Informe Laboratorio 5

Sección 1

Alumno Bruno Rosales Franz
e-mail: bruno.rosales@mail.udp.cl

Noviembre de 2025

Índice

Descripción de actividades

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

Para lo anterior deberá:

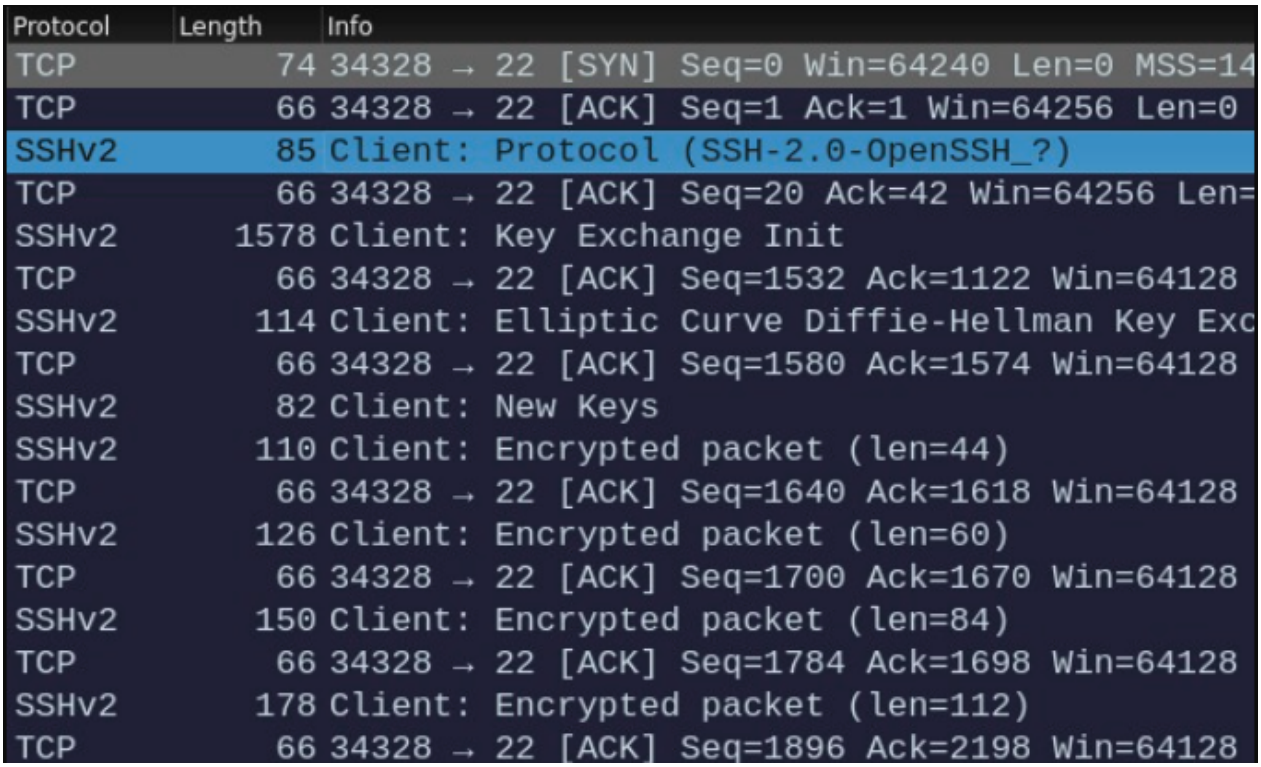
- Crear 4 contenedores en Docker por medio de un DockerFile, donde cada uno tendrá el siguiente SO: Ubuntu 16.10, Ubuntu 18.10, Ubuntu 20.10 y Ubuntu 22.10 a los cuales se llamarán C1, C2, C3 y C4 respectivamente.
  El equipo con Ubuntu 22.10 también será utilizado como S1.
- Para cada uno de ellos, 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 $\rightarrow$ S1
  - C2 $\rightarrow$ S1
  - C3 $\rightarrow$ S1
  - C4 $\rightarrow$ 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.

