Informe Laboratorio 5

Sección 2

Benjamin Polanco
e-mail: benjamin.polanco@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:

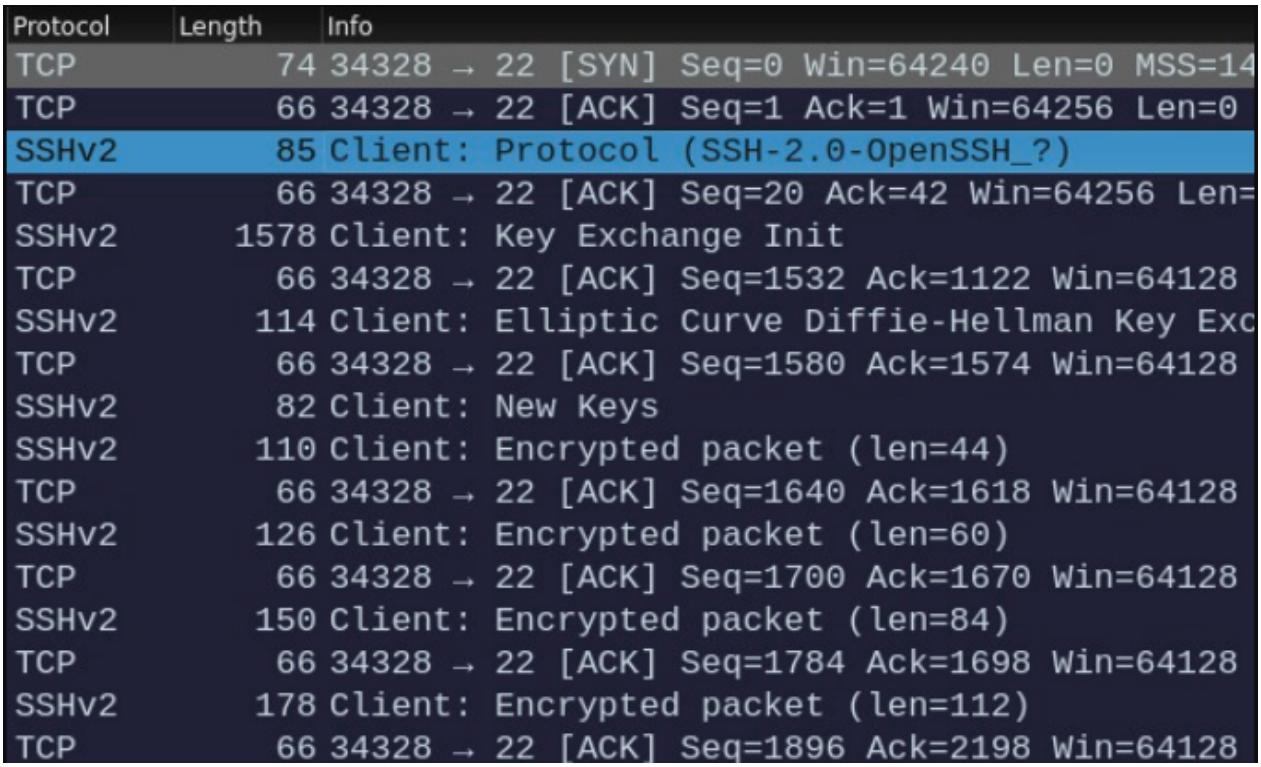
- Crear 4 contenedores en Docker por medio de un DockerFile, donde cada uno tendrá el siguiente SO: Ubuntu 16.10, Ubuntu 18.10, Ubuntu 20.10 y Ubuntu 22.10 a los cuales se llamarán C1, C2, C3 y C4 respectivamente.
  El equipo con Ubuntu 22.10 también será utilizado como S1.
- Para cada uno de ellos,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

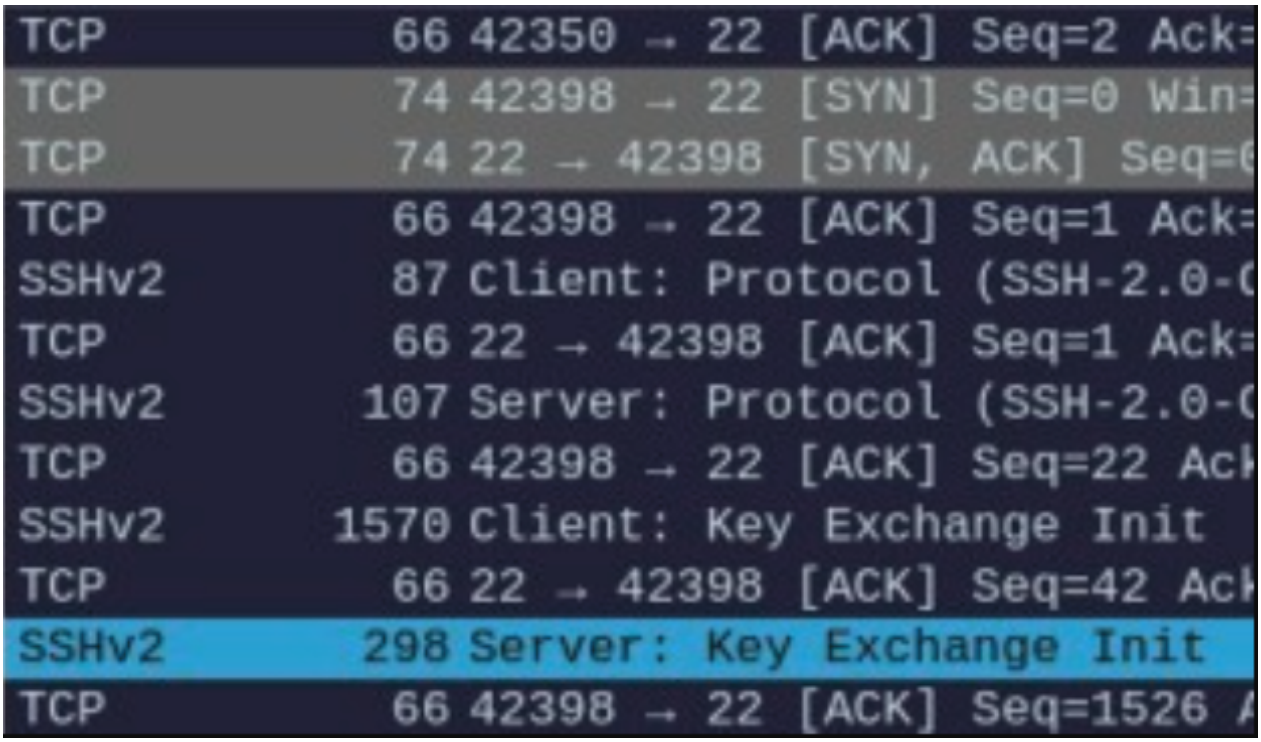


Figura 2: Captura del Key Exchange

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

A continuación se presentan los archivos Dockerfile utilizados para la creación de los contenedores. Dado que las versiones de Ubuntu solicitadas (16.10 a 22.10) se encuentran en estado *End of Life* (EOL), fue necesario modificar los repositorios en todos los casos apuntando a `old-releases.ubuntu.com` para permitir la instalación de paquetes.

1.1.1. C1

Este contenedor actúa únicamente como cliente SSH legado. Se instalan las herramientas de red básicas y el cliente OpenSSH.

```
FROM ubuntu:16.10

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

RUN apt-get update && apt-get install -y openssh-client tcpdump net-tools
    iputils-ping
```

Listing 1: Dockerfile para C1

1.1.2. C2

Similar al caso anterior, se configura el cliente sobre la versión 18.10 modificando las fuentes de apt para evitar errores de conexión a los repositorios archivados.

```
FROM ubuntu:18.10

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

RUN apt-get update && apt-get install -y openssh-client tcpdump net-tools
    iputils-ping
```

Listing 2: Dockerfile para C2

1.1.3. C3

Se despliega la versión 20.10 con las mismas herramientas de diagnóstico de red (`tcpdump`, `net-tools`) para realizar las capturas de tráfico solicitadas.

```
FROM ubuntu:20.10

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

RUN apt-get update && apt-get install -y openssh-client tcpdump net-tools
    iputils-ping
```

Listing 3: Dockerfile para C3

1.1.4. C4/S1

ste contenedor cumple el doble rol de Cliente 4 y Servidor 1[cite: 18]. Además de modificar los repositorios, se instala `openssh-server`, se crea el usuario "prueba"solicitado, y se habilita la autenticación por contraseña en la configuración del demonio SSH.

```
FROM ubuntu:22.10
```

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

RUN apt-get update && apt-get install -y openssh-client openssh-server
    tcpdump net-tools iputils-ping nano

RUN mkdir -p /var/run/sshd

RUN useradd -m -s /bin/bash prueba && echo 'prueba:prueba' | chpasswd

RUN sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /
    etc/ssh/sshd_config

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Listing 4: Dockerfile para S1/C4

1.2. Creación de las credenciales para S1

Para lograr el objetivo de establecer credenciales válidas en el servidor S1 y garantizar el acceso mediante contraseña, se implementó directamente en la definición de la imagen del contenedor (archivo `Dockerfile`).

