Informe Laboratorio 5

Sección 3

Lucas Araya
e-mail: lucas.araya_t@mail.udp.cl

Noviembre de 2024

Índice

Descripción de actividades

Para este último laboratorio, nuestro informante ya sabe que puede establecer un medio seguro sin un intercambio previo de una contraseña, gracias al protocolo diffie-hellman. El problema es que ahora no sabe si confiar en el equipo con el cual establezca comunicación, ya que las credenciales de usuario pueden haber sido divulgadas por algún soplón.

Para el presente laboratorio deberá:

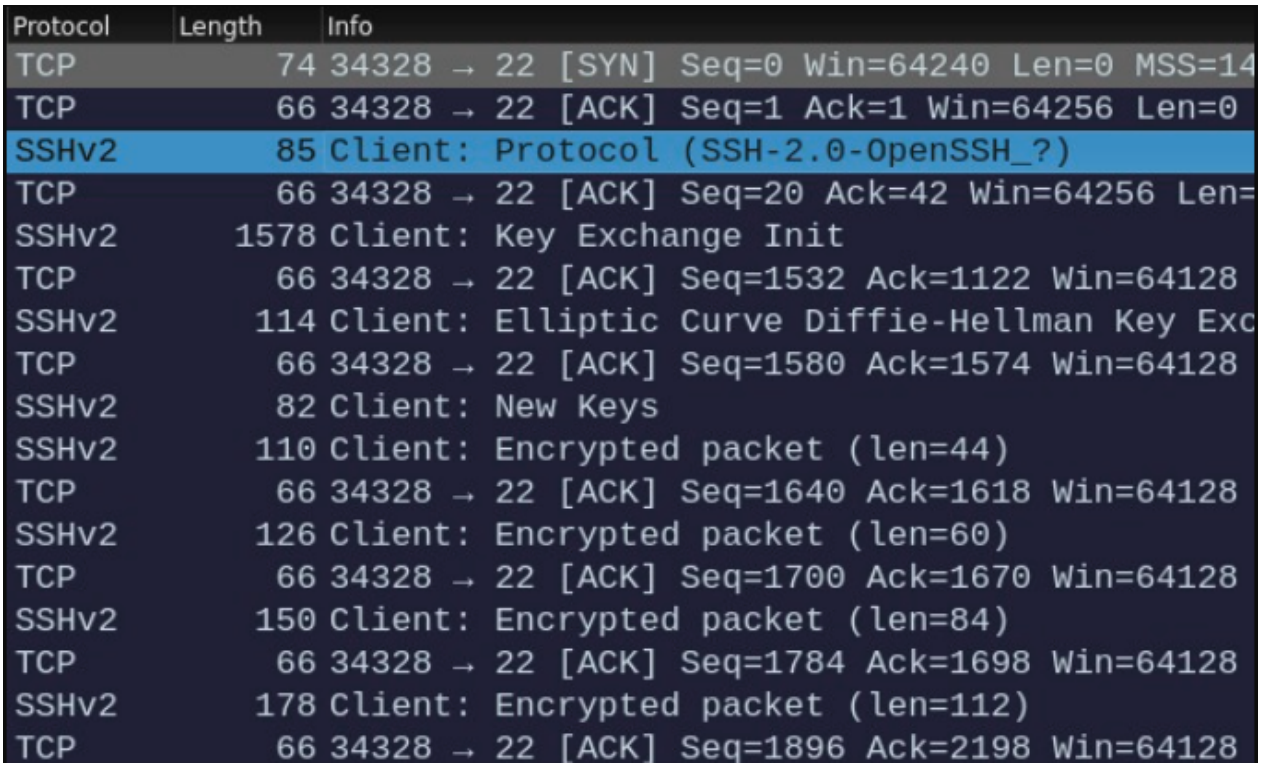
- Crear 4 contenedores en Docker o Podman, donde cada uno tendrá el siguiente SO: Ubuntu 16.10, Ubuntu 18.10, Ubuntu 20.10 y Ubuntu 22.10 a los cuales se llamarán C1, C2, C3 y C4 respectivamente.
  El equipo con Ubuntu 22.10 también será utilizado como S1.
- Para cada uno de ellos, deberá instalar el cliente openSSH disponible en los repositorios de apt, y para el equipo S1 deberá también instalar el servidor openSSH.
- En S1 deberá crear el usuario "**prueba**"con contraseña "**prueba**", para acceder a él desde los clientes por el protocolo SSH.
- En total serán 4 escenarios, donde cada uno corresponderá a los siguientes equipos:
  - C1 $\rightarrow$ S1
  - C2 $\rightarrow$ S1
  - C3 $\rightarrow$ S1
  - C4 $\rightarrow$ S1

Pasos:

1. Para cada uno de los 4 escenarios, deberá capturar el tráfico generado por cada conexión con el server. A partir de cada handshake, deberá analizar el patrón de tráfico generado por cada cliente y adicionalmente obtener el HASSH que lo identifique. De esta forma podrá obtener una huella digital para cada cliente a partir de su tráfico. Cada HASSH deberá compararlo con la base de datos HASSH disponible en el módulo de TLS, e identificar si el hash obtenido corresponde a la misma versión de su cliente.

   Indique el tamaño de los paquetes del flujo generados por el cliente y el contenido asociado a cada uno de ellos. Indique qué información distinta contiene el escenario siguiente (diff incremental). El objetivo de este paso es identificar claramente los cambios entre las distintas versiones de ssh.

2. Para poder identificar que el usuario efectivamente es el informante, éste utilizará una versión única de cliente. ¿Con qué cliente SSH se habrá generado el siguiente tráfico?



Protocol	Length	Info
TCP	74	34328 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=14
TCP	66	34328 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0
SSHv2	85	Client: Protocol (SSH-2.0-OpenSSH_?)
TCP	66	34328 → 22 [ACK] Seq=20 Ack=42 Win=64256 Len=
SSHv2	1578	Client: Key Exchange Init
TCP	66	34328 → 22 [ACK] Seq=1532 Ack=1122 Win=64128
SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exc
TCP	66	34328 → 22 [ACK] Seq=1580 Ack=1574 Win=64128
SSHv2	82	Client: New Keys
SSHv2	110	Client: Encrypted packet (len=44)
TCP	66	34328 → 22 [ACK] Seq=1640 Ack=1618 Win=64128
SSHv2	126	Client: Encrypted packet (len=60)
TCP	66	34328 → 22 [ACK] Seq=1700 Ack=1670 Win=64128
SSHv2	150	Client: Encrypted packet (len=84)
TCP	66	34328 → 22 [ACK] Seq=1784 Ack=1698 Win=64128
SSHv2	178	Client: Encrypted packet (len=112)
TCP	66	34328 → 22 [ACK] Seq=1896 Ack=2198 Win=64128

Figura 1: Tráfico generado del informante

Replique este tráfico generado en la imagen. Debe generar el tráfico con la misma versión resaltada en azul. Recuerde que toda la información generada es parte del sw, por lo tanto usted puede modificar toda la información.

3. Para que el informante esté seguro de nuestra identidad, nos pide que el patrón del tráfico de nuestro server también sea modificado, hasta que el Key Exchange Init del server sea menor a 300 bytes. Indique qué pasos realizó para lograr esto.

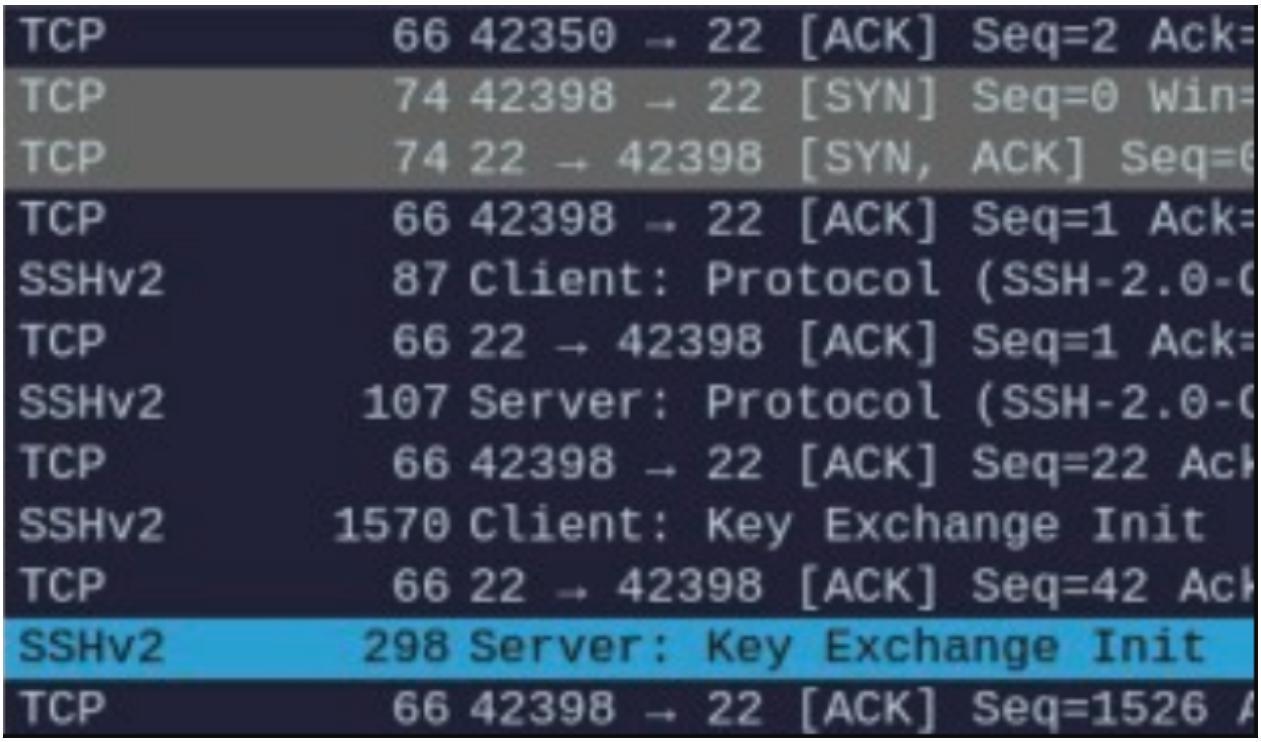


Figura 2: Captura del Key Exchange

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 códigos de los archivos `Dockerfile` utilizados para la implementación de los contenedores `C1`, `C2`, `C3` y `C4`:

- Los contenedores `C1`, `C2` y `C3` comparten la misma estructura de implementación, diferenciándose únicamente en la versión de Ubuntu utilizada (16.10, 18.10 y 20.10, respectivamente). En cada uno de estos contenedores se instala el cliente *OpenSSH*.
- En todos los `Dockerfile`, las líneas 4 y 5 son idénticas, ya que se emplean para configurar repositorios *End of Life (EOL)*, lo que permite la instalación de paquetes en versiones de Ubuntu que ya no cuentan con soporte oficial.

- El contenedor `C4` incluye, además de la configuración básica, la instalación y configuración del servidor *OpenSSH*, junto con la creación de un usuario denominado `prueba` con contraseña `prueba`.

1.1.1. C1

c1.dockerfile > FROM

```
FROM ubuntu:16.10

# Actualizar repositorios para versiones EOL y configurar apt
RUN sed -i 's/archive.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    sed -i 's/security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    apt-get update && apt-get install -y openssh-client && apt-get clean

CMD ["/bin/bash"]

```

Figura 3: Cliente 1 - ubuntu:16:10

1.1.2. C2

```
FROM ubuntu:18.10

# Actualizar repositorios para versiones EOL y configurar apt
RUN sed -i 's/archive.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    sed -i 's/security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    apt-get update && apt-get install -y openssh-client && apt-get clean

CMD ["/bin/bash"]
```

Figura 4: Cliente 2 - ubuntu:18:10

1.1.3. C3

```
FROM ubuntu:20.10

# Actualizar repositorios para versiones EOL y configurar apt
RUN sed -i 's/archive.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    sed -i 's/security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    apt-get update && apt-get install -y openssh-client && apt-get clean

CMD ["/bin/bash"]
```

Figura 5: Cliente 3 - ubuntu:20:10

1.1.4. C4/S1

```
FROM ubuntu:22.10

# Actualizar repositorios para versiones EOL y configurar apt
RUN sed -i 's/archive.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    sed -i 's/security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list && \
    apt-get update && apt-get install -y openssh-client openssh-server && apt-get clean

# Configuración inicial para el servidor SSH
RUN mkdir /var/run/sshd && \
    echo 'PermitRootLogin yes' >> /etc/ssh/sshd_config && \
    echo 'PasswordAuthentication yes' >> /etc/ssh/sshd_config && \
    useradd -m -s /bin/bash prueba && echo "prueba:prueba" | chpasswd

# Exponer el puerto 22 para SSH
EXPOSE 22

# Comando para iniciar el servidor SSH
CMD ["/usr/sbin/sshd", "-D"]
```

Figura 6: Cliente 4 / servidor - ubuntu:22:10

1.2. Creación de las credenciales para S1

Como se puede observar en la implementación, durante la configuración del contenedor `C4`, se instalan tanto el cliente como el servidor *OpenSSH*. Además, se configuran las credenciales mediante el comando:

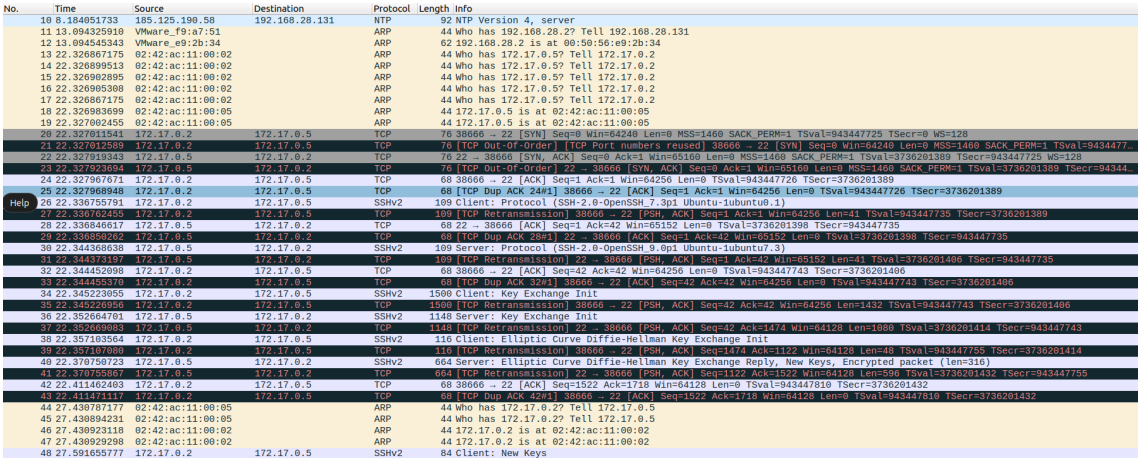
```
useradd -m -s /bin/bash prueba && echo "prueba:prueba" | chpasswd
```

Este comando crea un usuario llamado `prueba` con su correspondiente directorio personal (`-m`), asignándole el intérprete de comandos `/bin/bash` (`-s`). Posteriormente, establece la contraseña `prueba` utilizando el comando `chpasswd`.

Adicionalmente, se habilita la autenticación mediante contraseña y se expone el puerto 22 para permitir conexiones SSH al contenedor.