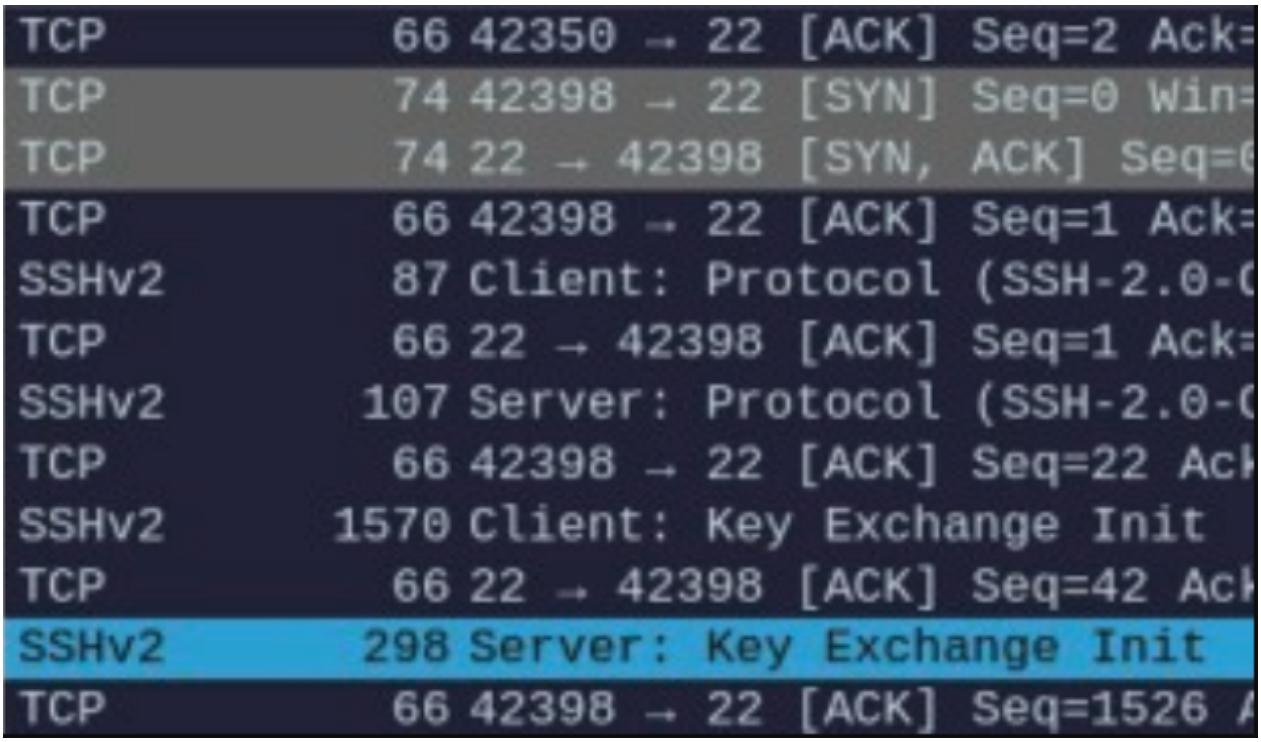


Figura 2: Captura del Key Exchange

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 escribe un Dockerfile genérico que sirve para C1, C2, C3 y C4-S1. Estos archivos contienen el código a ejecutar para que Docker descargue la imagen de cada SO respectivamente. Además, como paso extra, se verifica que funcionen ejecutando el contenedor, lo que devuelve la consola de Ubuntu. Específicamente, se instala para cada cliente el 'openssh-client' y para el Servidor (C4/S1) 'openssh-client' y 'openssh-server'. Esto se hace con una serie de comandos: 'FROM ...' es para seleccionar el sistema base, 'ENV ...' es para evitar que apt pida la interacción del usuario, lo que resulta molesto en Docker, 'sed -i ...' es para cambiar los repositorios a 'old-releases.ubuntu.com' (si no, no es posible encontrar la imagen correcta),

y finalmente, la serie de comandos con 'apt' es para actualizar, limpiar y dejar listo el cliente. En el caso del C4-S1 o el cliente-servidor, es que se instala 'openssh-server' y distintos paquetes de utilidad para realizar las actividades. Cabe destacar que todos los códigos se encuentran en sus respectivas carpetas (C1, C2, C3 y C4-S1) en el repositorio de GitHub.

Comando de instalación (depende de cómo estén los directorios):
'docker build -t lab_c1:16.10 ./C1'

```
C1 > Dockerfile > ...
1  FROM ubuntu:16.10
2  ENV DEBIAN_FRONTEND=noninteractive
3  RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
4      sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list
5  RUN apt-get update && apt-get install -y --no-install-recommends \
6      openssh-client \
7      && apt-get clean && rm -rf /var/lib/apt/lists/*
8  CMD ["/bin/bash"]
```

Figura 3: Dockerfile para la instalación del cliente 1.

```
Documentos/Cripto/Lab5 took 49s
❯ docker run -it lab_c1:16.10
root@8e672853996c:/#
```

Figura 4: Contenedor con Ubuntu 16.10 ejecutado.

1.1.1. C2

Comando de instalación (depende de cómo estén los directorios):
'docker build -t lab_c2:18.10 ./C2'

```
C2 > Dockerfile > ...
1  FROM ubuntu:18.10
2  ENV DEBIAN_FRONTEND=noninteractive
3  RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
4      sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list
5  RUN apt-get update && apt-get install -y --no-install-recommends \
6      openssh-client \
7      && apt-get clean && rm -rf /var/lib/apt/lists/*
8  CMD ["/bin/bash"]
```

Figura 5: Dockerfile para la instalación del cliente 2.

```
Documentos/Cripto/Lab5 took 38s
❯ docker run -it lab_c2:18.10
root@af22647d09e8:/#
```

Figura 6: Contenedor con Ubuntu 18.10 ejecutado.

1.1.2. C3

Comando de instalación (depende de cómo estén los directorios):
'docker build -t lab_c3:20.10 ./C3'

```
C3 > Dockerfile > ...
1  FROM ubuntu:20.10
2  ENV DEBIAN_FRONTEND=noninteractive
3  RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
4      sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list
5  RUN apt-get update && apt-get install -y --no-install-recommends \
6      openssh-client \
7      && apt-get clean && rm -rf /var/lib/apt/lists/*
8  CMD ["/bin/bash"]
```

Figura 7: Dockerfile para la instalación del cliente 3.

```
Documentos/Cripto/Lab5 took 44s
❯ docker run -it lab_c3:20.10
root@42b1050358b9:/#
```

Figura 8: Contenedor con Ubuntu 20.10 ejecutado.

1.1.3. C4/S1

Comando de instalación (depende de cómo estén los directorios):
'docker build -t lab_c4:22.10 ./C4-S1'

```
C4-S1 > Dockerfile > ...
1  FROM ubuntu:22.10
2
3  ENV DEBIAN_FRONTEND=noninteractive
4
5  RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
6      sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list
7
8  RUN apt-get update && apt-get install -y --no-install-recommends \
9      openssh-server \
10     openssh-client \
11     && apt-get clean && rm -rf /var/lib/apt/lists/*
12
13 RUN useradd -m -s /bin/bash prueba && \
14     echo "prueba:prueba" | chpasswd
15
16 RUN sed -i 's/#PasswordAuthentication/PasswordAuthentication/' /etc/ssh/sshd_config && \
17     sed -i 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
18
19 RUN mkdir /var/run/sshd
20
21 EXPOSE 22
22
23 CMD ["/usr/sbin/sshd", "-D"]
```

Figura 9: Dockerfile para la instalación del cliente-servidor 4.

```
Documentos/Cripto/Lab5
❯ docker run -it --name C4-S1 lab_c4:22.10 bash

root@153f7f3b77ae:/#
```

Figura 10: Contenedor con Ubuntu 22.10 ejecutado.

1.2. Creación de las credenciales para S1

Para la creación de las credenciales para conectarse desde los distintos clientes al servidor, se empieza desde el mismo Dockerfile de la Figura 9, donde se le entrega como instrucciones crear el usuario, como se aprecia a continuación.

```
13  RUN useradd -m -s /bin/bash prueba && \
14      echo "prueba:prueba" | chpasswd
15
16  RUN sed -i 's/#PasswordAuthentication/PasswordAuthentication/' /etc/ssh/sshd_config && \
17      sed -i 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
18
19  RUN mkdir /var/run/sshd
20
21  EXPOSE 22
```

Figura 11: Captura del código del Dockerfile de C4-S1, donde se aprecia la creación del usuario.

Para verificar esto, nos conectamos al cliente-servidor desde bash con el comando (sujeto a cómo estén los directorios y nombre del proyecto) 'docker run -it –name C4-S1 lab_c4:22.10 bash'. Una vez dentro, verificamos que se haya creado el usuario 'prueba' con el comando 'id prueba' (ver en Figura 12), el cual nos retorna de existir información sobre el usuario. Esto nos dice que se aplicó correctamente el código del Dockerfile, por lo que no es necesario verificar que se haya aplicado la configuración de que se permita acceso SSH por autentificación mediante contraseña.

```
root@153f7f3b77ae:/# id prueba
uid=1000(prueba) gid=1000(prueba) groups=1000(prueba)
```

Figura 12: Captura de la bash al verificar si existe el usuario 'prueba'.

Posteriormente, se verifica si está escuchando el servicio SSH mediante el comando 'ps aux — grep ssh', el cual verifica 'sshd' (demonio del servidor SSH, que en pocas palabras es el programa que se ejecuta en la máquina para permitir las conexiones) sé esté ejecutando y escuchando. En este caso, no se estaba ejecutando, por lo que se procedió a activarlo con '/usr/sbin/sshd' y verificamos que quedó activo ejecutando 'ps aux — grep ssh' nuevamente,

como se ve en la Figura **??**. En este caso muestra que está en modo 'listener', lo que da por listo el proceso de verificación.

```
root@153f7f3b77ae:/# ps aux | grep ssh
root          22  0.0  0.0  14852  3428 ?        Ss   21:51
 0:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
```

Figura 13: Captura de la bash luego de activar sshd.

Finalmente, se procede a conectarse desde el cliente 1, quien lo hace con el comando 'ssh prueba@172.17.0.3' (la IP se consigue desde Docker, con el comando 'docker inspect -f ...', que es para ver la IP de los contenedores, ver Figura 14). Este proceso nos lleva a conectarnos mediante SSH usando la contraseña del usuario 'prueba'. Una vez completado el proceso, tenemos acceso a la bash como se aprecia en la Figura 15.

```
❯ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IP
Address}}{{end}}' C4-S1

172.17.0.3
```

Figura 14: Resultado al verificar la IP del contenedor servidor.

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permi
tted by
applicable law.

prueba@153f7f3b77ae:~$ funciono!
```

Figura 15: Captura de la bash luego de conectarse exitosamente al servidor.

El proceso es el mismo para los demás clientes, como se aprecia en las figs. 16, 17, 18.

```
To restore this content, you can run the 'unminimize' command
.
Last login: Sat Nov 15 21:55:21 2025 from 172.17.0.4
prueba@153f7f3b77ae:~$
```

Figura 16: Captura de la bash luego de conectarse exitosamente al servidor.

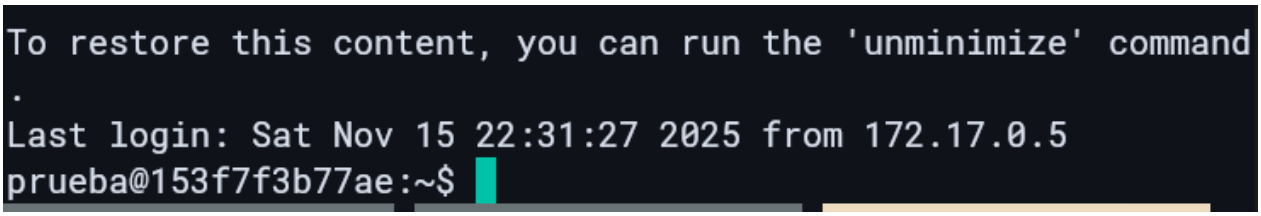


Figura 17: Captura de la bash luego de conectarse exitosamente al servidor.

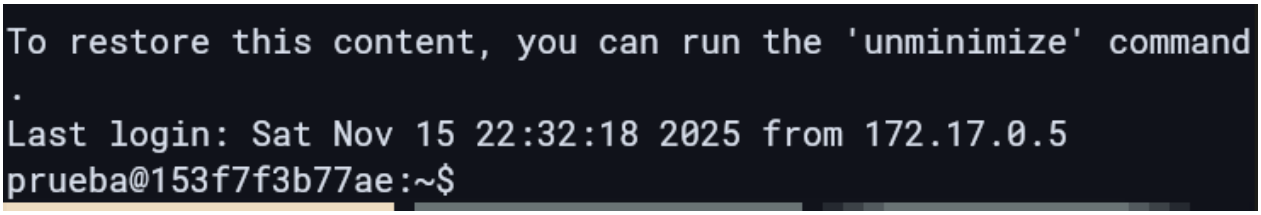


Figura 18: Captura de la bash luego de conectarse exitosamente al servidor.

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

Primero se tiene que HASSH es una técnica para identificar clientes y servidores SSH (como en nuestro caso) analizando los algoritmos anunciados en el saludo (Client Hello y Server Hello). Para esto se usa un código en Python que extrae los datos desde una captura formato .pcap y nos entrega los HASSH del cliente y servidor (HASSH. El procedimiento es simple, se inicia la captura desde el servidor con tshark (terminal wireshark para capturar tráfico en la red, se instala realizando los comandos 'apg-get update' y 'apt-get install -y –no-install-recommends tcpdump') mediante el comando 'tcpdump -i any port 22 -w c1_ssh.pcap' (ver Figura 19. Esto escucha todas las interfaces de red en el puerto 22 y guarda el resultado en el archivo 'c1_ssh.pcap'. Este archivo es procesado luego por el código del repositorio de HASSH para obtener el hash tanto del cliente como del servidor.

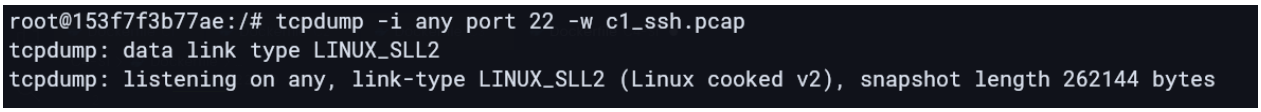


Figura 19: Captura de con tshark del tráfico de C1.

Ahora, luego de explicar todo, se inicia con el cliente C1 (IP 172.17.0.4), conectándolo mediante SSH al servidor S1 (IP 172.17.0.3) utilizando el puerto TCP 22. Durante el establecimiento de la conexión SSH se identifica el flujo de intercambio de claves ('SSH_MSG_KEXINIT'), en el cual se negocian algoritmos criptográficos. La captura se encuentra en el repositorio, y se puede apreciar en la Figura 20, mientras que el HASSH generado en la Figura 21.

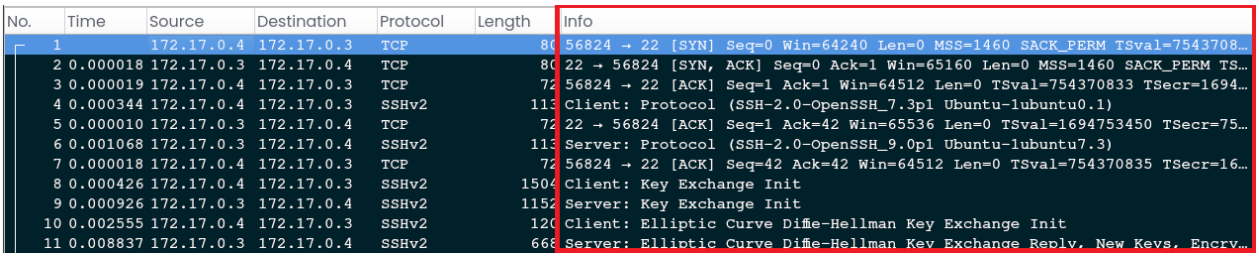


Figura 20: Captura resultante del tráfico de C1 con el servidor.

Figura 21: HASSH generado para la captura de C1.

Aquí se muestra la huella digital del tráfico SSH generado entre un cliente y un servidor, incluyendo los tamaños de los paquetes intercambiados (1504 bytes del cliente al servidor y 1152 bytes del servidor al cliente), junto con los valores HASSH que identifican de manera única las configuraciones criptográficas usadas en cada extremo. También se tiene la cadena de identificación de ambas versiones de OpenSSH empleadas, y los algoritmos de intercambio de claves (KEX), cifrado (Cipher), autenticación MAC y compresión que el cliente ofrece para negociar la conexión segura, lo que permite diferenciar versiones y configuraciones de SSH al comparar con las versiones más nuevas.

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1504 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **HASSH Cliente:** 0e4584cb9f2dd077dbf8ba0df8112d8e
- **HASSH Servidor:** a984ff804585fabe3cd08f4b3849024a
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1

- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512, diffie-hellman-group18-sha512, diffie-hellman-group-exchange-sha1, diffie-hellman-group14-sha256, diffie-hellman-group14-sha1, ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr, aes128-gcm@openssh.com, aes256-gcm@openssh.com, aes128-cbc, aes192-cbc, aes256-cbc, 3des-cbc
- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com, zlib

Este mismo proceso se realiza para cada parte.

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

Se inicia la captura, como se muestra en la siguiente Figura.

```
root@153f7f3b77ae:/# tcpdump -i any port 22 -w c2_ssh.pcap
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
```

Figura 22: Captura de con tshark del tráfico de C2.

Y se obtienen los siguientes resultados, como se aprecia en las figs. 23, 24.

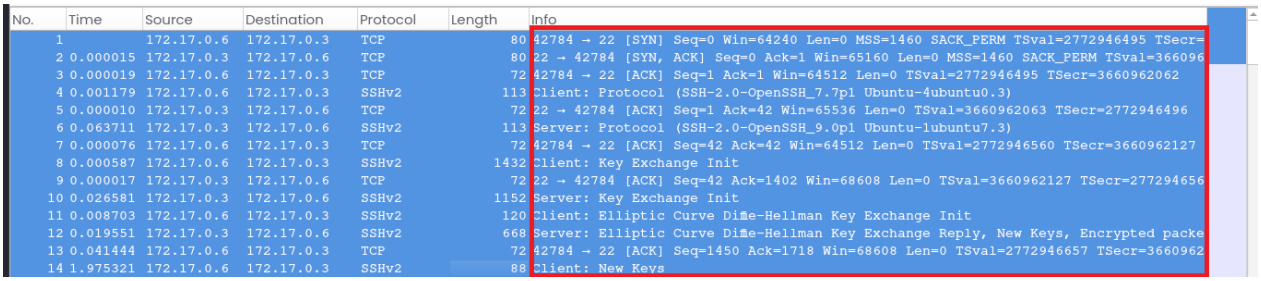


Figura 23: Captura resultante del tráfico de C2 con el servidor.

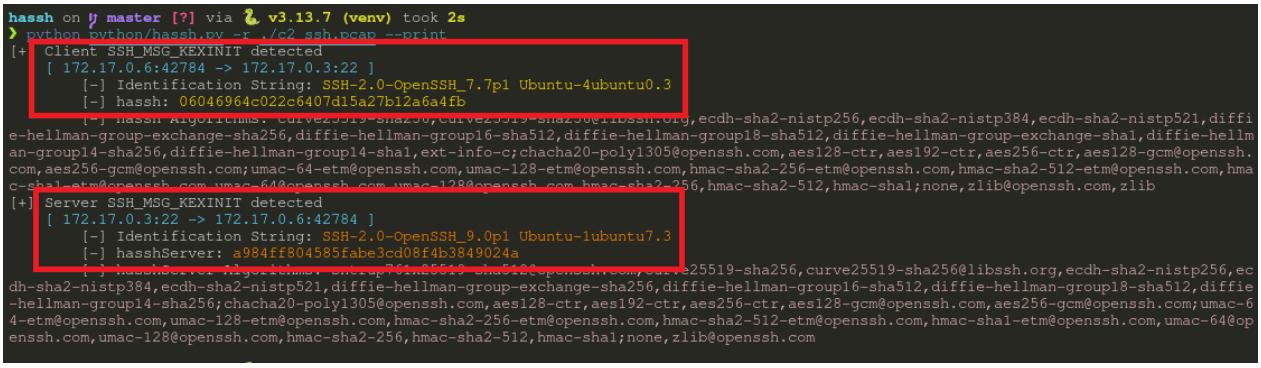


Figura 24: HASSH generado para la captura de C2.

La captura muestra el inicio del intercambio criptográfico SSH entre C2 (OpenSSH 7.7p1) y el servidor (OpenSSH 9.0p1), donde ambos envían el mensaje 'SSH_MSG_KEXINIT' anunciando sus algoritmos soportados para negociar una conexión segura. A partir de estos parámetros se generaron las huellas HASSH del cliente (06046964c022c6407d15a27b12a6a4fb) y del servidor (a984ff804585fabe3cd08f4b3849024a), que actúan como identificadores únicos de sus configuraciones criptográficas.

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1432 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **HASSH Cliente:** 06046964c022c6407d15a27b12a6a4fb
- **HASSH Servidor:** a984ff804585fabe3cd08f4b3849024a
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_7.7p1 Ubuntu-1ubuntu0.3
- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3

- **Algoritmos del Cliente KEX:** curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512, diffie-hellman-group18-sha512, diffie-hellman-group14-sha256, ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr, aes128-gcm@openssh.com, aes256-gcm@openssh.com
- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com, zlib

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

Se inicia la captura, como se muestra en la siguiente Figura.

```
root@153f7f3b77ae:/# tcpdump -i any port 22 -w c3_ssh.pcap
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
```

Figura 25: Captura de con tshark del tráfico de C3.

Y se obtienen los siguientes resultados, como se aprecia en las figs. 26, 27.

No.	Time	Source	Destination	Protocol	Length	Info
1		172.17.0.7	172.17.0.3	TCP	80	39926 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=388484366 TSecr=0
2	0.000018	172.17.0.3	172.17.0.7	TCP	80	22 → 39926 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=376163
3	0.000024	172.17.0.7	172.17.0.3	TCP	72	39926 → 22 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=388484366 TSecr=3761636246
4	0.001620	172.17.0.7	172.17.0.3	SSHv2	113	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
5	0.000013	172.17.0.3	172.17.0.7	TCP	72	22 → 39926 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=3761636248 TSecr=388484368
6	0.024365	172.17.0.3	172.17.0.7	SSHv2	113	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7	0.000047	172.17.0.7	172.17.0.3	TCP	72	39926 → 22 [ACK] Seq=42 Ack=42 Win=64512 Len=0 TSval=388484392 TSecr=3761636272
8	0.000204	172.17.0.7	172.17.0.3	SSHv2	1584	Client: Key Exchange Init
9	0.000009	172.17.0.3	172.17.0.7	TCP	72	22 → 39926 [ACK] Seq=42 Ack=1554 Win=68608 Len=0 TSval=3761636272 TSecr=388484392
10	0.009901	172.17.0.3	172.17.0.7	SSHv2	1152	Server: Key Exchange Init
11	0.002774	172.17.0.7	172.17.0.3	SSHv2	120	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
12	0.006135	172.17.0.3	172.17.0.7	SSHv2	668	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packe
13	0.040660	172.17.0.7	172.17.0.3	TCP	72	39926 → 22 [ACK] Seq=1602 Ack=1718 Win=68608 Len=0 TSval=388484452 TSecr=37616362
14	1.952614	172.17.0.7	172.17.0.3	SSHv2	88	Client: New Keys

Figura 26: Captura resultante del tráfico de C3 con el servidor.

Figura 27: HASSH generado para la captura de C3.

En este escenario, el cliente utiliza OpenSSH_8.3p1 y el servidor mantiene la versión OpenSSH_9.0p1, iniciando el intercambio criptográfico mediante el mensaje 'SSH_MSG_KEXINIT'. El cliente anuncia un conjunto de algoritmos muy similar al del escenario anterior, pero con una huella HASSH distinta (ae8bd7dd09970555aa4c6ed22adbbf56), lo que permite diferenciarlo como una variante específica de configuración criptográfica. El servidor conserva la misma huella HASSH (a984ff804585fabe3cd08f4b3849024a).

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1584 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **HASSH Cliente:** ae8bd7dd09970555aa4c6ed22adbbf56
- **HASSH Servidor:** a984ff804585fabe3cd08f4b3849024a
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1
- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512, diffie-hellman-group18-sha512, diffie-hellman-group14-sha256, ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr, aes128-gcm@openssh.com, aes256-gcm@openssh.com

- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com, zlib

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

Se inicia la captura, como se muestra en la siguiente Figura.

```
root@153f7f3b77ae:/# tcpdump -i any port 22 -w c4_ssh.pcap &
[1] 436
root@153f7f3b77ae:/# tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
```

Figura 28: Captura de con tshark del tráfico de C4.

Y se obtienen los siguientes resultados, como se aprecia en las figs. 29, 30.

No.	Time	Source	Destination	Protocol	Length	Info
1		172.17.0.3	172.17.0.3	TCP	80	55790 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=707033488 TSecr=(
2	0.000014	172.17.0.3	172.17.0.3	TCP	80	22 → 55790 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=70703
3	0.000013	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=707033488 TSecr=707033488
4	0.000337	172.17.0.3	172.17.0.3	SSHv2	113	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
5	0.000007	172.17.0.3	172.17.0.3	TCP	72	22 → 55790 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=707033488 TSecr=707033488
6	0.015519	172.17.0.3	172.17.0.3	SSHv2	113	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7	0.000014	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSval=707033504 TSecr=707033504
8	0.000528	172.17.0.3	172.17.0.3	SSHv2	1576	Client: Key Exchange Init
9	0.002628	172.17.0.3	172.17.0.3	SSHv2	1152	Server: Key Exchange Init
10	0.040538	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=1546 Ack=1122 Win=68096 Len=0 TSval=707033548 TSecr=70703350
11	0.082345	172.17.0.3	172.17.0.3	SSHv2	1280	Client: Diffie-Hellman Key Exchange Init
12	0.019593	172.17.0.3	172.17.0.3	SSHv2	1636	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
13	0.000012	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=2754 Ack=2686 Win=79360 Len=0 TSval=707033649 TSecr=70703364
14	0.041820	172.17.0.3	172.17.0.3	SSHv2	88	Client: New Keys

Figura 29: Captura resultante del tráfico de C4 con el servidor.

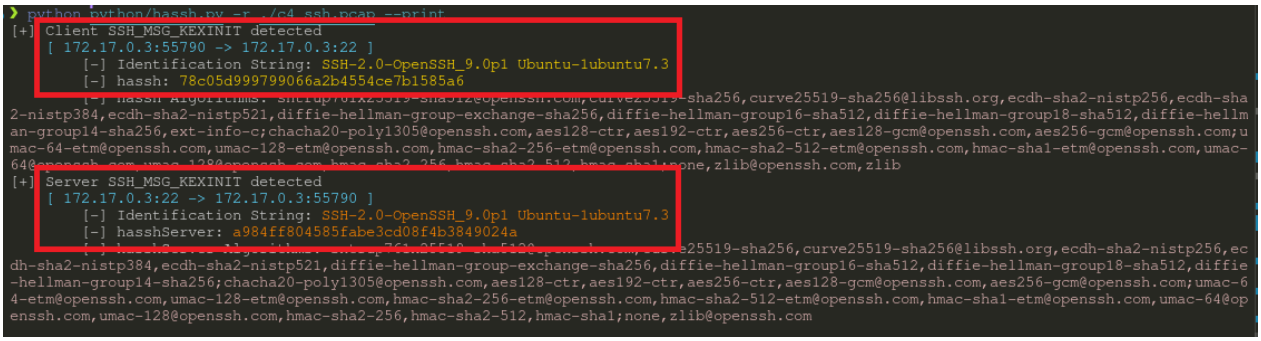


Figura 30: HASSH generado para la captura de C4.

El escenario C4 corresponde a una conexión SSH en la que tanto el cliente como el servidor utilizan OpenSSH 9.0p1 en Ubuntu, lo que genera un patrón de negociación completamente alineado entre ambos extremos, ya que son localmente iguales (mismo contenendor).

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1576 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **HASSH Cliente:** 78c05d999799066a2b4554ce7b1585a6
- **HASSH Servidor:** a984ff804585fabe3cd08f4b3849024a
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** sntrup761x25519-sha512@openssh.com, curve25519-sha256,

  curve25519-sha256@libssh.org, ecdh-sha2-nistp256,

  ecdh-sha2-nistp384, ecdh-sha2-nistp521,

  diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512,

  diffie-hellman-group18-sha512, diffie-hellman-group14-sha256,

  ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr,

  aes128-gcm@openssh.com, aes256-gcm@openssh.com

- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com, zlib

1.7. Compara la versión de HASSH obtenida con la base de datos para validar si el cliente corresponde al mismo

Se revisa en la sección de 'hashGen' del repositorio de HASSH (específicamente esta parte), donde se ven las posibles coincidencias con los clientes y servidor. Se tienen las imágenes de Ubuntu con versiones 16.10, 18.10, 20.10 y 22.10 para C1, C2, C3 y C4-S1 respectivamente. En este caso, únicamente se tienen datos para la versión 18.10, que es la única que calza dentro de las versiones disponibles. Los resultados son que para esta versión el HASSH esperado es:

- **sshClient:**openssh-client
- **sshClientVersion:** 1:7.7p1-4
- **clientIdentificationString:** SSH-2.0-OpenSSH_7.7p1 Ubuntu-4,
- **hassh**: 06046964c022c6407d15a27b12a6a4fb
- **hasshAlgorithms**: curve25519-sha256,curve25519-sha256@libssh.org, ...

Las versiones de Ubuntu 16.10, 20.10 y 22.10 no calzan con los valores HASSH registrados en la base de datos debido a que utilizan compilaciones y configuraciones de OpenSSH que no son iguales en el conjunto y orden de algoritmos anunciados durante el intercambio 'SSH_MSG_KEXINIT', lo cual afecta la generación del hash. Estas variaciones se explican por diferencias en paquetes, parches de seguridad, librerías criptográficas e incluso ajustes en los archivos de configuración de SSH presentes en cada versión del sistema. En cambio, la versión Ubuntu 18.10 sí coincide porque emplea una compilación y una lista de algoritmos que corresponden con la huella registrada en la base de datos HASSH, permitiendo una coincidencia exacta entre los valores obtenidos y los existentes en el repositorio de fingerprints.

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

Esto ya se hizo en secciones anteriores. Durante la captura de la sesión SSH se identificaron paquetes en texto plano correspondientes al intercambio inicial del protocolo antes de que se estableciera el canal cifrado. Estos paquetes contienen la cadena de identificación del software SSH utilizado por el cliente y el servidor, junto con las listas completas de algoritmos de intercambio de claves (KEX), cifrado, integridad (MAC) y compresión anunciados en los mensajes 'SSH_MSG_KEXINIT'. Esta información constituye la base para la huella criptográfica HASSH/HASSHServer, ya que se encuentra visible antes de que se produzca el intercambio de claves y el tráfico posterior que se ha cifrado.

1.8.1. C1

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1504 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1
- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521,
  diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512,
  diffie-hellman-group18-sha512, diffie-hellman-group-exchange-sha1,
  diffie-hellman-group14-sha256, diffie-hellman-group14-sha1,
  ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr,
  aes128-gcm@openssh.com, aes256-gcm@openssh.com,
  aes128-cbc, aes192-cbc,
  aes256-cbc, 3des-cbc
- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com,
  hmac-sha1-etm@openssh.com, umac-64@openssh.com,

umac-128@openssh.com, hmac-sha2-256,
hmac-sha2-512, hmac-sha1

- **Algoritmos del Cliente Compression:** none, zlib@openssh.com,
  zlib

1.8.2. C2

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1432 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_7.7p1 Ubuntu-1ubuntu0.3
- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** curve25519-sha256, curve25519-sha256@libssh.org,
  ecdh-sha2-nistp256, ecdh-sha2-nistp384,
  ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256,
  diffie-hellman-group16-sha512, diffie-hellman-group18-sha512,
  diffie-hellman-group14-sha256, ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr,
  aes192-ctr, aes256-ctr, aes128-gcm@openssh.com,
  aes256-gcm@openssh.com
- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com,
  hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com,
  hmac-sha1-etm@openssh.com, umac-64@openssh.com,
  umac-128@openssh.com, hmac-sha2-256,
  hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com,
  zlib

1.8.3. C3

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1584 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1
- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512, diffie-hellman-group18-sha512, diffie-hellman-group14-sha256, ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr, aes128-gcm@openssh.com, aes256-gcm@openssh.com
- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com, zlib

1.8.4. C4/S1

- **Tamaño Cliente-Servidor Protocolo SSHv2:** 113 bytes
- **Tamaño Servidor-Cliente Protocolo SSHv2:** 113 bytes
- **Tamaño Cliente-Servidor:** 1576 bytes
- **Tamaño Servidor-Cliente:** 1152 bytes
- **Identificación SSH Cliente:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3

- **Identificación SSH Servidor:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Algoritmos del Cliente KEX:** sntrup761x25519-sha512@openssh.com, curve25519-sha256,
  curve25519-sha256@libssh.org, ecdh-sha2-nistp256,
  ecdh-sha2-nistp384, ecdh-sha2-nistp521,
  diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512,
  diffie-hellman-group18-sha512, diffie-hellman-group14-sha256,
  ext-info-c
- **Algoritmos del Cliente Cipher:** chacha20-poly1305@openssh.com, aes128-ctr,
  aes192-ctr, aes256-ctr,
  aes128-gcm@openssh.com, aes256-gcm@openssh.com
- **Algoritmos del Cliente MAC:** umac-64-etm@openssh.com, umac-128-etm@openssh.com,
  hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com,
  hmac-sha1-etm@openssh.com, umac-64@openssh.com,
  umac-128@openssh.com, hmac-sha2-256,
  hmac-sha2-512, hmac-sha1
- **Algoritmos del Cliente Compression:** none, zlib@openssh.com,
  zlib

1.9. Diferencia entre C1 y C2

La principal diferencia entre las capturas C1 y C2 radica en las versiones de OpenSSH usadas por el cliente, lo cual impacta tanto en el tamaño de los paquetes como en los algoritmos de intercambio de claves ofrecidos. En C1, el cliente utiliza OpenSSH 7.3p1, mientras que en C2 emplea OpenSSH 7.7p1, una versión más reciente que introduce ajustes en el conjunto de algoritmos KEX y Ciphers. Esto se refleja en una reducción del tamaño total del flujo Cliente-Servidor en C2 (1432 bytes) respecto de C1 (1504 bytes), aunque ambos mantienen el mismo tamaño en los mensajes iniciales SSHv2 de intercambio de identificación (113 bytes por dirección). Además, C1 incluye un listado más extenso de algoritmos de cifrado, incluyendo opciones CBC y 3DES, mientras que C2 ofrece un conjunto más depurado y moderno centrado en algoritmos basados en CTR y GCM. Estas diferencias son consistentes con la evolución de OpenSSH, que a partir de versiones posteriores elimina algoritmos criptográficos considerados obsoletos o menos seguros, lo que explica la variación observada entre ambas capturas.

1.10. Diferencia entre C2 y C3

La diferencia entre C2 y C3 se encuentra también en la versión del cliente SSH utilizado, donde C2 opera con OpenSSH 7.7p1, mientras que C3 utiliza una versión más reciente, OpenSSH 8.3p1, lo que se ve en el aumento del tamaño total de los mensajes Cliente-Servidor (1432 bytes en C2 frente a 1584 bytes en C3), aunque ambos mantienen idéntico tamaño en el intercambio inicial SSHv2 (113 bytes por dirección). A nivel criptográfico, ambos clientes presentan prácticamente el mismo conjunto de algoritmos KEX, Ciphers, MAC y Compresión, por lo que el incremento en C3 se asocia principalmente a cambios internos en la estructura del mensaje y metadatos propios de la versión más reciente de OpenSSH. En consecuencia, C3 no introduce diferencias significativas en las propuestas criptográficas respecto de C2, pero sí evidencia una mayor carga inicial en el flujo de negociación, consistente con el avance de la implementación SSH en Ubuntu y la evolución del protocolo hacia configuraciones más extensibles y descriptivas.

1.11. Diferencia entre C3 y C4

La diferencia entre los escenarios C3 y C4/S1 radica en la versión del cliente SSH. Mientras C3 utiliza OpenSSH 8.3p1, el escenario C4/S1 emplea la versión más reciente OpenSSH 9.0p1, lo que introduce cambios directos en la negociación criptográfica. En particular, se incorpora un nuevo algoritmo de intercambio de claves post-cuántico, sntrup761x25519-sha512@openssh.com

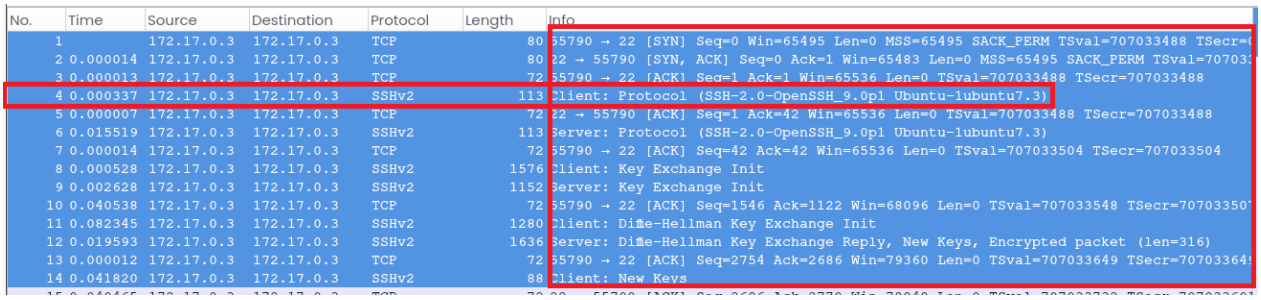
2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión “?”

Con la información de la captura (cadena de identificación SSH-2.0-OpenSSH, KEXINIT cliente grande 1578 bytes, ECDH Key Exchange, luego New Keys y paquetes cifrados) no es posible afirmar con absoluta certeza la versión exacta solo por la descripción. Sin embargo, por el tamaño del KEXINIT y el patrón (ECDH, lista moderna de ciphers/MACs y compresión), el tráfico es consistente con un cliente OpenSSH moderno (por ejemplo OpenSSH 8.x–9.x), es decir, coincide más con C3 (OpenSSH_8.3p1) o con C4 (OpenSSH_9.0p1). Para decidir entre 8.3 y 9.0 hay un punto clave: si el KEXINIT coincide en su largo. El valor más cercano es el de C4, con 1576 bytes, por lo que se estima que el tráfico es proveniente de un cliente con versión OpenSSH_9.0p1.

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

Para recrear este tráfico, se comienza levantando el contenedor de C4-S1. Se prende el demonio SSH con ’/usr/sbin/sshd’ y se instala tcpdump para la captura, con ’apt-get install -y –no-install-recommends tcpdump’. Posteriormente, se inicia la captura en segundo plano de tcpdump con ’tcpdump -i any port 22 -w c4_ssh_recreate.pcap &’ y se conecta al cliente con la IP del contenedor servidor. Esta IP se obtiene con ’docker inspect -f’range .Network-Settings.Networks.IPAdressend’ C4-S1’, y utilizando esta IP se conecta con el comando ’ssh prueba@¡IP¿’. Finalmente, se sale de la conexión, se baja el proceso de tcpdump deteniendo el proceso con ’kill 1 % ¡proceso-id¿’ y se obtiene la captura.



No.	Time	Source	Destination	Protocol	Length	Info
1		172.17.0.3	172.17.0.3	TCP	80	55790 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=707033488 TSecr=(
2	0.000014	172.17.0.3	172.17.0.3	TCP	80	22 → 55790 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=70703
3	0.000013	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=707033488 TSecr=707033488
4	0.000337	172.17.0.3	172.17.0.3	SSHv2	113	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
5	0.000007	172.17.0.3	172.17.0.3	TCP	72	22 → 55790 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=707033488 TSecr=707033488
6	0.015519	172.17.0.3	172.17.0.3	SSHv2	113	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7	0.000014	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSval=707033504 TSecr=707033504
8	0.000528	172.17.0.3	172.17.0.3	SSHv2	1576	Client: Key Exchange Init
9	0.002628	172.17.0.3	172.17.0.3	SSHv2	1152	Server: Key Exchange Init
10	0.040538	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=1546 Ack=1122 Win=68096 Len=0 TSval=707033548 TSecr=70703350
11	0.082345	172.17.0.3	172.17.0.3	SSHv2	1280	Client: Diffie-Hellman Key Exchange Init
12	0.019593	172.17.0.3	172.17.0.3	SSHv2	1636	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
13	0.000012	172.17.0.3	172.17.0.3	TCP	72	55790 → 22 [ACK] Seq=2754 Ack=2686 Win=79360 Len=0 TSval=707033649 TSecr=70703364
14	0.041820	172.17.0.3	172.17.0.3	SSHv2	88	Client: New Keys

Figura 31: Captura resultante del tráfico de C4 con el servidor.

Se puede apreciar que se tienen largos similares. Sin toda la información de paquetes, algoritmos y demás, es imposible saber a ciencia cierta qué versión de protocolo se está usando.

3. Desarrollo (Parte 3)

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

Reducir el tamaño del Key Exchange Init (KEXINIT) del servidor SSH a menos de 300 bytes es posible, pero implica reducir la lista de algoritmos que el servidor anuncia, lo cual puede resultar en brechas de seguridad o problemas de conexión con los clientes. Este paquete suele ser grande (aproximadamente 1500 bytes de las capturas de los clientes) porque incluye muchos algoritmos soportados para cambios de llave, compresión, MACs, etc. Se repite el procedimiento del paso anterior. Para recrear este tráfico, se comienza levantando el contenedor de C4-S1. Se prende el demonio SSH con '/usr/sbin/sshd' y se instala tcpdump para la captura y nano para la edición de archivos, con 'apt-get install -y –no-install-recommends tcpdump nano'. Ahora, para reducir el largo, se debe configurar 'sshd_config' para que solo anuncie uno o muy pocos algoritmos por categoría. Se accede a la este archivo mediante el comando 'nano /etc/ssh/sshd_config' y se modifica, como se aprecia en la siguiente Figura.

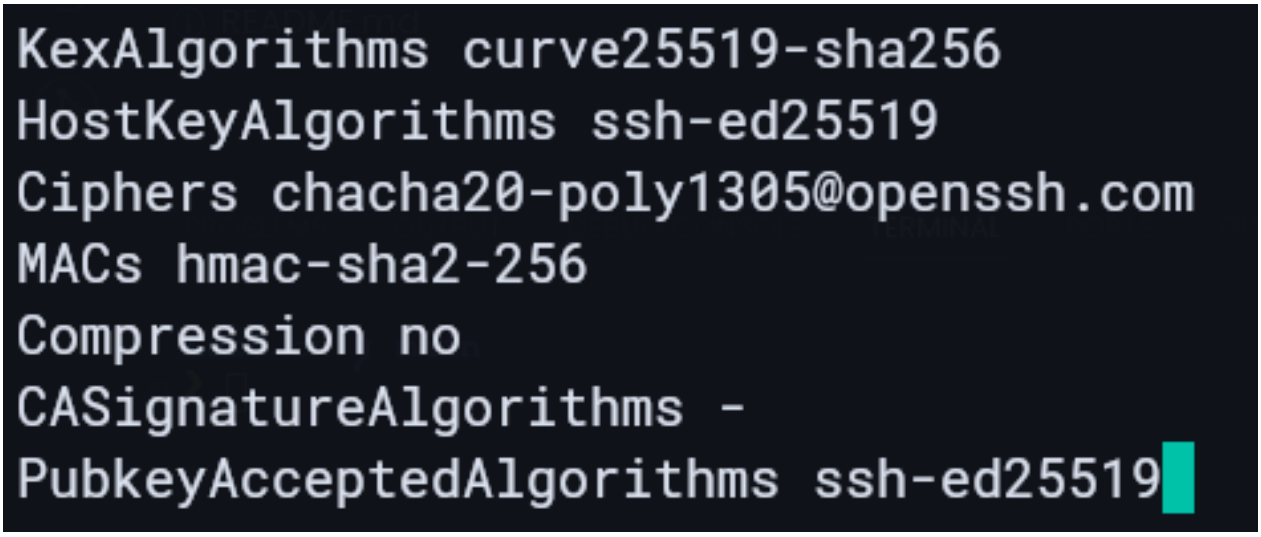


Figura 32: Archivo del servidor SSH editado.

Esto dejará el KEXINIT extremadamente pequeño, probablemente, ya que se le están dando pocos algoritmos a utilizar. Posteriormente, se necesita reiniciar SSH con service ssh restart y empezar la captura en segundo plano de tcpdump con 'tcpdump -i any port 22 -w c4_ssh_recreate.pcap &'. Se conecta al cliente con la IP del contenedor servidor. Esta IP se obtiene con 'docker inspect -f'range .NetworkSettings.Networks.IPAdressend' C4-S1', y utilizando esta IP se conecta con el comando 'ssh prueba@¡IP¿'. Finalmente, se sale de la conexión, se baja el proceso de tcpdump deteniendo el proceso con 'kill 1 % ¡proceso-id¿' y se obtiene la captura. Los resultados se aprecian en la Figura siguiente.

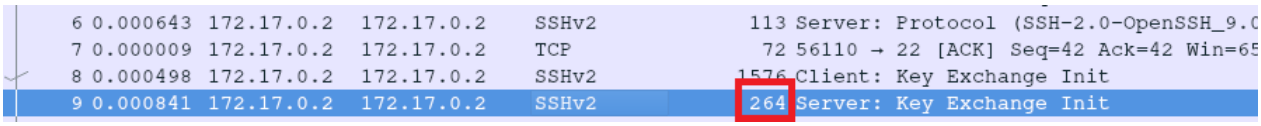


Figura 33: Captura del tráfico con el KEI modificado.

Se aprecia que se tiene un KEI de 264 bytes de largo, lo cual es diferente al 1572 que se tenía en la misma versión. Esto refleja los cambios hechos en el archivo de configuración de SSHD, donde se modificó la cantidad de algoritmos a usar.

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es una serie de herramientas que permiten implementar el protocolo SSH para establecer canales seguros (cifrados) entre cliente y servidor. Esto es su objetivo principal, reemplazando tecnologías antiguas y siendo un método seguro para el transporte de datos. En un entorno SSH, el cliente inicia la conexión con TCP y se intercambian los identificadores y los algoritmos criptográficos de ambas partes, a modo de acordar cuáles se van a usar. Este proceso es el que se analizó durante este laboratorio, el cual puede brindar vulnerabilidades si se explota.

4.2. Capas de Seguridad en OpenSSH

El tráfico SSH cumple los principios de seguridad de la información. Estos son confidencialidad, con criptografía simétrica luego del cambio de claves. Integridad, con algoritmos MAC, quienes verifican la información y que no haya sido modificada. La autenticidad se cumple con la validación de claves del servidor, permitiendo que no se acepten intercambios y conexiones seguras. Disponibilidad con el protocolo TCP, que no es propio de SSH, pero permite todo el proceso de retransmisión de paquetes y flujo. Finalmente, No repudio se cumple a medias, ya que existen mecanismos de autenticación, pero sin firmas digitales de terceros para verificar. Disponibilidad: aunque SSH no la garantiza por sí mismo, el protocolo funciona sobre TCP, lo que permite retransmisión de paquetes, control de flujo y sesiones estables, salvo ataques de denegación de servicios externos.

4.3. Identificación de que protocolos no se cumplen

Se tiene que SSH no cumple con la disponibilidad. OpenSSH no incorpora mecanismos que garanticen que el servicio permanezca accesible frente a interrupciones maliciosas. Por otra parte, el no repudio no se cumple, ya que es posible autenticar usuarios mediante claves o credenciales. OpenSSH no implementa un esquema de firma digital verificable por un tercero.

Conclusiones y comentarios

A partir del análisis realizado sobre el tráfico SSH capturado y las distintas versiones del cliente, se tiene que OpenSSH integra múltiples mecanismos de seguridad orientados a proteger la comunicación entre cliente y servidor. Para esto, se emplea un establecimiento de la conexión, junto al intercambio de claves y el uso de cifrado simétrico y MAC. Esto permite verificar en la práctica los principios de confidencialidad e integridad, ya que, luego de ver un mensaje New Keys, el contenido de los paquetes dejó de ser legible, lo que confirma que la información viaja protegida frente a un atacante que pudiera interceptarla.

Del mismo modo, los procesos de identificación del cliente y del servidor apoyan los principios de autenticidad y no repudio, aunque este último puede ser discutible dependiendo del mecanismo de llaves y la gestión de identidades que utilice el sistema. Sin embargo, se tiene que OpenSSH ofrece un diseño robusto que apunta también a la disponibilidad, salvo que existieran ataques externos (como DoS) que no fueron objeto de estudio en este laboratorio.

En conclusión, la evidencia obtenida en el tráfico capturado refleja que OpenSSH cumple de forma consistente con sus objetivos de seguridad, y que las variaciones entre versiones afectan directamente la huella criptográfica del cliente, lo que permite diferenciar implementaciones sin inspeccionar contenido protegido, reforzando la utilidad del fingerprinting mediante HASSH.