El procedimiento realizado constó de tres pasos técnicos fundamentales ejecutados durante la construcción del contenedor:

1. **Creación del usuario:** Se utilizó el comando `useradd` con la bandera `-m` para generar el directorio *home* del usuario y `-s` para asignar `/bin/bash` como su intérprete de comandos por defecto.

2. **Asignación de contraseña:** Se empleó el comando `chpasswd` mediante una tubería (pipe) para asignar la contraseña "prueba" al usuario "prueba" de forma no interactiva.

3. **Configuración del demonio SSH:** Se modificó el archivo de configuración `/etc/ssh/sshd_config` utilizando el editor de flujo `sed`. Esto fue necesario para cambiar la directiva `PasswordAuthentication` a `yes`, ya que las versiones recientes de Ubuntu suelen deshabilitar el acceso por contraseña por defecto.

Para verificar que el procedimiento fue exitoso, se realizó una conexión de prueba desde el cliente C1 hacia S1. Como se observa en la Figura **??**, el servidor solicitó la contraseña y permitió el acceso al sistema, confirmando la correcta creación de las credenciales.

```
➜ Lab5 git:(main) X docker compose exec c3 ssh prueba@S1
WARN[0000] /home/demau/dev/criptografia-seguridad-en-redes/Lab5/docker-compose.y
 it will be ignored, please remove it to avoid potential confusion
The authenticity of host 's1 (172.18.0.4)' can't be established.
ECDSA key fingerprint is SHA256:aTYO02yUY0OemsIbl3U+uaE2vyQ2bfG9fPhBMK/h+TM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 's1,172.18.0.4' (ECDSA) to the list of known hosts.
prueba@s1's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.6.87.2-microsoft-standard-WSL2 x86_64)
```

Figura 3: Acceso SSH exitoso al servidor S1 con el usuario prueba.

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

Se procedió a realizar la captura de tráfico generado por el contenedor C1 al conectarse con S1. Al aplicar el filtro de visualización `ssh` en Wireshark, se aisló el flujo de la comunicación, permitiendo identificar las fases iniciales de la conexión, incluyendo el saludo TCP y la negociación de versiones. El flujo general de paquetes capturados se presenta a continuación:

ssh

No.	Time	Source	Destination	Protocol	Length	Info
6	0.000915	172.18.0.5	172.18.0.4	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)
8	0.000978	172.18.0.4	172.18.0.5	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
10	0.001109	172.18.0.5	172.18.0.4	SSHv2	1498	Client: Key Exchange Init
11	0.001907	172.18.0.4	172.18.0.5	SSHv2	1146	Server: Key Exchange Init
12	0.002728	172.18.0.5	172.18.0.4	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
13	0.004558	172.18.0.4	172.18.0.5	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
15	4.439060	172.18.0.5	172.18.0.4	SSHv2	82	Client: New Keys
17	4.480924	172.18.0.5	172.18.0.4	SSHv2	110	Client: Encrypted packet (len=44)
19	4.480998	172.18.0.4	172.18.0.5	SSHv2	110	Server: Encrypted packet (len=44)
21	4.481069	172.18.0.5	172.18.0.4	SSHv2	134	Client: Encrypted packet (len=68)
22	4.486522	172.18.0.4	172.18.0.5	SSHv2	118	Server: Encrypted packet (len=52)
24	7.943688	172.18.0.5	172.18.0.4	SSHv2	214	Client: Encrypted packet (len=148)
26	7.986081	172.18.0.4	172.18.0.5	SSHv2	94	Server: Encrypted packet (len=28)
28	7.986258	172.18.0.5	172.18.0.4	SSHv2	178	Client: Encrypted packet (len=112)
30	7.995909	172.18.0.4	172.18.0.5	SSHv2	694	Server: Encrypted packet (len=628)
32	8.036603	172.18.0.4	172.18.0.5	SSHv2	110	Server: Encrypted packet (len=44)
34	8.036718	172.18.0.5	172.18.0.4	SSHv2	442	Client: Encrypted packet (len=376)
35	8.039663	172.18.0.4	172.18.0.5	SSHv2	174	Server: Encrypted packet (len=108)
36	8.039881	172.18.0.4	172.18.0.5	SSHv2	798	Server: Encrypted packet (len=732)
38	8.044775	172.18.0.4	172.18.0.5	SSHv2	142	Server: Encrypted packet (len=76)
40	80.920532	172.18.0.5	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
41	80.920796	172.18.0.4	172.18.0.5	SSHv2	102	Server: Encrypted packet (len=36)
43	81.334150	172.18.0.5	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
44	81.334394	172.18.0.4	172.18.0.5	SSHv2	102	Server: Encrypted packet (len=36)
46	81.544138	172.18.0.5	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
47	81.544363	172.18.0.4	172.18.0.5	SSHv2	102	Server: Encrypted packet (len=36)
49	81.703386	172.18.0.5	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
50	81.703609	172.18.0.4	172.18.0.5	SSHv2	102	Server: Encrypted packet (len=36)
52	82.320817	172.18.0.5	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
53	82.321072	172.18.0.4	172.18.0.5	SSHv2	118	Server: Encrypted packet (len=52)
55	82.321136	172.18.0.4	172.18.0.5	SSHv2	110	Server: Encrypted packet (len=44)
57	82.322405	172.18.0.4	172.18.0.5	SSHv2	242	Server: Encrypted packet (len=176)
59	82.322510	172.18.0.5	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
60	82.322546	172.18.0.5	172.18.0.4	SSHv2	126	Client: Encrypted packet (len=60)

Figura 4: Visualización general del tráfico SSH generado por C1

Posteriormente, se analizó en detalle el paquete *Key Exchange Init*. Este paquete es crítico para la identificación del cliente, ya que contiene la lista ordenada de algoritmos criptográficos soportados. Como se observa en la Figura 5, Wireshark permite extraer tanto el tamaño del paquete como la huella digital (HASSH) calculada a partir de dichos algoritmos.

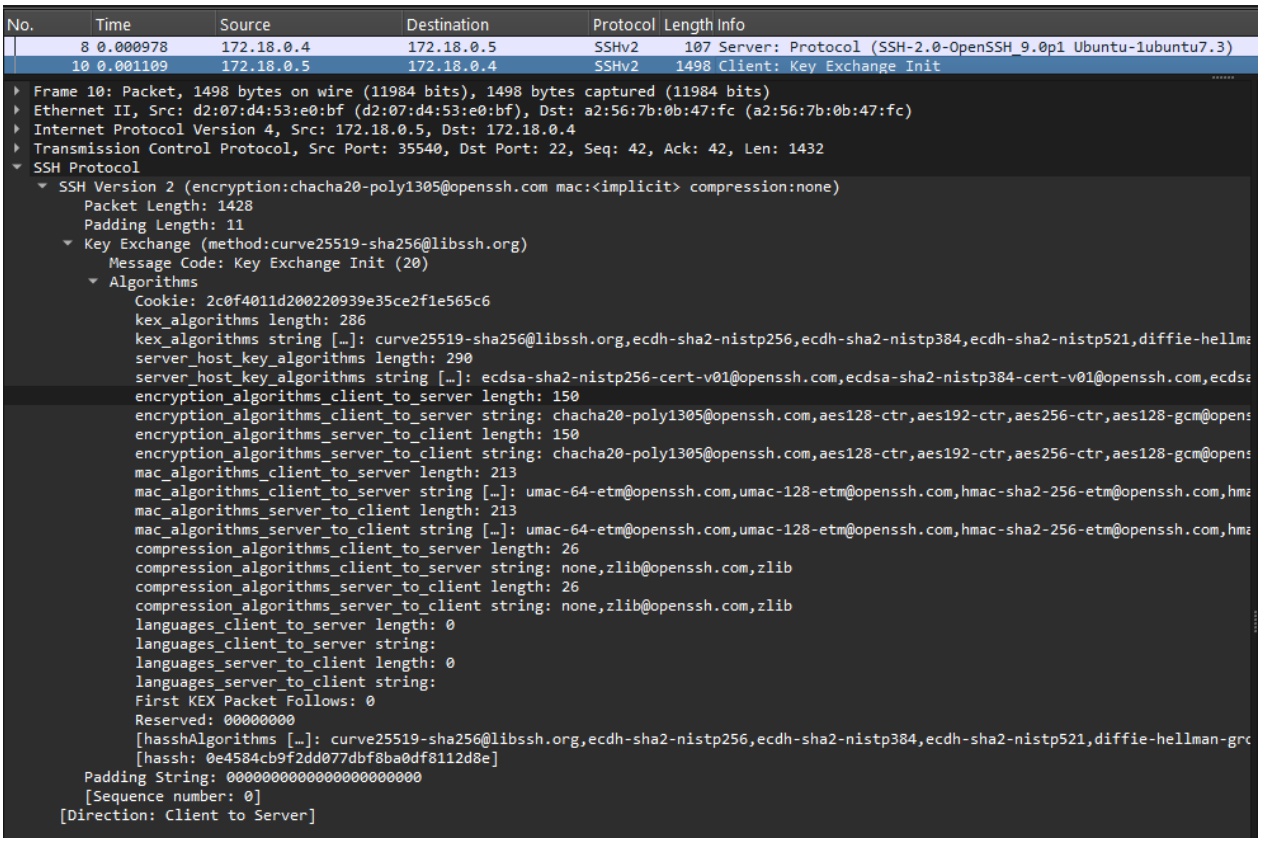


Figura 5: Detalle del Key Exchange Init y obtención del HASSH en C1

Finalmente, previo al establecimiento del túnel cifrado, el protocolo intercambia mensajes de identificación en texto plano. Esto permite verificar la versión exacta del cliente OpenSSH que se está ejecutando en el contenedor, tal como se evidencia en la siguiente captura:

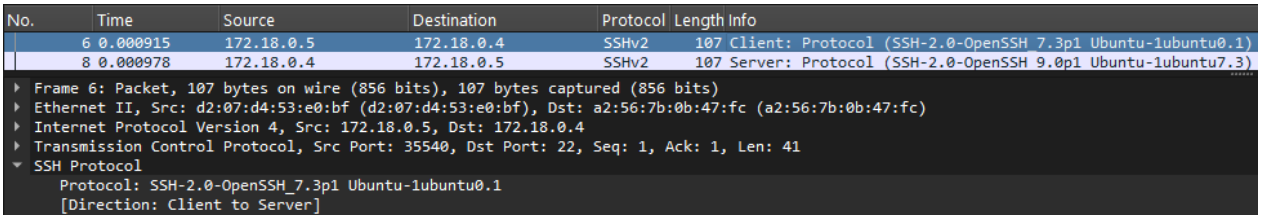


Figura 6: Identificación de la versión del cliente C1 en texto plano

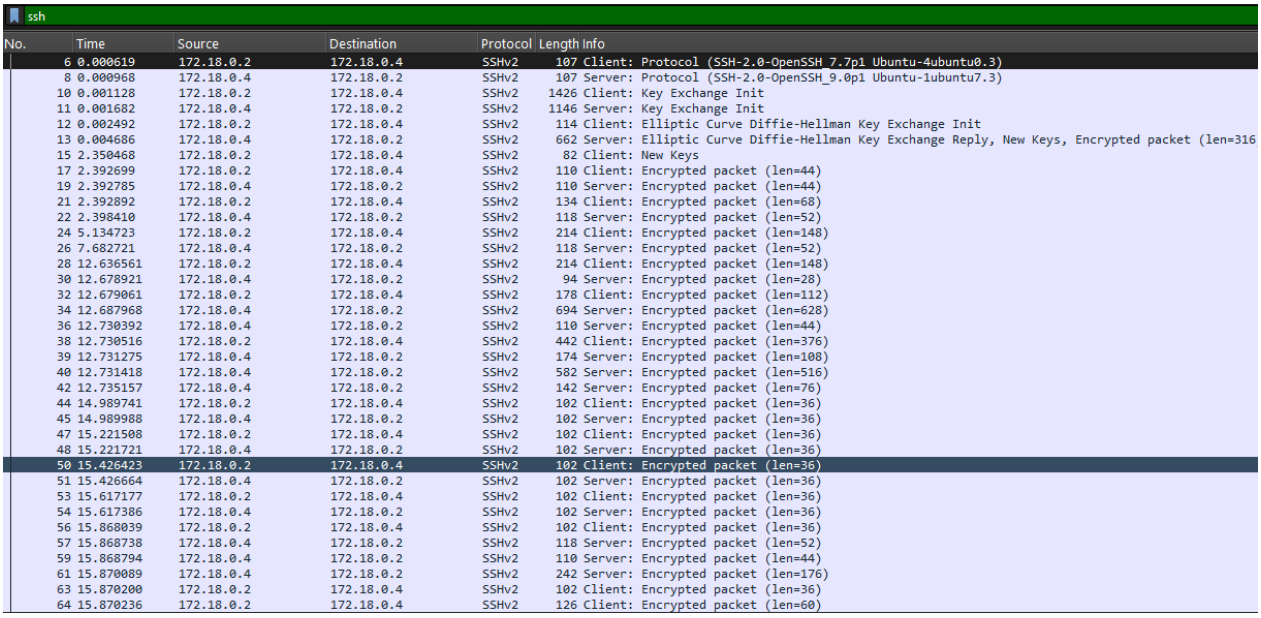
Del análisis del archivo de captura `c1.pcap`, se extrajeron y consolidaron los siguientes datos técnicos correspondientes a esta negociación:

- **Versión detectada (Texto plano):** SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1
- **Tamaño del paquete (Key Exchange Init):** 1498 bytes.

- **HASSH (Fingerprint MD5):** 0e4584cb9f2dd077dbf8ba0df8112d8e
- **Algoritmos principales (Inicio de la huella):**
  - curve25519-sha256@libssh.org
  - ecdh-sha2-nistp256
  - ecdh-sha2-nistp384

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

Siguiendo la metodología aplicada en el escenario anterior, se procedió a analizar el tráfico proveniente del cliente C2. A continuación se presentan las evidencias gráficas de la captura, la identificación del HASSH y la confirmación de la versión en texto plano.


ssh

No.	Time	Source	Destination	Protocol	Length	Info
6	0.000619	172.18.0.2	172.18.0.4	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3)
8	0.000968	172.18.0.4	172.18.0.2	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
10	0.001128	172.18.0.2	172.18.0.4	SSHv2	1426	Client: Key Exchange Init
11	0.001682	172.18.0.4	172.18.0.2	SSHv2	1146	Server: Key Exchange Init
12	0.002492	172.18.0.2	172.18.0.4	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
13	0.004686	172.18.0.4	172.18.0.2	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316
15	2.350468	172.18.0.2	172.18.0.4	SSHv2	82	Client: New Keys
17	2.392699	172.18.0.2	172.18.0.4	SSHv2	110	Client: Encrypted packet (len=44)
19	2.392785	172.18.0.4	172.18.0.2	SSHv2	110	Server: Encrypted packet (len=44)
21	2.392892	172.18.0.2	172.18.0.4	SSHv2	134	Client: Encrypted packet (len=68)
22	2.398410	172.18.0.4	172.18.0.2	SSHv2	118	Server: Encrypted packet (len=52)
24	5.134723	172.18.0.2	172.18.0.4	SSHv2	214	Client: Encrypted packet (len=148)
26	7.682721	172.18.0.4	172.18.0.2	SSHv2	118	Server: Encrypted packet (len=52)
28	12.636561	172.18.0.2	172.18.0.4	SSHv2	214	Client: Encrypted packet (len=148)
30	12.678921	172.18.0.4	172.18.0.2	SSHv2	94	Server: Encrypted packet (len=28)
32	12.679061	172.18.0.2	172.18.0.4	SSHv2	178	Client: Encrypted packet (len=112)
34	12.687968	172.18.0.4	172.18.0.2	SSHv2	694	Server: Encrypted packet (len=628)
36	12.730392	172.18.0.4	172.18.0.2	SSHv2	110	Server: Encrypted packet (len=44)
38	12.730516	172.18.0.2	172.18.0.4	SSHv2	442	Client: Encrypted packet (len=376)
39	12.731275	172.18.0.4	172.18.0.2	SSHv2	174	Server: Encrypted packet (len=108)
40	12.731418	172.18.0.4	172.18.0.2	SSHv2	582	Server: Encrypted packet (len=516)
42	12.735157	172.18.0.4	172.18.0.2	SSHv2	142	Server: Encrypted packet (len=76)
44	14.989741	172.18.0.2	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
45	14.989988	172.18.0.4	172.18.0.2	SSHv2	102	Server: Encrypted packet (len=36)
47	15.221508	172.18.0.2	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
48	15.221721	172.18.0.4	172.18.0.2	SSHv2	102	Server: Encrypted packet (len=36)
50	15.426423	172.18.0.2	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
51	15.426664	172.18.0.4	172.18.0.2	SSHv2	102	Server: Encrypted packet (len=36)
53	15.617177	172.18.0.2	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
54	15.617386	172.18.0.4	172.18.0.2	SSHv2	102	Server: Encrypted packet (len=36)
56	15.868039	172.18.0.2	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
57	15.868738	172.18.0.4	172.18.0.2	SSHv2	118	Server: Encrypted packet (len=52)
59	15.868794	172.18.0.4	172.18.0.2	SSHv2	110	Server: Encrypted packet (len=44)
61	15.870089	172.18.0.4	172.18.0.2	SSHv2	242	Server: Encrypted packet (len=176)
63	15.870200	172.18.0.2	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
64	15.870236	172.18.0.2	172.18.0.4	SSHv2	126	Client: Encrypted packet (len=60)

Figura 7: Visualización general del tráfico SSH generado por C2

Una vez identificado el flujo, se inspeccionó el paquete *Key Exchange Init*.

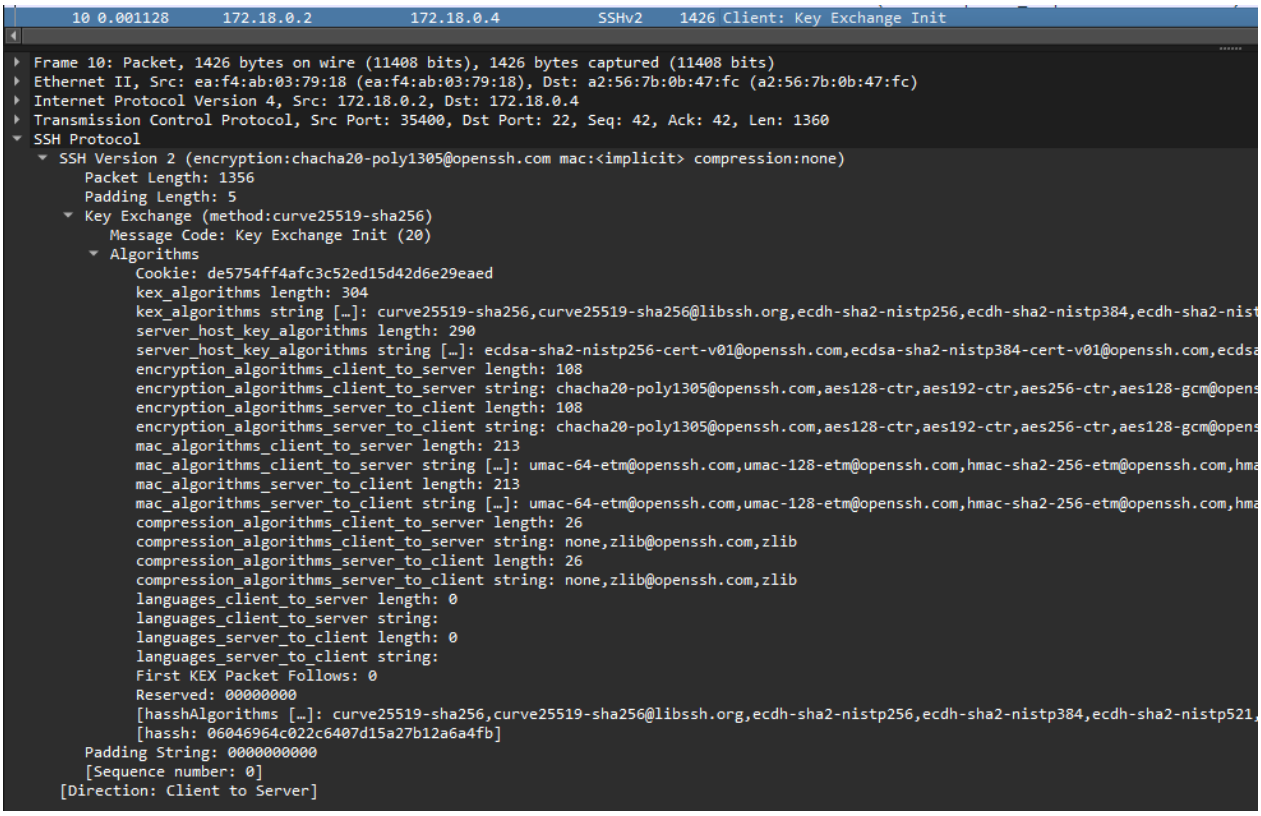


Figura 8: Detalle del Key Exchange Init y obtención del HASSH en C2

Para corroborar la identidad del cliente más allá de su huella criptográfica, se extrajo la cadena de identificación en texto plano enviada al servidor antes del cifrado.

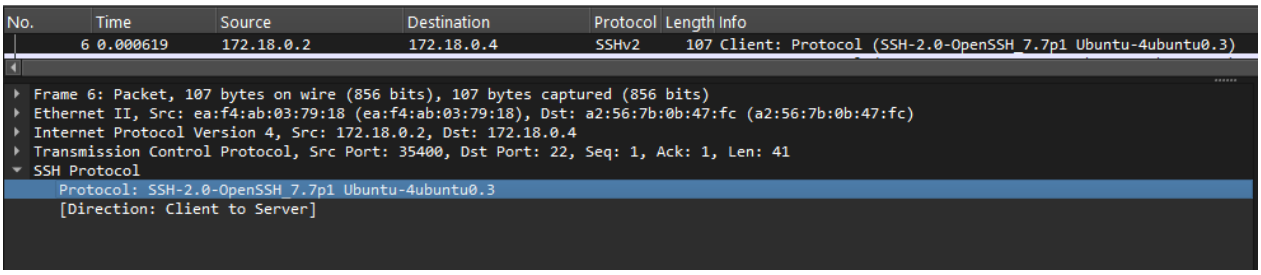


Figura 9: Identificación de la versión del cliente C2 en texto plano

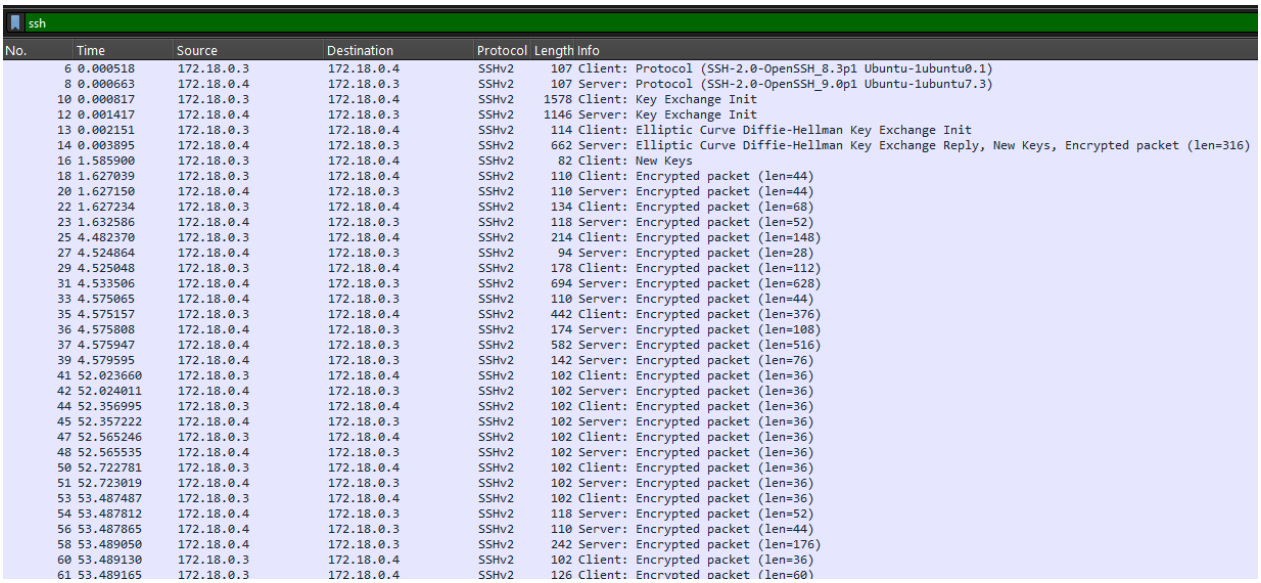
Del análisis del archivo de captura `c2.pcap`, se obtuvieron los siguientes datos:

- **Versión detectada (Texto plano):** SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3
- **Tamaño del paquete (Key Exchange Init):** 1426 bytes.
- **HASSH (Fingerprint MD5):** 06046964c022c6407d15a27b12a6a4fb

- **Algoritmos principales (Inicio de la huella):**
  - curve25519-sha256
  - curve25519-sha256@libssh.org
  - ecdh-sha2-nistp256

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

En este tercer escenario, se analizó el tráfico proveniente de un contenedor ejecutando Ubuntu 20.10. Al igual que en los casos anteriores, se capturó la secuencia completa de conexión para verificar el comportamiento del protocolo OpenSSH en su versión 8.3p1. La Figura 10 muestra el flujo de paquetes capturado, donde se observa la negociación inicial y el posterior intercambio de paquetes cifrados.



ssh

No.	Time	Source	Destination	Protocol	Length	Info
6	0.000518	172.18.0.3	172.18.0.4	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
8	0.000663	172.18.0.4	172.18.0.3	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
10	0.000817	172.18.0.3	172.18.0.4	SSHv2	1578	Client: Key Exchange Init
12	0.001417	172.18.0.4	172.18.0.3	SSHv2	1146	Server: Key Exchange Init
13	0.002151	172.18.0.3	172.18.0.4	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
14	0.003895	172.18.0.4	172.18.0.3	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
16	1.585900	172.18.0.3	172.18.0.4	SSHv2	82	Client: New Keys
18	1.627039	172.18.0.3	172.18.0.4	SSHv2	110	Client: Encrypted packet (len=44)
20	1.627150	172.18.0.4	172.18.0.3	SSHv2	110	Server: Encrypted packet (len=44)
22	1.627234	172.18.0.3	172.18.0.4	SSHv2	134	Client: Encrypted packet (len=68)
23	1.632586	172.18.0.4	172.18.0.3	SSHv2	118	Server: Encrypted packet (len=52)
25	4.482370	172.18.0.3	172.18.0.4	SSHv2	214	Client: Encrypted packet (len=148)
27	4.524864	172.18.0.4	172.18.0.3	SSHv2	94	Server: Encrypted packet (len=28)
29	4.525048	172.18.0.3	172.18.0.4	SSHv2	178	Client: Encrypted packet (len=112)
31	4.533506	172.18.0.4	172.18.0.3	SSHv2	694	Server: Encrypted packet (len=628)
33	4.575065	172.18.0.4	172.18.0.3	SSHv2	110	Server: Encrypted packet (len=44)
35	4.575157	172.18.0.3	172.18.0.4	SSHv2	442	Client: Encrypted packet (len=376)
36	4.575808	172.18.0.4	172.18.0.3	SSHv2	174	Server: Encrypted packet (len=108)
37	4.575947	172.18.0.4	172.18.0.3	SSHv2	582	Server: Encrypted packet (len=516)
39	4.579595	172.18.0.4	172.18.0.3	SSHv2	142	Server: Encrypted packet (len=76)
41	52.023660	172.18.0.3	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
42	52.024011	172.18.0.4	172.18.0.3	SSHv2	102	Server: Encrypted packet (len=36)
44	52.356995	172.18.0.3	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
45	52.357222	172.18.0.4	172.18.0.3	SSHv2	102	Server: Encrypted packet (len=36)
47	52.565246	172.18.0.3	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
48	52.565535	172.18.0.4	172.18.0.3	SSHv2	102	Server: Encrypted packet (len=36)
50	52.722781	172.18.0.3	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
51	52.723019	172.18.0.4	172.18.0.3	SSHv2	102	Server: Encrypted packet (len=36)
53	53.487487	172.18.0.3	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
54	53.487812	172.18.0.4	172.18.0.3	SSHv2	118	Server: Encrypted packet (len=52)
56	53.487865	172.18.0.4	172.18.0.3	SSHv2	110	Server: Encrypted packet (len=44)
58	53.489050	172.18.0.4	172.18.0.3	SSHv2	242	Server: Encrypted packet (len=176)
60	53.489130	172.18.0.3	172.18.0.4	SSHv2	102	Client: Encrypted packet (len=36)
61	53.489165	172.18.0.3	172.18.0.4	SSHv2	126	Client: Encrypted packet (len=60)

Figura 10: Visualización general del tráfico SSH generado por C3

El análisis profundo del paquete *Key Exchange Init* reveló un nuevo cambio en la huella digital respecto a la versión anterior. Como se aprecia en la siguiente imagen, el tamaño del paquete aumentó considerablemente, lo que se correlaciona con la actualización y adición de algoritmos criptográficos soportados por defecto en esta distribución más moderna.

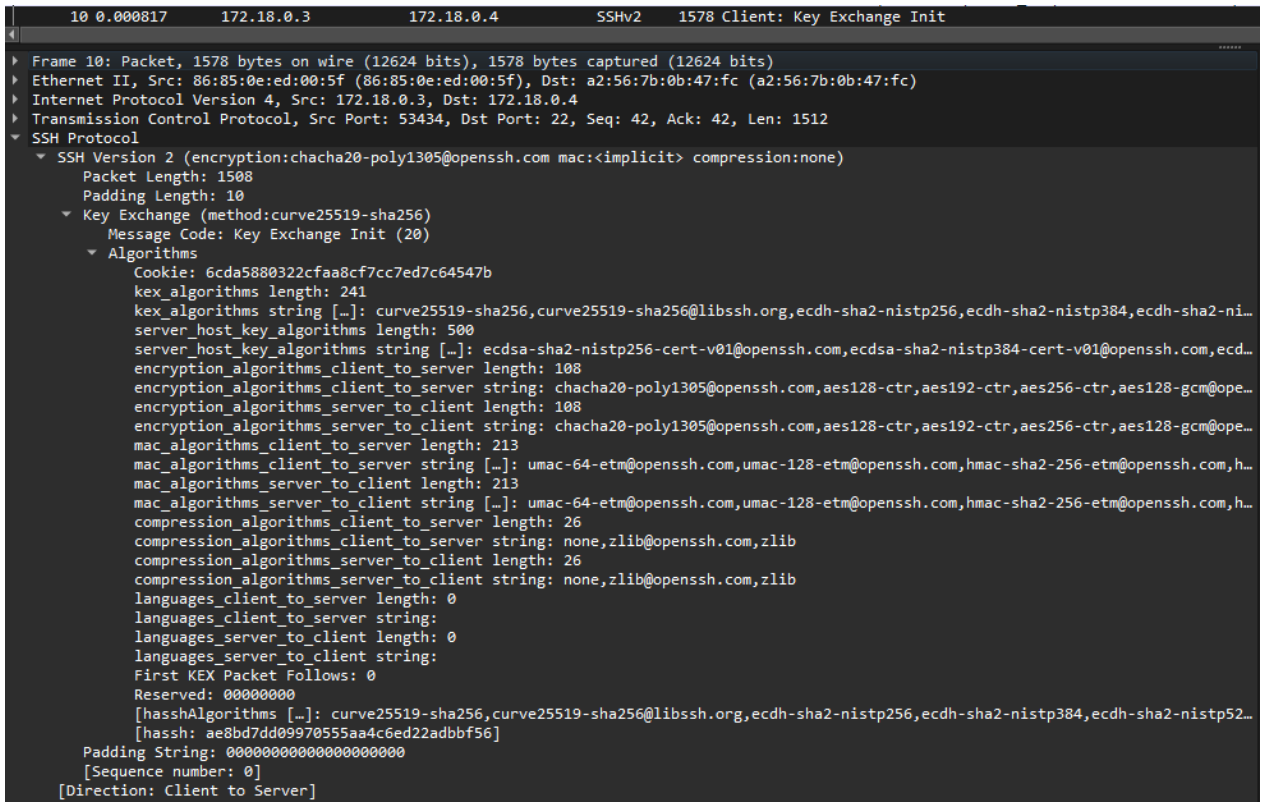


Figura 11: Detalle del Key Exchange Init y obtención del HASSH en C3

Finalmente, la inspección del paquete de identificación en texto plano confirmó que la versión del cliente utilizada es OpenSSH 8.3p1, validando así la correcta configuración del entorno de pruebas para este escenario.

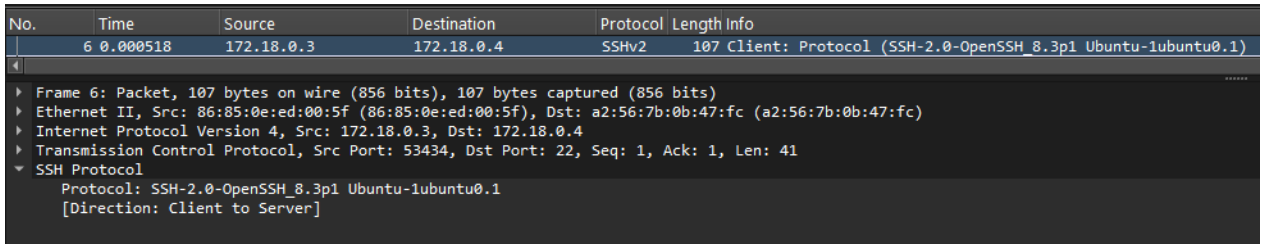


Figura 12: Identificación de la versión del cliente C3 en texto plano

Del análisis del archivo de captura `c3.pcap`, se obtuvieron los siguientes datos técnicos consolidados:

- **Versión detectada (Texto plano):** SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1
- **Tamaño del paquete (Key Exchange Init):** 1578 bytes.

- **HASSH (Fingerprint MD5):** ae8bd7dd09970555aa4c6ed22adbbf56
- **Algoritmos principales (Inicio de la huella):**
  - curve25519-sha256
  - curve25519-sha256@libssh.org
  - ecdh-sha2-nistp256

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

Finalmente, se analizó el tráfico generado por el cliente C4. A diferencia de los casos anteriores, esta captura se realizó sobre la interfaz de *loopback* (`lo`), lo que se evidencia en las direcciones IP de origen y destino (`::1`, correspondientes a IPv6 localhost). La Figura 13 muestra el flujo completo, donde destaca el uso del protocolo SSHv2 sobre una conexión local.

ssh

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000158	::1	::1	SSHv2	127	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
6	0.001351	::1	::1	SSHv2	127	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
8	0.001660	::1	::1	SSHv2	1590	Client: Key Exchange Init
9	0.003135	::1	::1	SSHv2	1166	Server: Key Exchange Init
10	0.034652	::1	::1	SSHv2	1294	Client: Diffie-Hellman Key Exchange Init
11	0.041522	::1	::1	SSHv2	1650	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
13	5.678805	::1	::1	SSHv2	102	Client: New Keys
15	5.722826	::1	::1	SSHv2	130	Client: Encrypted packet (len=44)
17	5.722892	::1	::1	SSHv2	130	Server: Encrypted packet (len=44)
19	5.722947	::1	::1	SSHv2	154	Client: Encrypted packet (len=68)
20	5.728439	::1	::1	SSHv2	138	Server: Encrypted packet (len=52)
22	9.018816	::1	::1	SSHv2	234	Client: Encrypted packet (len=148)
23	9.061327	::1	::1	SSHv2	114	Server: Encrypted packet (len=28)
25	9.061458	::1	::1	SSHv2	198	Client: Encrypted packet (len=112)
26	9.079784	::1	::1	SSHv2	714	Server: Encrypted packet (len=628)
27	9.080064	::1	::1	SSHv2	666	Client: Encrypted packet (len=580)
28	9.080091	::1	::1	SSHv2	130	Server: Encrypted packet (len=44)
29	9.080128	::1	::1	SSHv2	462	Client: Encrypted packet (len=376)
30	9.082719	::1	::1	SSHv2	626	Server: Encrypted packet (len=540)
31	9.083780	::1	::1	SSHv2	194	Server: Encrypted packet (len=108)
33	9.083927	::1	::1	SSHv2	602	Server: Encrypted packet (len=516)
34	9.096476	::1	::1	SSHv2	162	Server: Encrypted packet (len=76)
36	13.248679	::1	::1	SSHv2	122	Client: Encrypted packet (len=36)
37	13.248999	::1	::1	SSHv2	122	Server: Encrypted packet (len=36)
39	13.676231	::1	::1	SSHv2	122	Client: Encrypted packet (len=36)
40	13.676500	::1	::1	SSHv2	122	Server: Encrypted packet (len=36)
42	13.819172	::1	::1	SSHv2	122	Client: Encrypted packet (len=36)
43	13.819414	::1	::1	SSHv2	122	Server: Encrypted packet (len=36)
45	13.991271	::1	::1	SSHv2	122	Client: Encrypted packet (len=36)
46	13.991462	::1	::1	SSHv2	122	Server: Encrypted packet (len=36)
48	14.326996	::1	::1	SSHv2	122	Client: Encrypted packet (len=36)
49	14.327271	::1	::1	SSHv2	138	Server: Encrypted packet (len=52)
51	14.327311	::1	::1	SSHv2	130	Server: Encrypted packet (len=44)
53	14.328857	::1	::1	SSHv2	262	Server: Encrypted packet (len=176)
55	14.328946	::1	::1	SSHv2	122	Client: Encrypted packet (len=36)
56	14.328977	::1	::1	SSHv2	146	Client: Encrypted packet (len=60)

Figura 13: Visualización general del tráfico SSH en Loopback (C4)

El análisis del paquete *Key Exchange Init* presenta el tamaño de paquete más grande de todo el laboratorio. Como se observa en la siguiente captura, OpenSSH 9.0p1 incorpora algoritmos de intercambio de claves híbridos post-cuánticos (como `sntrup761`), lo que incrementa significativamente la carga útil del paquete y genera una huella HASSH única.

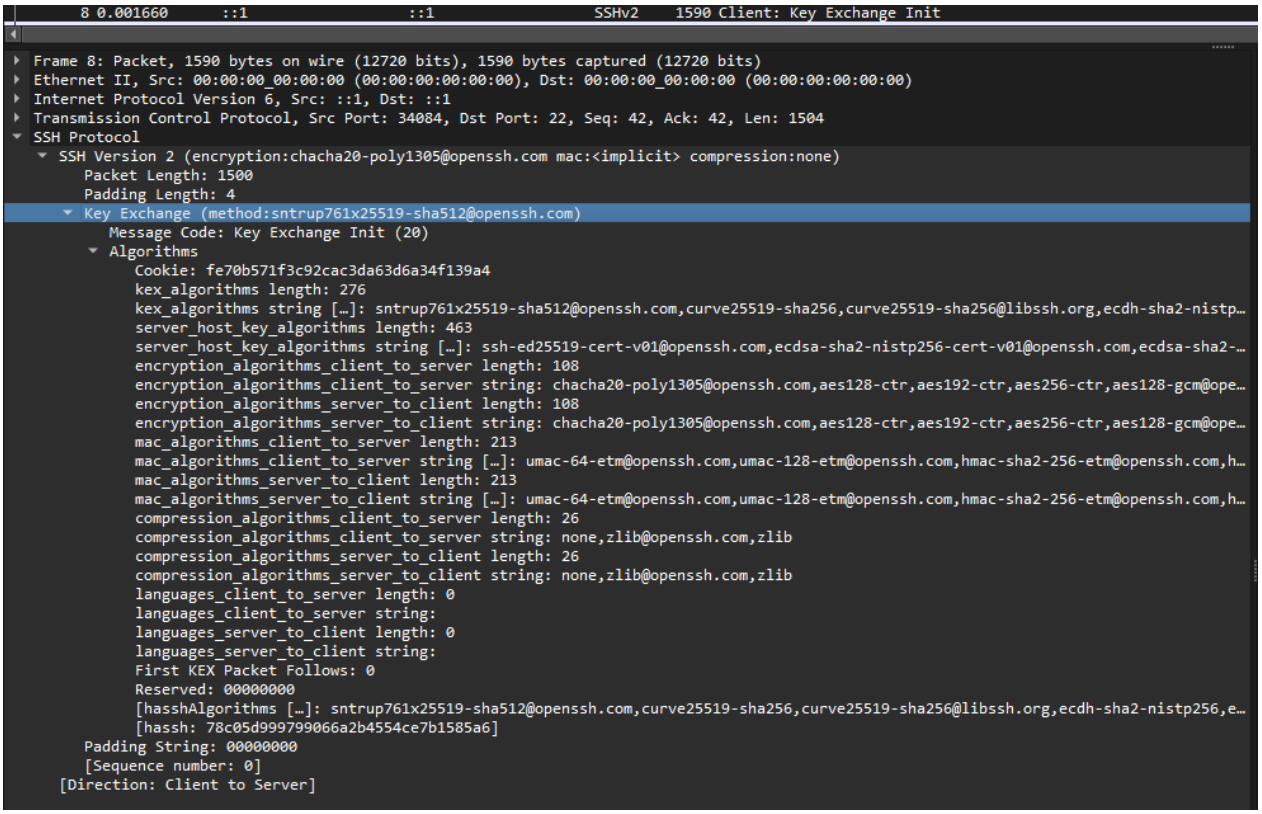


Figura 14: Detalle del Key Exchange Init, HASSH y algoritmos post-cuánticos en C4

Por último, se verificó la versión del cliente mediante la inspección del paquete de protocolo en texto plano. La captura confirma que se está utilizando OpenSSH 9.0p1 sobre Ubuntu 22.10, validando el escenario de prueba más moderno.

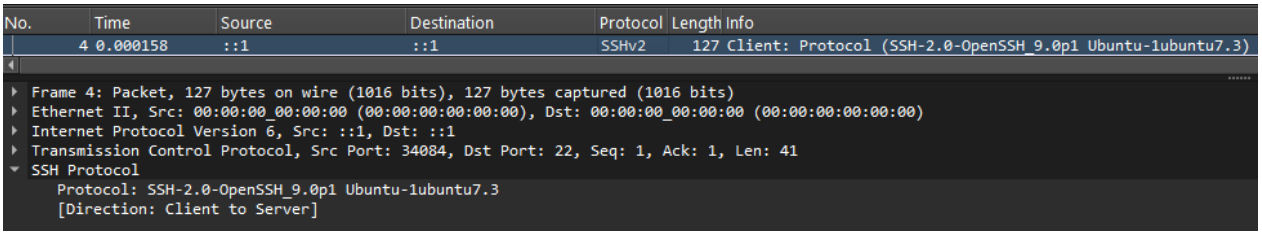


Figura 15: Identificación de la versión OpenSSH 9.0p1 en texto plano

Del análisis forense del archivo de captura `c4.pcap`, se obtuvieron los siguientes datos técnicos consolidados:

- **Versión detectada (Texto plano):** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Tamaño del paquete (Key Exchange Init):** 1590 bytes.
- **HASSH (Fingerprint MD5):** 78c05d999799066a2b4554ce7b1585a6

- **Algoritmos principales:**
  - sntrup761x25519-sha512@openssh.com (Híbrido Post-Cuánticos)
  - curve25519-sha256
  - curve25519-sha256@libssh.org

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

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

1.8.1. C1

En el análisis del tráfico generado por el primer cliente, el paquete de protocolo capturado revela la cadena de identificación `SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1`. Esta información en texto plano expone que el sistema está utilizando el protocolo SSH versión 2.0 y ejecuta una versión antigua del software OpenSSH (7.3p1), coincidente con la distribución Ubuntu 16.10 utilizada en este escenario.

1.8.2. C2

Para el segundo escenario, la captura del tráfico muestra que la cadena de identificación enviada al servidor es `SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3`. Este mensaje en claro indica una actualización en el software del cliente respecto al caso anterior, correspondiendo a la versión predeterminada incluida en los repositorios de Ubuntu 18.10.

1.8.3. C3

En el caso del tercer cliente, la información contenida en el paquete inicial permite identificar la cadena `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`. Este dato confirma que el cliente está operando con la versión 8.3p1 de OpenSSH, lo cual es consistente con el entorno de Ubuntu 20.10 desplegado en el contenedor C3.

1.8.4. C4/S1

Finalmente, el tráfico capturado en la interfaz de loopback para el último cliente presenta la cadena `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`. Esta información en texto plano delata el uso de la versión más reciente de todo el laboratorio (OpenSSH 9.0p1), propia del sistema Ubuntu 22.10, antes de que comience cualquier tipo de encriptación de la sesión.

1.9. Diferencia entre C1 y C2

Al comparar el tráfico de C1 (Ubuntu 16.10) con C2 (Ubuntu 18.10), la diferencia principal está en el tamaño del paquete de negociación.

- **Comparación:** El paquete bajó de 1498 bytes en C1 a 1426 bytes en C2.
- **Análisis:** Esto indica que al pasar a una versión más nueva de OpenSSH (7.7p1), se eliminaron algunos algoritmos antiguos que venían por defecto en la versión anterior, lo que hizo que la lista de propuestas fuera más corta y cambiara el HASSH.

1.10. Diferencia entre C2 y C3

Entre C2 (Ubuntu 18.10) y C3 (Ubuntu 20.10) se nota un cambio de tendencia, donde el paquete vuelve a crecer.

- **Comparación:** El tamaño aumentó de 1426 bytes en C2 a 1578 bytes en C3.
- **Análisis:** La diferencia se debe a que la versión más moderna de Ubuntu (20.10) incluye soporte para nuevos algoritmos de seguridad que no existían en la versión anterior. Al haber más opciones disponibles para negociar, el paquete pesa más y la huella digital es distinta.

1.11. Diferencia entre C3 y C4

Finalmente, al comparar C3 con el cliente más actual C4 (Ubuntu 22.10), se registra el tamaño más grande de todos.

- **Comparación:** Hubo un leve aumento de 1578 bytes en C3 a 1590 bytes en C4.
- **Análisis:** C4 utiliza la versión más reciente de OpenSSH (9.0p1) de todo el laboratorio. Al ser el software más actual, trae habilitada la lista más completa de algoritmos de cifrado modernos, lo que genera el paquete de inicialización más pesado y una huella HASSH única para esta versión.

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión "?"

Para identificar al cliente cuya versión aparece ofuscada como `SSH-2.0-OpenSSH_?` en la evidencia del laboratorio, se utilizó el tamaño del paquete *Key Exchange Init* como huella distintiva.

- **Evidencia del problema:** La captura de referencia muestra que el paquete *Key Exchange Init* tiene una longitud exacta de **1578 bytes**.
- **Análisis comparativo:** Al contrastar este valor con los datos experimentales obtenidos en la Parte 1:
  - C1 (OpenSSH 7.3p1): 1498 bytes.
  - C2 (OpenSSH 7.7p1): 1426 bytes.
  - **C3 (OpenSSH 8.3p1): 1578 bytes.**
  - C4 (OpenSSH 9.0p1): 1590 bytes.

  El patrón de tráfico coincide exactamente con el generado por el cliente **C3**. Por lo tanto, se determina que la versión oculta corresponde a **OpenSSH 8.3p1** sobre Ubuntu 20.10.

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

Para cumplir con el requerimiento de replicar el tráfico del cliente incógnito y generar un paquete que contenga explícitamente la cadena de versión ofuscada `SSH-2.0-OpenSSH_?`, se utilizó la técnica de *Banner Spoofing* mediante la inyección manual de paquetes TCP crudos.

Dado que el análisis previo identificó al cliente C3 como la fuente original, se procedió a inyectar el tráfico desde dicho contenedor hacia el servidor S1. El procedimiento se realizó ejecutando un comando desde el host que instruye al contenedor C3 a establecer una conexión directa con `netcat` y enviar la cadena de texto modificada, simulando el inicio del protocolo SSH.

El comando técnico ejecutado fue:

```
docker compose exec c3 bash -c "echo 'SSH-2.0-OpenSSH_?' | nc S1 22"
```

Como resultado, se logró obtener lo solicitado replicando y capturarando el paquete con la versión modificada. La Figura 16 muestra la captura en Wireshark donde el paquete número 5 exhibe claramente la cadena `SSH-2.0-OpenSSH_?` en la columna de información.

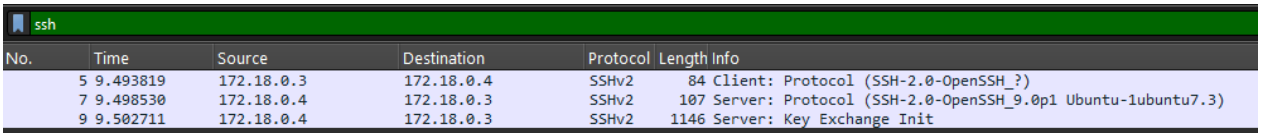


Figura 16: Evidencia de la replicación exitosa: Captura del paquete con la versión ofuscada SSH-2.0-OpenSSH_?

3. Desarrollo (Parte 3)

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

Para lograr esta reducción drástica, se configuró el servidor S1 para que anunciara una única opción por cada categoría de seguridad.

Procedimiento realizado:

1. Se accedió al contenedor S1 y se modificó el archivo de configuración `sshd_config` mediante un comando único.

2. Se añadieron restricciones para limitar la negociación a algoritmos de nombres cortos y llaves pequeñas. Específicamente, se forzó el uso de `curve25519` (intercambio), `aes128-ctr` (cifrado) y, lo más importante, se restringió el algoritmo de llave de host a `ssh-ed25519`, eliminando el peso extra de las llaves RSA.

3. El comando ejecutado fue:

```
docker compose exec s1 bash -c "echo 'HostKeyAlgorithms ssh-ed25519'
   >> /etc/ssh/sshd_config && \
echo 'KexAlgorithms curve25519-sha256' >> /etc/ssh/sshd_config && \
echo 'Ciphers aes128-ctr' >> /etc/ssh/sshd_config && \
echo 'MACs hmac-sha2-256' >> /etc/ssh/sshd_config && \
service ssh restart"
```

Evidencia: Tras aplicar la configuración y capturar nuevamente el tráfico, se observa en la Figura 17 que el paquete del servidor (número 12) redujo su tamaño a **258 bytes**, cumpliendo exitosamente con el requisito de ser menor a 300 bytes.

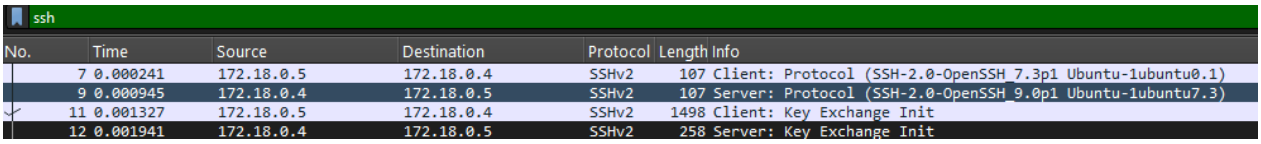


Figura 17: Captura de Wireshark mostrando la reducción exitosa del paquete Server Key Exchange Init a 258 bytes

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es la implementación de código abierto más utilizada del protocolo Secure Shell (SSH), diseñada para proporcionar un canal seguro de administración sobre redes no confiables, reemplazando herramientas antiguas y vulnerables. Su funcionamiento se basa en una

arquitectura cliente-servidor que garantiza la confidencialidad y la integridad de los datos mediante un esquema de criptografía híbrida. El protocolo inicia la comunicación utilizando criptografía asimétrica para el intercambio de claves y la autenticación, lo que permite negociar una clave de sesión compartida; posteriormente, utiliza criptografía simétrica de alto rendimiento para cifrar todo el tráfico de datos, comandos y transferencias de archivos que fluyen a través del túnel establecido.

4.2. Capas de Seguridad en OpenSSH

La arquitectura de seguridad de OpenSSH se organiza en tres capas jerárquicas que operan secuencialmente para proteger la comunicación. La base es la Capa de Transporte, responsable de la negociación inicial de algoritmos, el intercambio de claves (generalmente Diffie-Hellman), la autenticación del servidor mediante claves de host y la provisión de cifrado e integridad perfecta hacia adelante. Una vez asegurado el canal, interviene la Capa de Autenticación de Usuario, que verifica la identidad del cliente utilizando diversos métodos como contraseñas, criptografía de clave pública o autenticación basada en host. Finalmente, la Capa de Conexión opera sobre el canal ya autenticado y cifrado, permitiendo multiplexar múltiples flujos lógicos a través de una única conexión, gestionando servicios como sesiones de terminal interactivas, reenvío de puertos TCP y tunelización.

4.3. Identificación de que protocolos no se cumplen

Desde una perspectiva teórica y de diseño, el protocolo SSH presenta limitaciones en el cumplimiento estricto de ciertos principios de seguridad. En primer lugar, el principio de Disponibilidad no está garantizado intrínsecamente por el protocolo; debido a que la fase de negociación de claves requiere operaciones matemáticas computacionalmente costosas, el servicio es susceptible a ataques de denegación de servicio (DoS) o agotamiento de recursos si se inician múltiples conexiones simultáneas maliciosas. En cuanto al No Repudio, este principio no se cumple plenamente cuando se configura la autenticación por contraseña, ya que, a diferencia de las firmas digitales con claves privadas, una contraseña es un secreto compartido que no vincula de manera matemática e irrefutable una acción con una identidad única. Por último, existen debilidades en la confidencialidad de los metadatos, dado que el protocolo transmite las versiones de software y sistema operativo en texto plano durante el saludo inicial, exponiendo información de la infraestructura antes de establecer el cifrado.

Conclusiones y comentarios

Con el desarrollo de este laboratorio, se logró comprender en profundidad el funcionamiento interno del protocolo SSH mediante el uso de contenedores Docker. Un punto clave observado fue que las conexiones SSH no son idénticas entre sí; el tráfico de red varía considerablemente dependiendo de la antigüedad del sistema operativo utilizado.

Se pudo evidenciar que las versiones más modernas de Ubuntu (como el caso del cliente C4) generan paquetes de negociación mucho más pesados. Esto se debe a que incorporan por defecto medidas de seguridad más avanzadas, como la protección post-cuántica, lo que incrementa la cantidad de datos intercambiados al inicio en comparación con las versiones más antiguas.

Respecto a la identificación de sistemas, la actividad de replicación demostró que no es seguro confiar únicamente en el texto que muestra la conexión (banner). Aunque fue posible enviar una versión falsa al servidor, el tamaño del paquete y la huella digital (HASSH) se mantuvieron, lo que indica que analizar el comportamiento del tráfico es un método mucho más fiable que leer el mensaje de bienvenida para identificar un cliente real.

Adicionalmente, se comprobó la efectividad de realizar una configuración personalizada ("hardening") en el servidor. Al eliminar las opciones innecesarias y dejar solo los algoritmos requeridos, el tamaño del paquete de negociación se redujo drásticamente (de más de 1000 bytes a menos de 300). Esto demuestra que la configuración por defecto no siempre es la más eficiente y que limpiar los cifrados permitidos mejora tanto el rendimiento como la seguridad.

En conclusión, si bien SSH es un protocolo robusto para garantizar la confidencialidad, este laboratorio permitió demostrar que la seguridad real depende de cómo se configure. El uso de configuraciones por defecto o métodos de autenticación simples (como contraseñas) mantiene ciertas vulnerabilidades que pueden ser mitigadas con una administración adecuada.