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

A continuación se puede observar el trafico capturado al hacer la conexion desde el cliente 1 al servidor OpenSSH:



No.	Time	Source	Destination	Protocol	Length	Info
10	8.184051733	185.125.190.58	192.168.28.131	NTP	92	NTP Version 4, server
11	13.094325910	VMware_f9:a7:51		ARP	44	Who has 192.168.28.2? Tell 192.168.28.131
12	13.094545343	VMware_e9:2b:34		ARP	62	192.168.28.2 is at 00:50:56:e9:2b:34
13	22.326867175	02:42:ac:11:00:02		ARP	44	Who has 172.17.0.5? Tell 172.17.0.2
14	22.326899513	02:42:ac:11:00:02		ARP	44	Who has 172.17.0.5? Tell 172.17.0.2
15	22.326902895	02:42:ac:11:00:02		ARP	44	Who has 172.17.0.5? Tell 172.17.0.2
16	22.326905308	02:42:ac:11:00:02		ARP	44	Who has 172.17.0.5? Tell 172.17.0.2
17	22.326867175	02:42:ac:11:00:02		ARP	44	Who has 172.17.0.5? Tell 172.17.0.2
18	22.326983699	02:42:ac:11:00:05		ARP	44	172.17.0.5 is at 02:42:ac:11:00:05
19	22.327002455	02:42:ac:11:00:05		ARP	44	172.17.0.5 is at 02:42:ac:11:00:05
20	22.327011541	172.17.0.2	172.17.0.5	TCP	76	38666 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=943447725 TSecr=0 WS=128
21	22.327012589	172.17.0.2	172.17.0.5	TCP	76	[TCP Out-Of-Order] [TCP Port numbers reused] 38666 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=9434477...
22	22.327019343	172.17.0.5	172.17.0.2	TCP	76	22 → 38666 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3736201389 TSecr=943447725 WS=128
23	22.327923694	172.17.0.5	172.17.0.2	TCP	76	[TCP Out-Of-Order] 22 → 38666 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3736201389 TSecr=94344...
24	22.327967671	172.17.0.2	172.17.0.5	TCP	68	38666 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=943447726 TSecr=3736201389
25	22.327968948	172.17.0.2	172.17.0.5	TCP	68	[TCP Dup ACK 24#1] 38666 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=943447726 TSecr=3736201389
26	22.336755791	172.17.0.2	172.17.0.5	SSHv2	109	Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)
27	22.336762455	172.17.0.2	172.17.0.5	TCP	109	[TCP Retransmission] 38666 → 22 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=41 TSval=943447735 TSecr=3736201389
28	22.336846617	172.17.0.5	172.17.0.2	TCP	68	22 → 38666 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3736201398 TSecr=943447735
29	22.336850262	172.17.0.5	172.17.0.2	TCP	68	[TCP Dup ACK 28#1] 22 → 38666 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3736201398 TSecr=943447735
30	22.344368638	172.17.0.5	172.17.0.2	SSHv2	109	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
31	22.344373197	172.17.0.5	172.17.0.2	TCP	109	[TCP Retransmission] 22 → 38666 [PSH, ACK] Seq=1 Ack=42 Win=65152 Len=41 TSval=3736201406 TSecr=943447735
32	22.344452098	172.17.0.2	172.17.0.5	TCP	68	38666 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=943447743 TSecr=3736201406
33	22.344455176	172.17.0.2	172.17.0.5	TCP	68	[TCP Dup ACK 32#1] 38666 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=943447743 TSecr=3736201406
34	22.345223055	172.17.0.2	172.17.0.5	SSHv2	1500	Client: Key Exchange Init
35	22.345226956	172.17.0.2	172.17.0.5	TCP	1500	[TCP Retransmission] 38666 → 22 [PSH, ACK] Seq=42 Ack=42 Win=64256 Len=1432 TSval=943447743 TSecr=3736201406
36	22.352664701	172.17.0.5	172.17.0.2	SSHv2	1148	Server: Key Exchange Init
37	22.352669083	172.17.0.5	172.17.0.2	TCP	1148	[TCP Retransmission] 22 → 38666 [PSH, ACK] Seq=42 Ack=1474 Win=64128 Len=1080 TSval=3736201414 TSecr=943447743
38	22.357103564	172.17.0.2	172.17.0.5	SSHv2	116	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
39	22.357107080	172.17.0.2	172.17.0.5	TCP	116	[TCP Retransmission] 38666 → 22 [PSH, ACK] Seq=1474 Ack=1122 Win=64128 Len=48 TSval=943447755 TSecr=3736201414
40	22.370750723	172.17.0.5	172.17.0.2	SSHv2	664	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
41	22.370755867	172.17.0.5	172.17.0.2	TCP	664	[TCP Retransmission] 22 → 38666 [PSH, ACK] Seq=1122 Ack=1522 Win=64128 Len=596 TSval=3736201432 TSecr=943447755
42	22.411462403	172.17.0.2	172.17.0.5	TCP	68	38666 → 22 [ACK] Seq=1522 Ack=1718 Win=64128 Len=0 TSval=943447810 TSecr=3736201432
43	22.411471117	172.17.0.2	172.17.0.5	TCP	68	[TCP Dup ACK 42#1] 38666 → 22 [ACK] Seq=1522 Ack=1718 Win=64128 Len=0 TSval=943447810 TSecr=3736201432
44	27.430787177	02:42:ac:11:00:05		ARP	44	Who has 172.17.0.2? Tell 172.17.0.5
45	27.430894231	02:42:ac:11:00:05		ARP	44	Who has 172.17.0.2? Tell 172.17.0.5
46	27.430923118	02:42:ac:11:00:02		ARP	44	172.17.0.2 is at 02:42:ac:11:00:02
47	27.430929298	02:42:ac:11:00:02		ARP	44	172.17.0.2 is at 02:42:ac:11:00:02
48	27.591655777	172.17.0.2	172.17.0.5	SSHv2	84	Client: New Keys

Figura 7: Trafico capturado cliente 1

Como se puede observar, en el paquete número 20 comienza el *three-way handshake*, que marca el inicio de la conexión al servidor. Este proceso inicia con un paquete SYN (Synchronize), enviado a través del protocolo TCP desde el cliente hacia el servidor ubicado en la dirección 172.117.0.5. Posteriormente, el servidor responde con un paquete SYN-ACK (Synchronize-Acknowledge), y finalmente, el cliente completa el intercambio con un paquete ACK (Acknowledge). Este procedimiento establece una conexión confiable entre ambos extremos.

A continuación, se observa cómo tanto el cliente como el servidor intercambian información sobre el protocolo de seguridad y las versiones de OpenSSH utilizadas. En este caso, se utiliza la versión SSH-2.0-OpenSSH-7.3p1 Ubuntu-1ubuntu0.1. Finalmente, se detalla el uso del algoritmo Diffie-Hellman para el intercambio de claves, garantizando la generación de una clave compartida segura entre cliente y servidor.

Para obtener el HASSH, se debe revisar los paquetes generados durante el intercambio de claves. En la sección del protocolo SSH, se encuentra el siguiente valor de HASSH generado por el cliente: `0e4584cb9f2dd077dbf8ba0df8112d8e`. Este valor corresponde a un hash único generado como parte del proceso de autenticación y establecimiento de la conexión segura.

```
First KEX Packet Follows: 0
Reserved: 00000000
[hasshAlgorithms [truncated]: curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sh
[hassh: 0e4584cb9f2dd077dbf8ba0df8112d8e]
ng String: 00000000000000000000000
```

Figura 8: HASSH cliente 1

Ahora, centrándonos en los tamaños generados en los paquetes, se puede notar que tanto el inicio del *three-way handshake* como la respuesta, es decir, el paquete SYN y el paquete SYN-ACK, tienen un tamaño de 76 bytes, mientras que el paquete ACK final tiene un tamaño de 68 bytes. Por otro lado, el inicio de la conexión de OpenSSH tiene un tamaño de 109 bytes, el intercambio de llaves tiene un tamaño de 1500 bytes desde el cliente y 1148 bytes desde el servidor, mientras que el inicio de Diffie-Hellman tiene un tamaño de 116 bytes y la réplica de Diffie-Hellman es de 664 bytes.

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

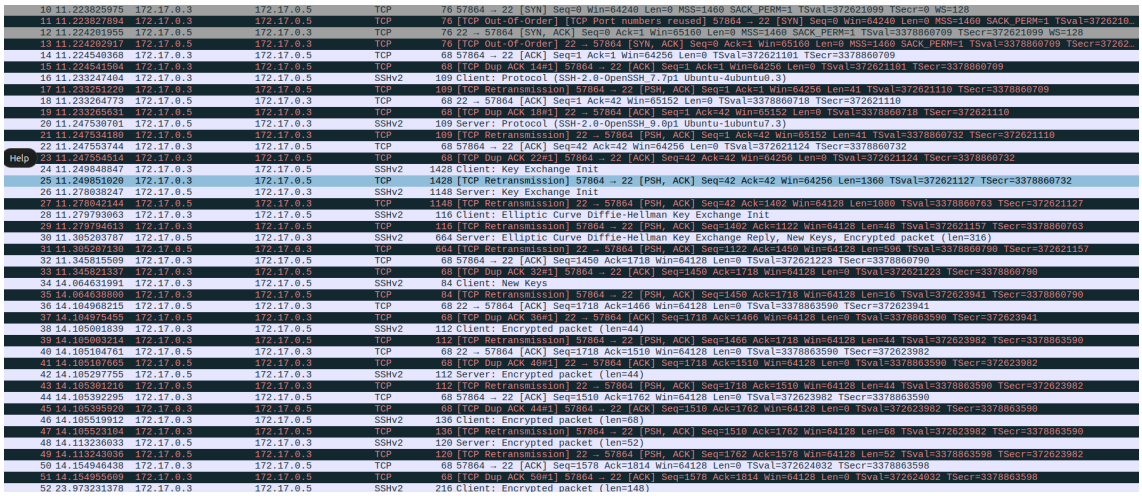


Figura 9: Trafico capturado cliente 2

En el paquete número 10 comienza el *three-way handshake*, como se puede observar, tanto el paquete SYN como el SYN-ACK tienen un tamaño de 76 bytes, mientras que el paquete ACK final tiene un tamaño de 68 bytes. Posteriormente, se observa el inicio de la conexión de OpenSSH, con un tamaño de 109 bytes, donde se indica la versión del protocolo, que en este caso es `SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3`. La estructura es similar a la captura en el cliente 1.

Luego, comienza el intercambio de llaves, donde la solicitud del cliente tiene un tamaño de 1428 bytes, mientras que la respuesta del servidor es de 1148 bytes. Finalmente, se inicia el intercambio Diffie-Hellman, con un tamaño de 116 bytes desde el cliente y 664 bytes desde el servidor.

Por otro lado, al momento de buscar el HASSH, se encuentra el siguiente valor: `06046964c022c6407d1`

```
[hasshAlgorithms [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hell
[hassh: 06046964c022c6407d15a27b12a6a4fb]
```

Figura 10: HASSH cliente 2

Como se puede observar los pasos son igual a los de la captura anterior

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

Figura 11: Trafico capturado cliente 3

Como se puede observar, nuevamente los dos primeros mensajes del *three-way handshake* son de 76 bytes, mientras que el paquete ACK final tiene un tamaño de 68 bytes. Este es el proceso estándar de establecimiento de conexión TCP, que asegura la confiabilidad de la comunicación entre el cliente y el servidor.

Una vez establecido el canal de comunicación, se inicia la conexión mediante OpenSSH. El primer mensaje de esta fase tiene un tamaño de 109 bytes, y es importante notar que la versión del protocolo SSH utilizada por el cliente ha cambiado nuevamente, pasando a ser `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`. En cuanto a la respuesta del servidor, también tiene un tamaño de 109 bytes, pero la versión del protocolo del servidor se mantiene constante, siendo `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`.

Posteriormente, comienza el intercambio de llaves, que es una fase crítica para establecer una conexión segura. En este caso, la solicitud enviada por el cliente tiene un tamaño de 1580 bytes, mientras que la respuesta del servidor es de 1148 bytes.

Finalmente, se inicia el intercambio mediante el algoritmo Diffie-Hellman. En este paso, la solicitud del cliente tiene un tamaño de 116 bytes, mientras que la respuesta del servidor es de 664 bytes.

El HASSH obtenido es `ae8bd7dd09970555aa4c6ed22adbbf56`

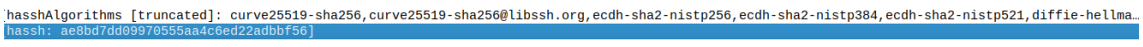
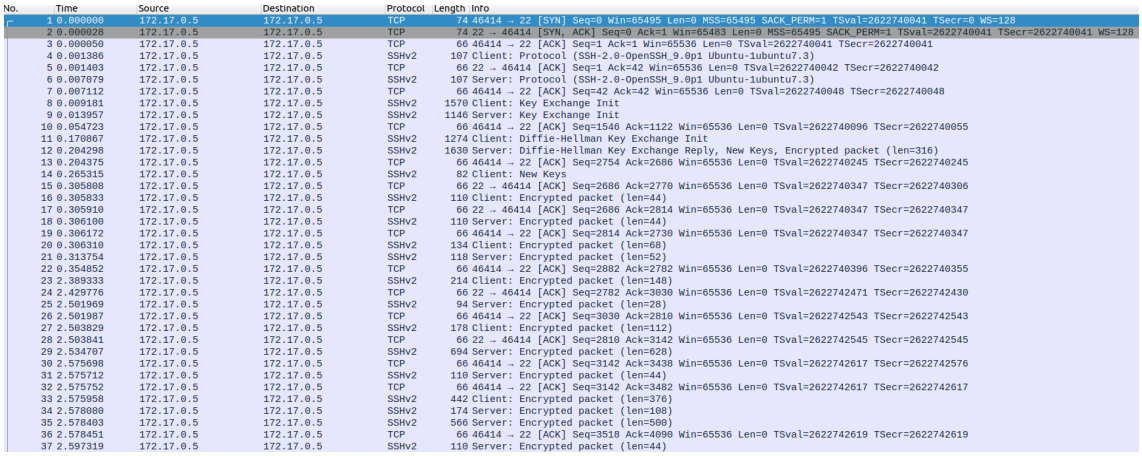


Figura 12: HASSH cliente 3

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



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.17.0.5	172.17.0.5	TCP	74	46414 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2622740041 TSecr=0 WS=128
2	0.000028	172.17.0.5	172.17.0.5	TCP	74	22 → 46414 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2622740041 TSecr=2622740041 WS=128
3	0.000050	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2622740041 TSecr=2622740041
4	0.001386	172.17.0.5	172.17.0.5	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
5	0.001403	172.17.0.5	172.17.0.5	TCP	66	22 → 46414 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=2622740042 TSecr=2622740042
6	0.007079	172.17.0.5	172.17.0.5	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7	0.007112	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSval=2622740048 TSecr=2622740048
8	0.009181	172.17.0.5	172.17.0.5	SSHv2	1570	Client: Key Exchange Init
9	0.013957	172.17.0.5	172.17.0.5	SSHv2	1146	Server: Key Exchange Init
10	0.054723	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=1546 Ack=1122 Win=65536 Len=0 TSval=2622740096 TSecr=2622740055
11	0.170867	172.17.0.5	172.17.0.5	SSHv2	1274	Client: Diffie-Hellman Key Exchange Init
12	0.204298	172.17.0.5	172.17.0.5	SSHv2	1630	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
13	0.204375	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=2754 Ack=2686 Win=65536 Len=0 TSval=2622740245 TSecr=2622740245
14	0.265315	172.17.0.5	172.17.0.5	SSHv2	82	Client: New Keys
15	0.305808	172.17.0.5	172.17.0.5	TCP	66	22 → 46414 [ACK] Seq=2686 Ack=2770 Win=65536 Len=0 TSval=2622740347 TSecr=2622740306
16	0.305833	172.17.0.5	172.17.0.5	SSHv2	110	Client: Encrypted packet (len=44)
17	0.305910	172.17.0.5	172.17.0.5	TCP	66	22 → 46414 [ACK] Seq=2686 Ack=2814 Win=65536 Len=0 TSval=2622740347 TSecr=2622740347
18	0.306100	172.17.0.5	172.17.0.5	SSHv2	110	Server: Encrypted packet (len=44)
19	0.306172	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=2814 Ack=2730 Win=65536 Len=0 TSval=2622740347 TSecr=2622740347
20	0.306310	172.17.0.5	172.17.0.5	SSHv2	134	Client: Encrypted packet (len=68)
21	0.313754	172.17.0.5	172.17.0.5	SSHv2	118	Server: Encrypted packet (len=52)
22	0.354852	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=2882 Ack=2782 Win=65536 Len=0 TSval=2622740396 TSecr=2622740355
23	2.389333	172.17.0.5	172.17.0.5	SSHv2	214	Client: Encrypted packet (len=148)
24	2.429776	172.17.0.5	172.17.0.5	TCP	66	22 → 46414 [ACK] Seq=2782 Ack=3030 Win=65536 Len=0 TSval=2622742471 TSecr=2622742430
25	2.501969	172.17.0.5	172.17.0.5	SSHv2	94	Server: Encrypted packet (len=28)
26	2.501987	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=3030 Ack=2810 Win=65536 Len=0 TSval=2622742543 TSecr=2622742543
27	2.503829	172.17.0.5	172.17.0.5	SSHv2	178	Client: Encrypted packet (len=112)
28	2.503841	172.17.0.5	172.17.0.5	TCP	66	22 → 46414 [ACK] Seq=2810 Ack=3142 Win=65536 Len=0 TSval=2622742545 TSecr=2622742545
29	2.534707	172.17.0.5	172.17.0.5	SSHv2	694	Server: Encrypted packet (len=628)
30	2.575698	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=3142 Ack=3438 Win=65536 Len=0 TSval=2622742617 TSecr=2622742576
31	2.575712	172.17.0.5	172.17.0.5	SSHv2	110	Server: Encrypted packet (len=44)
32	2.575752	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=3142 Ack=3482 Win=65536 Len=0 TSval=2622742617 TSecr=2622742617
33	2.575958	172.17.0.5	172.17.0.5	SSHv2	442	Client: Encrypted packet (len=376)
34	2.578080	172.17.0.5	172.17.0.5	SSHv2	174	Server: Encrypted packet (len=108)
35	2.578403	172.17.0.5	172.17.0.5	SSHv2	566	Server: Encrypted packet (len=500)
36	2.578451	172.17.0.5	172.17.0.5	TCP	66	46414 → 22 [ACK] Seq=3518 Ack=4090 Win=65536 Len=0 TSval=2622742619 TSecr=2622742619
37	2.597319	172.17.0.5	172.17.0.5	SSHv2	110	Server: Encrypted packet (len=44)

Figura 13: Trafico capturado cliente 4

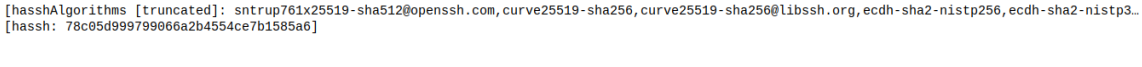
Para capturar el tráfico generado por el contenedor **C4**, se utilizó *tshark* en lugar de Wireshark, ya que este último no lograba capturar tráfico dentro de un mismo contenedor. Al analizar los datos recopilados, se pueden observar los siguientes detalles:

1. Durante el establecimiento de la conexión mediante el *three-way handshake*, tanto el paquete `SYN` como el `SYN-ACK` tienen un tamaño de **74 bytes**, mientras que el `ACK` final es de **66 bytes**.

2. Cuando inicia la conexión de *OpenSSH* desde el cliente, el paquete correspondiente tiene un tamaño de **107 bytes**, y la versión del protocolo utilizada es `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`. Es importante destacar que esta versión del protocolo se mantiene constante también en el servidor. La respuesta del servidor a esta conexión tiene un tamaño de **107 bytes**, lo que refleja una simetría en esta etapa del intercambio.

3. Posteriormente, durante el intercambio de claves, el paquete enviado desde el cliente tiene un tamaño de **1570 bytes**, mientras que el paquete del servidor es de **1146 bytes**.

4. Finalmente, en el intercambio mediante el algoritmo de *Diffie-Hellman*, el inicio de la comunicación tiene un tamaño de **1274 bytes**, mientras que la réplica enviada desde el servidor tiene un tamaño de **1630 bytes**.



```
[hasshAlgorithms [truncated]: sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp3…
[hassh: 78c05d999799066a2b4554ce7b1585a6]
```

Figura 14: HASSH cliente 4

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

1.7.1. C1

En la información contenida en la captura desde el cliente **C1**, en el inicio del *three-way handshake*, se puede notar que se envían datos básicos como el puerto de destino (`38666`) y el puerto `22`, este último expuesto al servidor mediante el *Dockerfile*. El resto del *three-way handshake* incluye información típica del tráfico de paquetes TCP, como los puertos, las *flags* asociadas, y otros datos esenciales para el establecimiento de la conexión.

Cuando comienza la conexión entre el cliente y el servidor a través de OpenSSH, se observa que el paquete contiene un nuevo campo correspondiente al protocolo SSH. Este campo proporciona información valiosa como la versión utilizada y la dirección del paquete, indicando si se origina desde el cliente hacia el servidor o viceversa.

```
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window: 502
    [Calculated window size: 64256]
    [Window size scaling factor: 128]
    Checksum: 0x5879 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [Timestamps]
  ▾ [SEQ/ACK analysis]
      [iRTT: 0.000955082 seconds]
      [Bytes in flight: 41]
      [Bytes sent since last PSH flag: 41]
    TCP payload (41 bytes)
▾ SSH Protocol
    Protocol: SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1
    [Direction: client-to-server]
```

Figura 15: Inicio de conexión por OpenSSH.

Posteriormente, en el intercambio de llaves, se puede notar una mayor cantidad de información en el campo del protocolo SSH. Este incluye detalles sobre los algoritmos utilizados, la *Cookie*, el tamaño de las claves y de los algoritmos empleados, así como el valor de *HASSH*, entre otros datos significativos. Además, se muestra un mensaje que especifica si se trata del inicio del intercambio o de una réplica asociada.

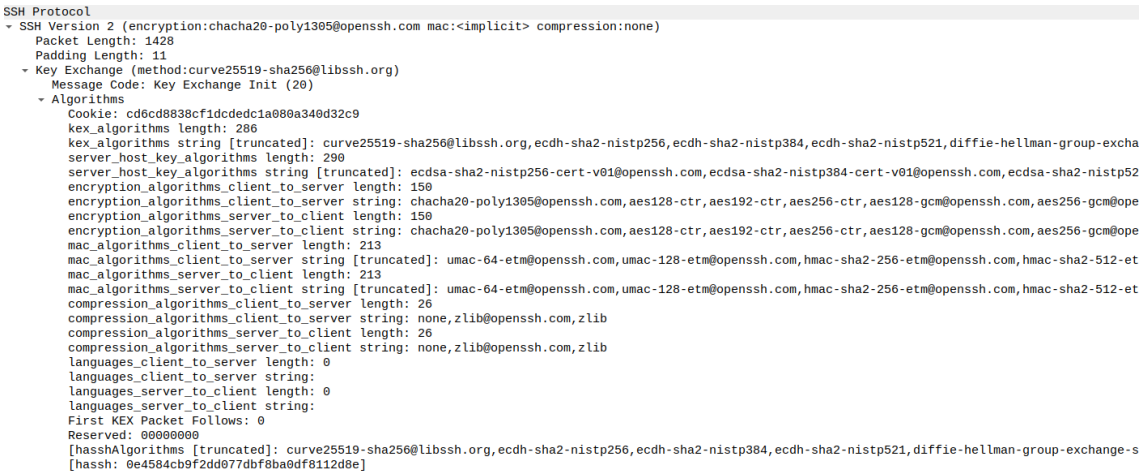


Figura 16: Paquete capturado en el intercambio de llaves.

Finalmente, durante el intercambio mediante el algoritmo *Diffie-Hellman*, se detalla información clave, como el tamaño del paquete, el tamaño del *padding*, y un mensaje que indica el inicio del intercambio de claves mediante *Elliptic Curve Diffie-Hellman*. También se incluye un mensaje con la dirección del paquete y la longitud de la clave pública utilizada.

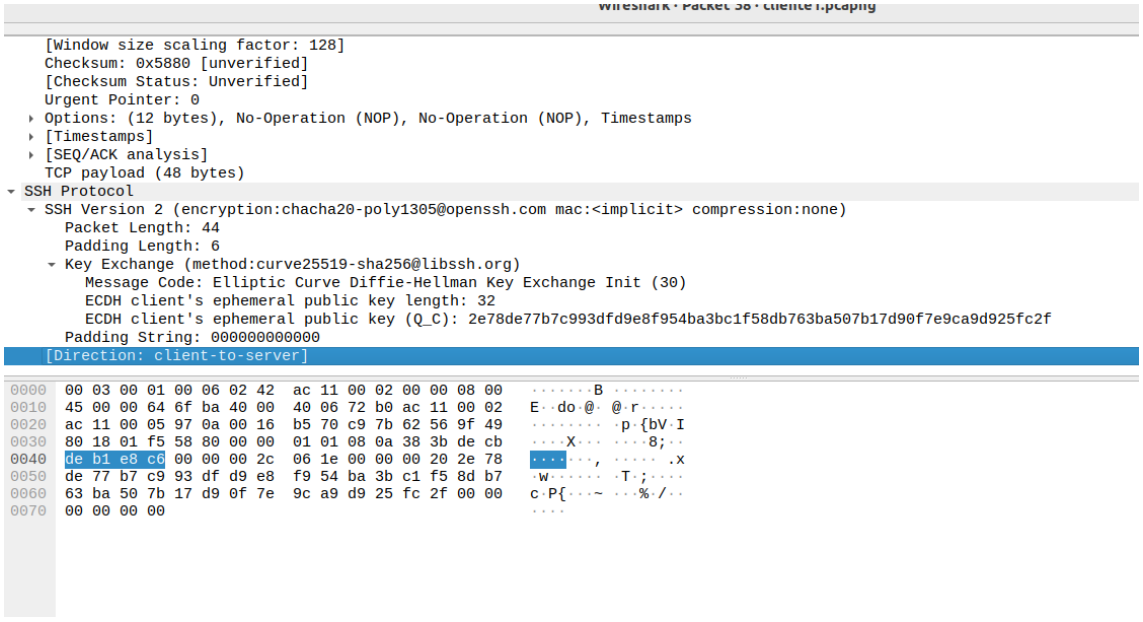


Figura 17: Paquetes capturados durante el intercambio mediante *Diffie-Hellman*.

1.7.2. C2

La información presentada en el tráfico capturado desde el cliente **C2** es consistente con la observada anteriormente. Por esta razón, se incluyen nuevamente las imágenes correspondien-

tes para referencia visual, mientras que las diferencias incrementales detectadas se detallarán en la siguiente sección.

```
▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
▸ [Timestamps]
▸ [SEQ/ACK analysis]
  TCP payload (41 bytes)
SSH Protocol
  Protocol: SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3
  [Direction: client-to-server]
```

Figura 18: Inicio de conexión por OpenSSH cliente 2.

```
SSH Protocol
▾ SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    Packet Length: 1356
    Padding Length: 5
  ▾ Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
    ▾ Algorithms
        Cookie: d943377677059641 69e2b1af0fd2c12a
        kex_algorithms length: 304
        kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hel
        server_host_key_algorithms length: 290
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@opens
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@opens
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@
        compression_algorithms_client_to_server length: 26
        compression_algorithms_client_to_server string: none,zlib@openssh.com,zlib
        compression_algorithms_server_to_client length: 26
        compression_algorithms_server_to_client string: none,zlib@openssh.com,zlib
        languages_client_to_server length: 0
        languages_client_to_server string:
        languages_server_to_client length: 0
        languages_server_to_client string:
        First KEX Packet Follows: 0
        Reserved: 00000000
        [hasshAlgorithms [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-
        [hassh: 06046964c022c6407d15a27b12a6a4fb]
    Padding String: 0000000000
  [Direction: client-to-server]
```

Figura 19: Paquete capturado en el intercambio de llaves.

```
  TCP payload (48 bytes)
SSH Protocol
▾ SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    Packet Length: 44
    Padding Length: 6
  ▾ Key Exchange (method:curve25519-sha256)
      Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
      ECDH client's ephemeral public key length: 32
      ECDH client's ephemeral public key (Q_C): 61a78db194b5403b2f7af7b25dc4630ff9fd11d810f0746c42c6667f40f7405a
    Padding String: 000000000000
  [Direction: client-to-server]
```

Figura 20: Paquetes capturados durante el intercambio mediante *Diffie-Hellman*.

1.7.3. C3

Siguiendo el patron anterior, solo se presentaran las imagenes correspondientes.

```
[Next Sequence Number: 42    (relative sequence number)]
Acknowledgment Number: 42    (relative ack number)
Acknowledgment number (raw): 1309079957
1000 .... = Header Length: 32 bytes (8)
▸ Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x587b [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
▸ [Timestamps]
▸ [SEQ/ACK analysis]
TCP payload (41 bytes)
▾ SSH Protocol
Protocol: SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1
[Direction: client-to-server]
```

Figura 21: Inicio de conexión por OpenSSH cliente 3.

```
SSH Protocol
▾ SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    Packet Length: 1508
    Padding Length: 10
  ▾ Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
    ▾ Algorithms
        Cookie: a1240257693be630867b0e915fdd65d4
        kex_algorithms length: 241
        kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-he
        server_host_key_algorithms length: 500
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@opens
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@opens
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@
        compression_algorithms_client_to_server length: 26
        compression_algorithms_client_to_server string: none,zlib@openssh.com,zlib
        compression_algorithms_server_to_client length: 26
        compression_algorithms_server_to_client string: none,zlib@openssh.com,zlib
        languages_client_to_server length: 0
        languages_client_to_server string:
        languages_server_to_client length: 0
        languages_server_to_client string:
        First KEX Packet Follows: 0
        Reserved: 00000000
        [hasshAlgorithms [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-
        [hassh: ae8bd7dd09970555aa4c6ed22adbbf56]
    Padding String: 00000000000000000000
  [Direction: client-to-server]
```

Figura 22: Paquete capturado en el intercambio de llaves.

```
SSH Protocol
▾ SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    Packet Length: 44
    Padding Length: 6
  ▾ Key Exchange (method:curve25519-sha256)
      Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
      ECDH client's ephemeral public key length: 32
      ECDH client's ephemeral public key (Q_C): 2054530aa59309e4221f487a5e4f1829235be0d0c443256601c05f5d129f1a2a
    Padding String: 000000000000
  [Direction: client-to-server]
```

Figura 23: Paquetes capturados durante el intercambio mediante *Diffie-Hellman*.

1.7.4. C4/S1

Ahora se sigue lo hecho anteriormente, aunque en las capturas se pueden ver diferencias evidentes en comparación a los 3 clientes anteriores, estas diferencias se discutan con mayor precisión en la siguiente sección.

```
SSH Protocol
  Protocol: SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
  [Direction: client-to-server]
```

Figura 24: Inicio de conexión por OpenSSH cliente 4.

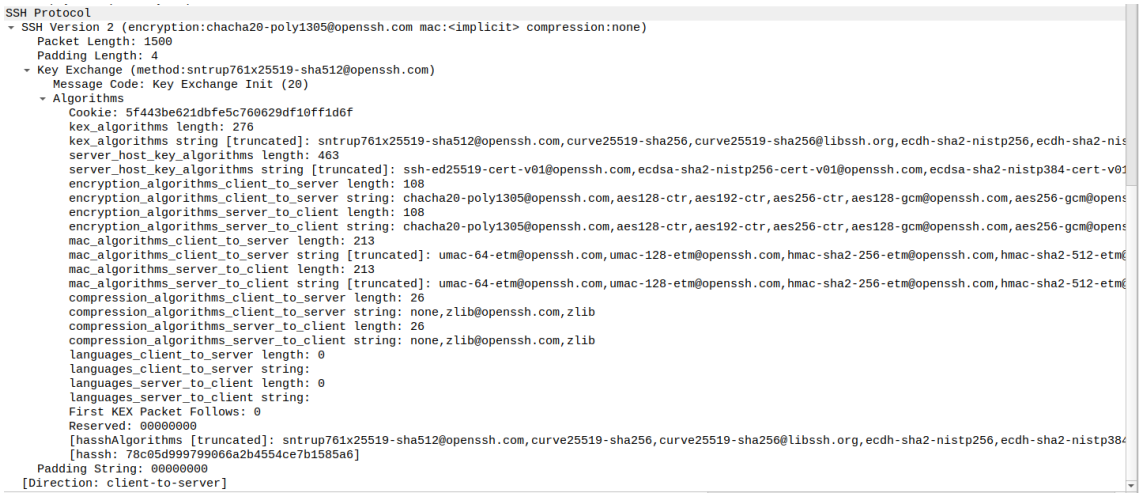


Figura 25: Paquete capturado en el intercambio de llaves.

```
SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    Packet Length: 1204
    Padding Length: 8
    Key Exchange (method:sntrup761x25519-sha512@openssh.com)
      Message Code: Diffie-Hellman Key Exchange Init (30)
      Multi Precision Integer Length: 1190
      DH client e: c631d41f27931146201dbd0d731c8ccc8664ce01fb2caa47c4193f278e889b52fd42bbc0…
    Padding String: 0000000000000000
  [Direction: client-to-server]
```

Figura 26: Paquetes capturados durante el intercambio mediante *Diffie-Hellman*.

1.8. Diferencia entre C1 y C2

Al analizar las diferencias entre los clientes **C1** y **C2**, se identificaron las siguientes variaciones:

La primera diferencia notable se encuentra en la versión del protocolo utilizada en el cliente. En **C2**, se emplea una versión posterior en comparación con la utilizada en **C1**.

En el caso del *three-way handshake*, los tamaños de los paquetes se mantuvieron consistentes en ambos clientes, siendo de 76 bytes para los paquetes SYN y SYN-ACK, y de 68 bytes para el ACK final. Asimismo, la cantidad de bytes en los paquetes de inicio de la conexión con OpenSSH fue idéntica, alcanzando 109 bytes.

En el intercambio mediante *Diffie-Hellman*, los tamaños también se mantuvieron constantes: 116 bytes para el inicio desde el cliente y 664 bytes para la respuesta del servidor. Sin embargo, se observó una diferencia en el tamaño de los paquetes durante el intercambio de llaves. En **C2**, el paquete inicial tuvo un tamaño de 1428 bytes, menor en comparación con los 1500 bytes registrados en **C1**. Por otro lado, la respuesta del servidor permaneció constante en ambos casos.

En cuanto a la información contenida en cada paquete, se constató que el método utilizado para el intercambio de llaves es el mismo en ambos clientes. No obstante, se observó un incremento en el tamaño del campo `kex_algorithms` en los paquetes de **C2**, indicando un mayor número o variedad de algoritmos de intercambio de claves.

Finalmente, al analizar los paquetes correspondientes a *Diffie-Hellman*, no se identificaron diferencias significativas entre ambos clientes, ya que los paquetes presentan una estructura y contenido muy similares.

1.9. Diferencia entre C2 y C3

En la captura correspondiente al cliente **C3**, se observa que el patrón general se mantiene similar a los casos anteriores. Todo el proceso del *three-way handshake* conserva los mismos tamaños en los paquetes: 76 bytes para el SYN y el SYN-ACK, y 68 bytes para el ACK final.

En el inicio de la conexión con OpenSSH, se identifica nuevamente una diferencia en la versión del protocolo del cliente. En **C3**, se utiliza una versión más reciente en comparación con la empleada en **C2**. A pesar de este cambio, el tamaño en bytes del paquete inicial de la conexión permanece constante en 109 bytes, tanto para el cliente como para el servidor.

En cuanto al intercambio de llaves, se observa un incremento en el tamaño del paquete inicial enviado por el cliente, que pasó de 1428 bytes en **C2** a 1580 bytes en **C3**. La respuesta del servidor, como es de esperarse, se mantiene igual, reflejando la consistencia del servidor en los procesos de intercambio de claves.

Al analizar los tamaños de los paquetes de *Diffie-Hellman*, se nota que estos también permanecen constantes, sin cambios respecto a los clientes anteriores.

Sin embargo, al observar el contenido de los paquetes en texto plano, se identifican algunas diferencias: - En el paquete del intercambio de llaves, el campo `packet_length` incrementó su tamaño a 1508. - El método de intercambio de llaves se mantiene igual, pero el tamaño del campo `kex_algorithms` disminuyó. - El campo `server_host_key_algorithms` experimentó un aumento en su tamaño, pasando de 290 a 500 bytes.

Por último, al analizar los paquetes correspondientes a *Diffie-Hellman*, se observa que estos conservan la misma estructura y tamaños. La única diferencia notable es la clave pública, que varía entre cada cliente, como era de esperarse en un proceso de generación de claves dinámico y único.

1.10. Diferencia entre C3 y C4

En la captura correspondiente al cliente **C4**, se observa una leve disminución en el tamaño de los paquetes durante el proceso de *three-way handshake*, donde cada paquete enviado por

el cliente presenta una reducción de 2 bytes.

Por otro lado, se identifica un cambio en la versión del protocolo utilizado, que coincide con la versión del servidor. Este comportamiento es esperado, ya que ambos (cliente y servidor) están instalados en el mismo contenedor, lo que garantiza que empleen la misma versión.

En el inicio del intercambio de llaves, el tamaño del paquete inicial enviado por el cliente disminuye en 10 bytes, mientras que la respuesta del servidor se reduce en 2 bytes.

Al analizar los paquetes correspondientes al intercambio mediante *Diffie-Hellman*, se nota un cambio significativo: - El nombre del método ya no hace referencia a curvas elípticas (*elliptic curve*), como en los clientes anteriores. - El tamaño de los paquetes aumenta considerablemente: el paquete inicial del cliente pasa de 116 bytes a 1274 bytes, y la respuesta del servidor incrementa de 664 bytes a 1630 bytes.

En el análisis en texto plano de los paquetes: - Durante el intercambio de llaves, el método cambia de `curve25519-sha256` a `sntrup761x25519-sha512@openssh.com`. - El tamaño del campo `kex_algorithms` también aumenta. - La clave del `server_host_key` muestra una reducción en su tamaño, pasando a 463 bytes. - El resto de los campos se mantienen constantes.

En cuanto a los paquetes de *Diffie-Hellman*, se observa un cambio drástico en los datos presentes: - El campo `ECDH client's ephemeral public key` desaparece y es reemplazado por un atributo denominado `multi-precision integer length` y `DH client e`. - Los campos de `padding` y la dirección del mensaje permanecen constantes.

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión "?"

Observando detenidamente la captura de tráfico, se puede notar que casi todos los paquetes registrados por Wireshark tienen un incremento de 2 bytes en su tamaño. Esto podría llevar a la conclusión errónea sobre ciertos detalles, como en la última observación realizada para el cliente **C4**, donde inicialmente se asumió una disminución en el tamaño de los paquetes del *three-way handshake*. Sin embargo, considerando este incremento generalizado, el tamaño de estos paquetes se mantendría constante respecto a los clientes anteriores.

Ahora, analizando el tráfico en cuestión y suponiendo que se agregan esos 2 bytes adicionales a todos los paquetes, el cliente que generó este tráfico podría ser **C3**, ya que el tamaño registrado durante el intercambio de llaves coincidiría de manera exacta con lo esperado para este cliente.

Aunque los tamaños también podrían coincidir con **C4**, es importante recordar que el método de *Diffie-Hellman* cambia entre estos clientes. En el tráfico capturado, se observa el uso de una curva elíptica como método, lo cual corresponde al cliente **C3**. Por lo tanto, es razonable concluir que el cliente responsable de este tráfico capturado es efectivamente **C3**.

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

Con el objetivo de replicar y analizar el tráfico generado por el cliente C3, se desarrolló un script en Python utilizando la biblioteca *Scapy*. Este código permite modificar la captura de tráfico SSH previamente generada, alterando la versión reportada del protocolo e incluyendo un signo de interrogación en lugar de la versión completa. Asimismo, se ajustó el tamaño de los paquetes capturados a un valor fijo de 85 bytes, con el fin de estandarizar la estructura de los datos y facilitar el análisis.

El proceso comienza leyendo el archivo de captura original (`cliente3.pcapng`) y filtrando los paquetes correspondientes al protocolo SSH que contienen la cadena específica de la versión `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`. Posteriormente, se verifica si estos paquetes pertenecen al cliente identificado mediante su dirección IP. Si ambas condiciones se cumplen, se reemplaza la cadena mencionada por `SSH-2.0-OpenSSH_?` y se ajusta la carga útil del paquete al tamaño deseado. Este ajuste asegura que la longitud del paquete sea exactamente 85 bytes, rellenando con ceros si es necesario o truncando los datos en caso de ser demasiado largos.

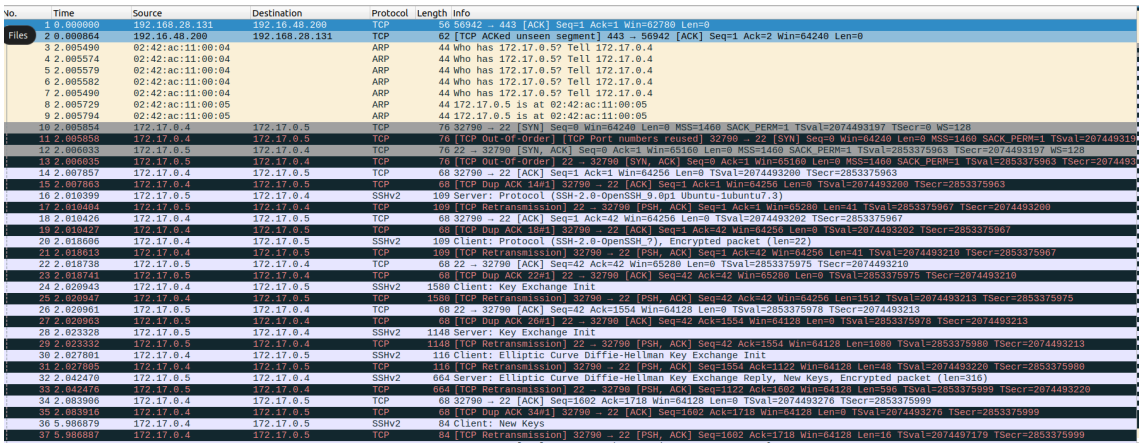
Una vez realizadas las modificaciones, los campos relacionados con la longitud y el checksum son eliminados para que puedan ser recalculados automáticamente al guardar el archivo de captura modificado (`mod.pcapng`). Este proceso asegura que el tráfico sea válido y pueda ser interpretado por herramientas de análisis como Wireshark o *tshark*.

Como se observa en la imagen 28, el protocolo cambia de acuerdo con la modificación realizada, reflejando el nuevo identificador del cliente. Sin embargo, el tamaño final del paquete se mantuvo constante.

modcap.py > ...

```python
from scapy.all import rdpcap, wrpcap, Raw, IP, TCP

# Archivos de entrada y salida
pcap_original = 'cliente3.pcapng'
pcap_mod = 'mod.pcapng'

# Dirección IP del cliente
ip_cliente = '172.17.0.5'

# Leer los paquetes del archivo PCAP original
paquetes = rdpcap(pcap_original)

# Longitud deseada para el paquete
tamaño_deseado = 85

# Lista para guardar los paquetes modificados
paquetes_modificados = []

# Iterar sobre los paquetes
for paquete in paquetes:
    # Verificar si el paquete tiene la capa Raw y contiene datos
    if paquete.haslayer(Raw) and paquete[Raw].load:
        # Verificar si contiene la cadena específica
        if b'SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1' in paquete[Raw].load:
            # Verificar si la dirección IP coincide
            if paquete[IP].src == ip_cliente:
                # Modificar la carga útil
                paquete[Raw].load = paquete[Raw].load.replace(
                    b'SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1',
                    b'SSH-2.0-OpenSSH_?'
                )

                # Ajustar el tamaño del paquete a 85 bytes
                payload = paquete[Raw].load
                if len(payload) > tamaño_deseado:
                    paquete[Raw].load = payload[:tamaño_deseado]
                else:
                    paquete[Raw].load = payload.ljust(tamaño_deseado, b'\x00')

                # Eliminar campos que deben recalcularse
                del paquete[IP].len
                del paquete[IP].chksum
                if paquete.haslayer(TCP):
                    del paquete[TCP].chksum

    # Agregar el paquete (modificado o no) a la lista
    paquetes_modificados.append(paquete)

# Escribir los paquetes modificados en el archivo de salida
wrpcap(pcap_mod, paquetes_modificados)
```

Figura 27: Código desarrollado para modificar el tráfico capturado.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.28.131	192.16.48.200	TCP	56	56942 → 443 [ACK] Seq=1 Ack=1 Win=62780 Len=0
2	0.000864	192.16.48.200	192.168.28.131	TCP	62	[TCP ACKed unseen segment] 443 → 56942 [ACK] Seq=1 Ack=2 Win=64240 Len=0
3	2.005490	02:42:ac:11:00:04		ARP	44	Who has 172.17.0.5? Tell 172.17.0.4
4	2.005574	02:42:ac:11:00:04		ARP	44	Who has 172.17.0.5? Tell 172.17.0.4
5	2.005579	02:42:ac:11:00:04		ARP	44	Who has 172.17.0.5? Tell 172.17.0.4
6	2.005582	02:42:ac:11:00:04		ARP	44	Who has 172.17.0.5? Tell 172.17.0.4
7	2.005490	02:42:ac:11:00:04		ARP	44	Who has 172.17.0.5? Tell 172.17.0.4
8	2.005729	02:42:ac:11:00:05		ARP	44	172.17.0.5 is at 02:42:ac:11:00:05
9	2.005794	02:42:ac:11:00:05		ARP	44	172.17.0.5 is at 02:42:ac:11:00:05
10	2.005854	172.17.0.4	172.17.0.5	TCP	76	32790 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2074493197 TSecr=0 WS=128
11	2.005858	172.17.0.4	172.17.0.5	TCP	76	[TCP Out-Of-Order] [TCP Port numbers reused] 32790 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=207449319
12	2.006033	172.17.0.5	172.17.0.4	TCP	76	22 → 32790 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2853375963 TSecr=2074493197 WS=128
13	2.006035	172.17.0.5	172.17.0.4	TCP	76	[TCP Out-Of-Order] 22 → 32790 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2853375963 TSecr=2074493
14	2.007857	172.17.0.4	172.17.0.5	TCP	68	32790 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2074493200 TSecr=2853375963
15	2.007863	172.17.0.4	172.17.0.5	TCP	68	[TCP Dup ACK 14#1] 32790 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2074493200 TSecr=2853375963
16	2.010399	172.17.0.5	172.17.0.4	SSHv2	109	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
17	2.010404	172.17.0.5	172.17.0.4	TCP	109	[TCP Retransmission] 22 → 32790 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=41 TSval=2853375967 TSecr=2074493200
18	2.010426	172.17.0.4	172.17.0.5	TCP	68	32790 → 22 [ACK] Seq=1 Ack=42 Win=64256 Len=0 TSval=2074493202 TSecr=2853375967
19	2.010427	172.17.0.4	172.17.0.5	TCP	68	[TCP Dup ACK 18#1] 32790 → 22 [ACK] Seq=1 Ack=42 Win=64256 Len=0 TSval=2074493202 TSecr=2853375967
20	2.018606	172.17.0.4	172.17.0.5	SSHv2	109	Client: Protocol (SSH-2.0-OpenSSH_?), Encrypted packet (len=22)
21	2.018613	172.17.0.4	172.17.0.5	TCP	109	[TCP Retransmission] 32790 → 22 [PSH, ACK] Seq=1 Ack=42 Win=64256 Len=41 TSval=2074493210 TSecr=2853375907
22	2.018738	172.17.0.5	172.17.0.4	TCP	68	22 → 32790 [ACK] Seq=42 Ack=42 Win=65280 Len=0 TSval=2853375975 TSecr=2074493210
23	2.018741	172.17.0.5	172.17.0.4	TCP	68	[TCP Dup ACK 22#1] 22 → 32790 [ACK] Seq=42 Ack=42 Win=65280 Len=0 TSval=2853375975 TSecr=2074493210
24	2.020943	172.17.0.4	172.17.0.5	SSHv2	1580	Client: Key Exchange Init
25	2.020947	172.17.0.4	172.17.0.5	TCP	1580	[TCP Retransmission] 32790 → 22 [PSH, ACK] Seq=42 Ack=42 Win=64256 Len=1512 TSval=2074493213 TSecr=2853375975
26	2.020961	172.17.0.5	172.17.0.4	TCP	68	22 → 32790 [ACK] Seq=42 Ack=1554 Win=64128 Len=0 TSval=2853375978 TSecr=2074493213
27	2.020963	172.17.0.5	172.17.0.4	TCP	68	[TCP Dup ACK 26#1] 22 → 32790 [ACK] Seq=42 Ack=1554 Win=64128 Len=0 TSval=2853375978 TSecr=2074493213
28	2.023328	172.17.0.5	172.17.0.4	SSHv2	1148	Server: Key Exchange Init
29	2.023332	172.17.0.5	172.17.0.4	TCP	1148	[TCP Retransmission] 22 → 32790 [PSH, ACK] Seq=42 Ack=1554 Win=64128 Len=1080 TSval=2853375980 TSecr=2074493213
30	2.027801	172.17.0.4	172.17.0.5	SSHv2	116	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
31	2.027805	172.17.0.4	172.17.0.5	TCP	116	[TCP Retransmission] 32790 → 22 [PSH, ACK] Seq=1554 Ack=1122 Win=64128 Len=48 TSval=2074493220 TSecr=2853375980
32	2.042470	172.17.0.5	172.17.0.4	SSHv2	664	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
33	2.042476	172.17.0.5	172.17.0.4	TCP	664	[TCP Retransmission] 22 → 32790 [PSH, ACK] Seq=1122 Ack=1602 Win=64128 Len=596 TSval=2853375999 TSecr=2074493220
34	2.083906	172.17.0.4	172.17.0.5	TCP	68	32790 → 22 [ACK] