. El tamaño del paquete de negociación aumenta de 1498 bytes a 1578 bytes. Esta diferencia de 80 bytes refleja la incorporación de nuevos algoritmos en OpenSSH 8.3p1 (C3), específicamente métodos de intercambio de claves más robustos, además el HASSH cambia completamente, evidenciando una reestructuración profunda de la suite criptográfica por defecto en Ubuntu 20.10.

## 1.10. Diferencia entre C3 y C4

La comparación entre C3 y C4 muestra un refinamiento en la implementación. A pesar de que C4 (OpenSSH 9.0p1) es una versión más moderna, el tamaño de su paquete de negociación disminuye levemente a 1572 bytes (frente a los 1578 de C3). Esto sugiere una optimización en la lista de algoritmos, eliminando opciones obsoletas o ajustando el padding, pero el cambio más relevante es el aumento del tamaño del paquete de inicialización de claves (ECDH Init), que crece a 1276 bytes en C4, indicando el uso por defecto de un esquema de intercambio de claves híbrido más pesado y seguro.

## 2. Desarrollo (Parte 2)

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

Para determinar la versión del cliente utilizado por el informante, se realizó un análisis comparativo entre el patrón de tráfico proporcionado en el enunciado y las capturas obtenidas experimentalmente en la Parte 1.

El indicador clave identificado en la evidencia del informante es la longitud del paquete **Client: Key Exchange Init**, el cual presenta un tamaño exacto de **1578 bytes**. Al contrastar este valor con los resultados experimentales, se observa que:

- Los clientes C1 (OpenSSH 7.2p2) y C2 (OpenSSH 7.7p1) generan un paquete de 1498 bytes.
- El cliente C4 (OpenSSH 9.0p1) genera un paquete de 1572 bytes.
- **El cliente C3 (OpenSSH 8.3p1) genera un paquete de exactamente 1578 bytes.**

Esta coincidencia exacta en el tamaño del campo de negociación permite concluir que el cliente incógnito corresponde a la versión **OpenSSH 8.3p1** sobre Ubuntu 20.10, replicada en el escenario C3, por lo tanto el cliente 3 sería el informante

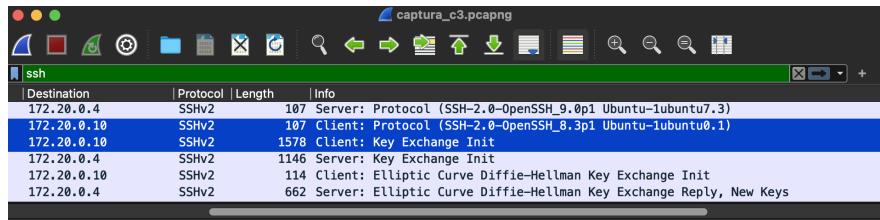
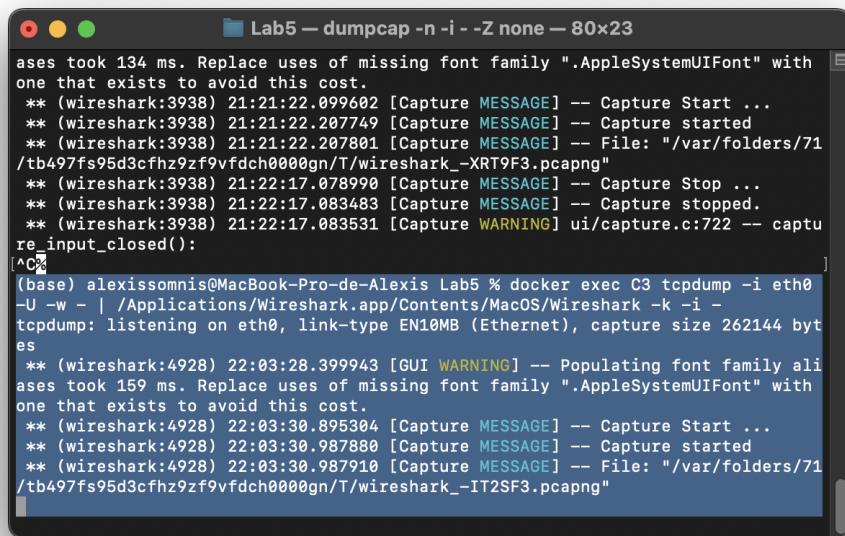


Figura 13: evidencia version de cliente

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

La replicación del tráfico tiene como objetivo validar empíricamente que el cliente C3 es efectivamente el utilizado en el escenario del informante. Este proceso consiste en reproducir manualmente la conexión para generar y capturar la misma huella de red.

En primer lugar, se configura el entorno de monitoreo estableciendo una tubería de captura desde el contenedor C3 hacia la interfaz gráfica de Wireshark, lo que permite visualizar los paquetes en tiempo real.



```
Lab5 — dumpcap -n -i -Z none — 80x23
ases took 134 ms. Replace uses of missing font family ".AppleSystemUIFont" with
one that exists to avoid this cost.
** (wireshark:3938) 21:21:22.099602 [Capture MESSAGE] -- Capture Start ...
** (wireshark:3938) 21:21:22.207749 [Capture MESSAGE] -- Capture started
** (wireshark:3938) 21:21:22.207801 [Capture MESSAGE] -- File: "/var/folders/71
/tb497fs95d3cfhz9zf9vfdch0000gn/T/wireshark_-XRT9F3.pcapng"
** (wireshark:3938) 21:22:17.078990 [Capture MESSAGE] -- Capture Stop ...
** (wireshark:3938) 21:22:17.083483 [Capture MESSAGE] -- Capture stopped.
** (wireshark:3938) 21:22:17.083531 [Capture WARNING] ui/capture.c:722 -- captu
re_input_closed():
[ ^C ]
(base) alexissomnis@MacBook-Pro-de-Alexis Lab5 % docker exec C3 tcpdump -i eth0
-U -w - | /Applications/Wireshark.app/Contents/MacOS/Wireshark -k -i -
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
es
** (wireshark:4928) 22:03:28.399943 [GUI WARNING] -- Populating font family ali
ases took 159 ms. Replace uses of missing font family ".AppleSystemUIFont" with
one that exists to avoid this cost.
** (wireshark:4928) 22:03:30.895304 [Capture MESSAGE] -- Capture Start ...
** (wireshark:4928) 22:03:30.987880 [Capture MESSAGE] -- Capture started
** (wireshark:4928) 22:03:30.987910 [Capture MESSAGE] -- File: "/var/folders/71
/tb497fs95d3cfhz9zf9vfdch0000gn/T/wireshark_-IT2SF3.pcapng"
```

Figura 14: Configuración del monitor de red en C3

Simultáneamente, se accede a la terminal del cliente C3 y se ejecuta el comando de conexión SSH hacia la dirección IP del servidor (172.20.0.10). Se completa el proceso de autenticación aceptando la huella del servidor e ingresando las credenciales del usuario **prueba**, logrando un acceso exitoso a la línea de comandos remota, tal como se evidencia en la confirmación del sistema operativo.

```

Lab5 – prueba@5317369dfb29: ~ — docker exec -it C3 bash — 127x40
Permission denied, please try again.
Permission denied, please try again.
Permission denied (publickey,password).
(base) alexissomnis@MacBook-Pro-de-Alexis: Lab5 % docker exec C2 ssh prueba@172.20.0.10
Pseudo-terminal will not be allocated because stdin is not a terminal.
Host key verification failed.
(base) alexissomnis@MacBook-Pro-de-Alexis: Lab5 % docker exec C3 ssh prueba@172.20.0.10
Pseudo-terminal will not be allocated because stdin is not a terminal.
Host key verification failed.
(base) alexissomnis@MacBook-Pro-de-Alexis: Lab5 % docker exec S1 ssh prueba@127.0.0.1
Pseudo-terminal will not be allocated because stdin is not a terminal.
Host key verification failed.
(base) alexissomnis@MacBook-Pro-de-Alexis: Lab5 % docker exec -it C3 bash
root@974a832d7ea5: # prueba
bash: prueba: command not found
root@974a832d7ea5: # ssh prueba@172.20.0.10
[The authenticity of host '172.20.0.10 (172.20.0.10)' can't be established.
ECDSA key fingerprint is SHA256:uygDVPUUqJED3nBy87diwAEATWJKamz9rPBh1/BRo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.20.0.10' (ECDSA) to the list of known hosts.
prueba@172.20.0.10's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.10.14-linuxkit 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.

prueba@974a832d7ea5: ~$ ]

```

Figura 15: Ejecución de la conexión SSH desde C3

Finalmente, se analiza el tráfico resultante en Wireshark. Se observa que la comunicación se establece desde la IP del cliente (172.20.0.4) hacia el servidor, generando un flujo de paquetes idéntico al patrón de referencia (1578 bytes para el *Client Key Exchange Init*). Esto confirma que la configuración de algoritmos y versiones de C3 produce exactamente la firma de tráfico buscada.

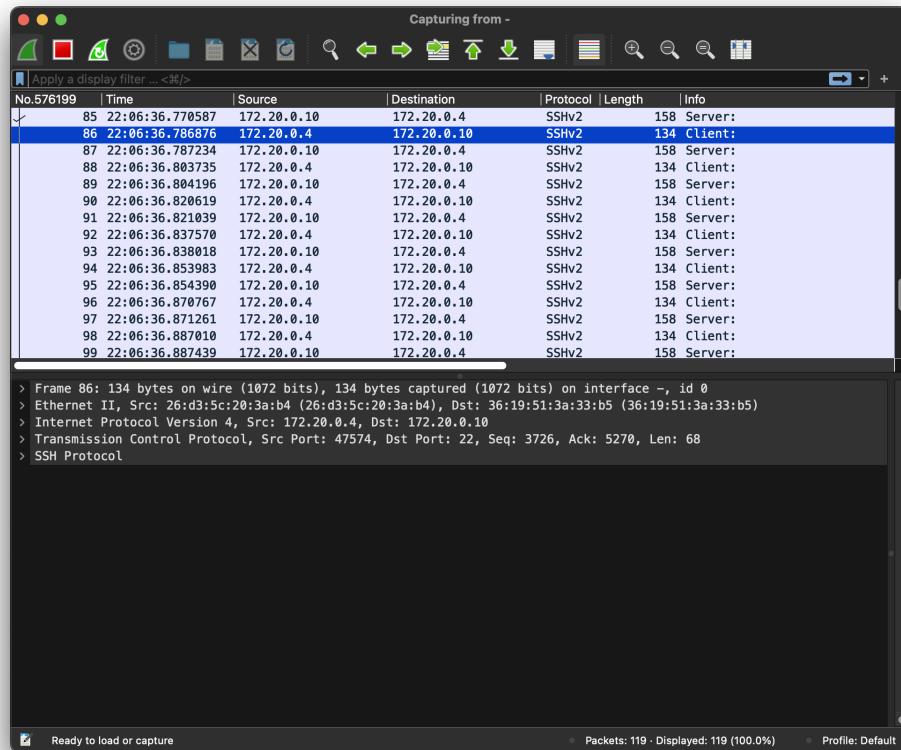


Figura 16: Análisis del tráfico replicado en Wireshark

### 3. Desarrollo (Parte 3)

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

El objetivo de esta sección es modificar la configuración del servidor SSH para reducir el tamaño del paquete *Server Key Exchange Init* a menos de 300 bytes. Por defecto, este paquete tenía un tamaño de 1146 bytes debido a la gran cantidad de algoritmos ofrecidos.

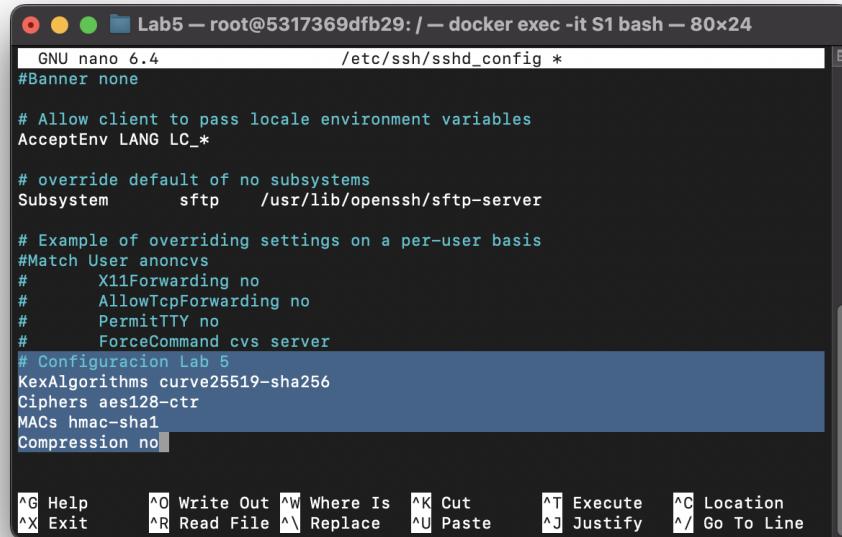
Para lograr la reducción, se realizaron los siguientes pasos en el contenedor del servidor (S1):

- Edición de la configuración:** Se accedió al archivo `/etc/ssh/sshd_config` utilizando el editor `nano`. Se limitó explícitamente la oferta de algoritmos a una sola opción por categoría (intercambio de claves, cifrado y MAC), además de deshabilitar la compresión. Las líneas añadidas al final del archivo fueron:

```
KexAlgorithms curve25519-sha256
Ciphers aes128-ctr
```

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

```
MACs hmac-sha1  
Compression no
```



```
GNU nano 6.4          /etc/ssh/sshd_config *
#Banner none

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

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

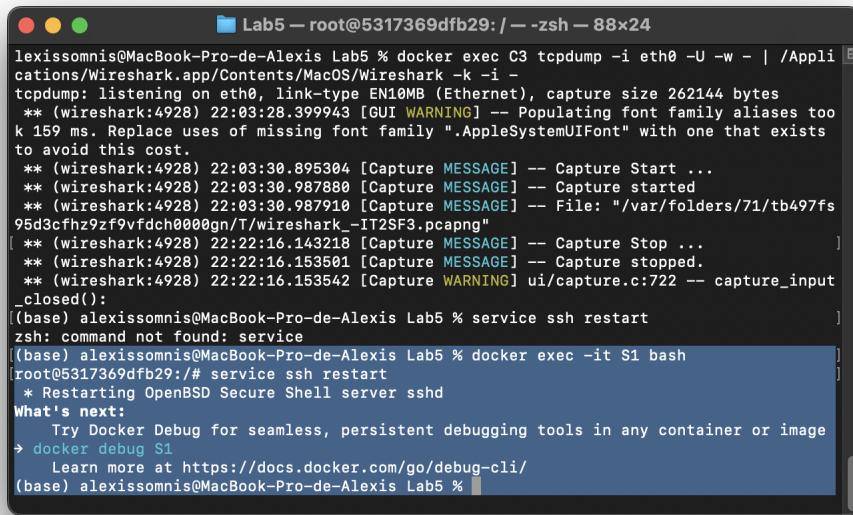
# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server
# Configuration Lab 5
KexAlgorithms curve25519-sha256
Ciphers aes128-ctr
MACs hmac-sha1
Compression no

^G Help      ^O Write Out ^W Where Is  ^K Cut      ^T Execute   ^C Location
^X Exit      ^R Read File ^V Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

Figura 17: Modificación del archivo sshd\_config en el servidor S1

2. **Reinicio del servicio:** Se aplicaron los cambios reiniciando el demonio SSH mediante el comando `service ssh restart`, asegurando que la nueva configuración restrictiva estuviera activa.

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



The screenshot shows a terminal window titled "Lab5 — root@5317369dfb29: / -- zsh -- 88x24". The terminal output is as follows:

```
lexisomnis@MacBook-Pro-de-Alexis Lab5 % docker exec C3 tcpdump -i eth0 -U -w - | Applications/Wireshark.app/Contents/MacOS/Wireshark -k -i -tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
** (wireshark:4928) 22:03:28.399943 [GUI WARNING] -- Populating font family aliases took 159 ms. Replace uses of missing font family ".AppleSystemUIFont" with one that exists to avoid this cost.
** (wireshark:4928) 22:03:30.895304 [Capture MESSAGE] -- Capture Start ...
** (wireshark:4928) 22:03:30.987880 [Capture MESSAGE] -- Capture started
** (wireshark:4928) 22:03:30.987910 [Capture MESSAGE] -- File: "/var/folders/71/tb497fs95d3cfhz9zf9vfdch0000gn/T/wireshark_IT2SF3.pcapng"
[ ** (wireshark:4928) 22:22:16.143218 [Capture MESSAGE] -- Capture Stop ... ]
** (wireshark:4928) 22:22:16.153501 [Capture MESSAGE] -- Capture stopped.
** (wireshark:4928) 22:22:16.153542 [Capture WARNING] ui/capture.c:722 -- capture_input _closed():
[(base) alexisomnis@MacBook-Pro-de-Alexis Lab5 % service ssh restart
zsh: command not found: service
[(base) alexisomnis@MacBook-Pro-de-Alexis Lab5 % docker exec -it S1 bash
root@5317369dfb29:/# service ssh restart
* Restarting OpenBSD Secure Shell server sshd
What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image
→ docker debug S1
Learn more at https://docs.docker.com/go/debug-cli/
(base) alexisomnis@MacBook-Pro-de-Alexis Lab5 % ]
```

Figura 18: Reinicio del servicio SSH para aplicar cambios

3. **Verificación del tráfico:** Se estableció una nueva conexión desde el cliente C2 hacia el servidor modificado y se capturó el tráfico resultante. El análisis en Wireshark confirma que el tamaño del paquete *Server: Key Exchange Init* se redujo drásticamente a **258 bytes**, cumpliendo exitosamente con el requerimiento de ser menor a 300 bytes.

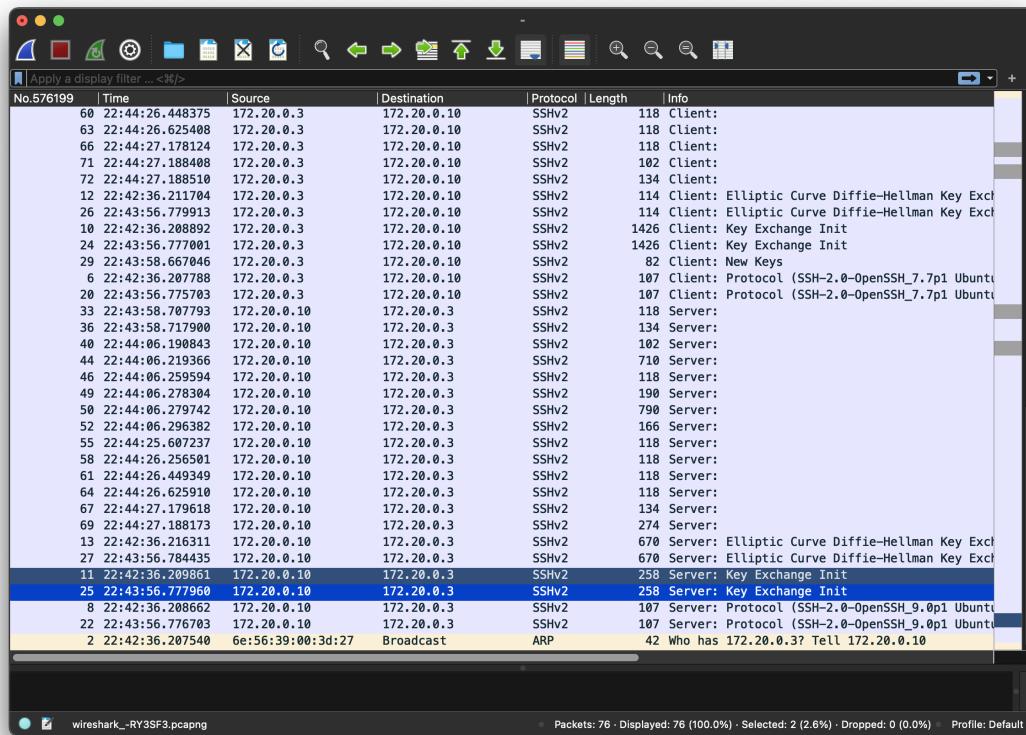


Figura 19: Evidencia en Wireshark: Paquete reducido a 258 bytes

## 4. Desarrollo (Parte 4)

### 4.1. Explicación OpenSSH en general

OpenSSH (*Open Secure Shell*) se define como una suite de herramientas de conectividad diseñada para proporcionar comunicaciones cifradas sobre redes inseguras. Su implementación se basa en el protocolo SSH y opera bajo una arquitectura cliente-servidor, teniendo como función principal el reemplazo de herramientas de administración remota obsoletas y vulnerables, tales como Telnet o FTP, las cuales transmiten credenciales y datos en texto plano

El funcionamiento de la suite se estructura en torno a la interacción coordinada de dos componentes principales:

- **El Servidor (sshd):** Corresponde al proceso demonio que se ejecuta en el sistema remoto (en este laboratorio, el contenedor S1). Su responsabilidad reside en escuchar las solicitudes de conexión en un puerto específico (estándar 22), gestionar el intercambio de claves y validar la autenticación de los usuarios

- **El Cliente (ssh):** Constituye el programa ejecutado en el sistema local (contenedores C1, C2, C3) encargado de iniciar la negociación, transmitir los comandos del usuario y procesar la respuesta cifrada enviada por el servidor

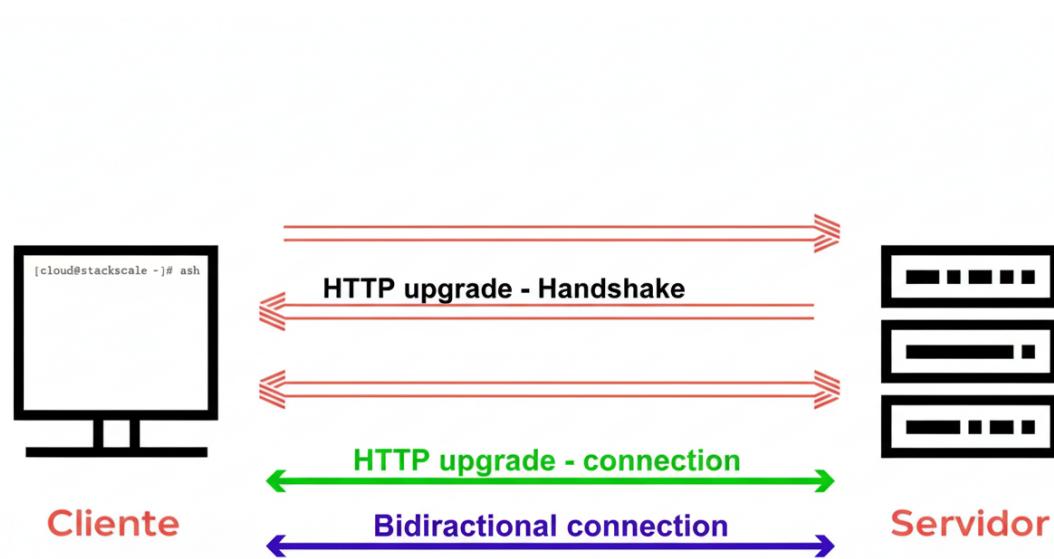


Figura 20: representacion fases de ssh

El establecimiento de una sesión segura mediante OpenSSH se consolida a través de una secuencia de fases ordenadas, las cuales se evidencian en las capturas de tráfico realizadas durante la experiencia práctica

1. **Negociación (Handshake):** Se acuerdan los algoritmos criptográficos para el intercambio de claves, el cifrado simétrico y la integridad del mensaje (etapa analizada en los paquetes *Key Exchange Init*).
2. **Autenticación:** Se verifica la identidad del usuario mediante mecanismos de prueba, tales como contraseñas o, para mayor robustez, pares de claves asimétricas (públicas/privadas).
3. **Canal Seguro:** Una vez autenticado el usuario, se establece un túnel cifrado donde la totalidad de la información (incluyendo comandos interactivos y transferencias de archivos mediante `scp` o `sftp`) se transmite protegida contra intercepciones y modificaciones.

## 4.2. Capas de Seguridad en OpenSSH

El protocolo OpenSSH implementa una arquitectura de seguridad en capas diseñada para mitigar riesgos en redes no confiables, cada fase de la conexión contribuye al cumplimiento de los principios fundamentales de la seguridad de la información, a continuación, se detallan los mecanismos técnicos empleados

### 4.2.1. Confidencialidad

La confidencialidad se garantiza mediante el establecimiento de un canal cifrado simétricamente, tras la negociación inicial (KEX), todo el tráfico de la sesión (incluyendo credenciales, comandos y salida de terminal) es encriptado utilizando algoritmos robustos como AES-CTR, ChaCha20-Poly1305 o AES-GCM. Esto asegura que, aunque un atacante intercepte los paquetes, el contenido sea ilegible sin las llaves de sesión efímeras generadas durante el *handshake*.

### 4.2.2. Integridad

Para asegurar que los datos no han sido alterados en tránsito, OpenSSH emplea algoritmos de Código de Autenticación de Mensajes (MAC), tales como `hmac-sha2-256` o `umac-128`, cada paquete enviado incluye un resumen criptográfico (hash) que el receptor verifica antes de procesar la información, en los cífrados autenticados modernos (como ChaCha20-Poly1305), la integridad es una propiedad intrínseca del algoritmo de cifrado, proporcionando una verificación aún más eficiente.

### 4.2.3. Autenticidad

La autenticidad se aborda en dos direcciones:

- **Autenticidad del Servidor:** Se logra mediante las *Host Keys* (claves de host). Al conectarse por primera vez, el cliente almacena la huella digital de la clave pública del servidor, en conexiones subsecuentes, se verifica matemáticamente que el servidor posea la clave privada correspondiente, previniendo ataques de *Man-in-the-Middle* (MitM).
- **Autenticidad del Cliente:** Se verifica mediante la autenticación de usuario, ya sea por contraseña (basada en conocimiento) o, preferentemente, mediante criptografía asimétrica (pares de claves pública/privada), donde el cliente debe demostrar la posesión de la clave privada sin transmitirla por la red.

### 4.2.4. Disponibilidad y No Repudio

Si bien SSH no está diseñado primariamente para garantizar disponibilidad (protección contra DoS), implementa mecanismos de control de acceso y límites de tasa de conexión en la configuración del demonio, respecto al no repudio, el uso de claves criptográficas asimétricas individuales permite vincular acciones a una identidad específica con mayor certeza que las contraseñas compartidas, aunque el protocolo en sí mismo no genera pruebas forenses de no repudio por defecto.

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

A partir del análisis del tráfico capturado y la configuración implementada en la experiencia, se determina que si bien OpenSSH provee un canal seguro robusto, existen limitaciones intrínsecas respecto a ciertos principios de seguridad en el contexto evaluado:

#### 4.3.1. Confidencialidad Parcial (Metadatos)

Aunque el contenido de la sesión (comandos y datos) se encuentra cifrado, se evidencia que la **confidencialidad no es absoluta durante la fase de negociación**. Los mensajes de intercambio de versiones (*Protocol Version Exchange*) y la negociación de algoritmos (*Key Exchange Init*) se transmiten en texto plano. Esta exposición de metadatos permite a un observador externo realizar técnicas de *fingerprinting* (como se demostró mediante el cálculo del HASSH), revelando información sensible sobre la infraestructura y versiones de software sin necesidad de descifrar el túnel.

#### 4.3.2. No Repudio (Limitado)

El principio de **no repudio no se garantiza plenamente** con la configuración utilizada, al emplear autenticación basada en contraseñas compartidas o genéricas (usuario prueba), el sistema solo valida el conocimiento del secreto, pero no vincula la acción de manera irrefutable a una identidad física específica. En caso de incidente, el servidor no genera evidencia criptográfica suficiente para impedir que un usuario niegue haber realizado la conexión, ya que la credencial pudo haber sido utilizada por cualquier agente con conocimiento de la misma.

## Conclusiones y comentarios

### 5. Conclusiones y comentarios

La implementación de entornos aislados mediante Docker establece un marco de referencia para la caracterización técnica del protocolo OpenSSH en sus distintas versiones. El análisis comparativo del tráfico de red evidencia una correlación directa entre la actualización del software cliente y la modificación de los parámetros de negociación criptográfica. Se observa que la incorporación de algoritmos modernos, como los esquemas de intercambio de claves híbridos post-cuánticos en versiones recientes, altera significativamente la longitud de los paquetes de *handshake* y genera huellas digitales (HASSH) distintivas para cada implementación.

El examen de los paquetes iniciales confirma la exposición de metadatos críticos en texto plano durante la fase de negociación. La disponibilidad de información sobre versiones de software y listas de algoritmos soportados valida la factibilidad de realizar técnicas de *fingerprinting* pasivo sin necesidad de desencriptar el túnel seguro. Este hecho técnico demuestra

## 5 CONCLUSIONES Y COMENTARIOS

---

que la confidencialidad del protocolo no es absoluta en sus etapas tempranas, permitiendo la identificación precisa de los extremos de la comunicación.

Finalmente, la reconfiguración del servidor S1 mediante la restricción de suites criptográficas en el archivo `sshd_config` prueba la capacidad de control sobre la negociación del protocolo. La reducción del tamaño del paquete *Server Key Exchange Init* de 1146 a 258 bytes verifica que la limitación de algoritmos no solo reduce la superficie de ataque al eliminar opciones obsoletas, sino que también optimiza el volumen de datos transmitidos durante el establecimiento de la sesión.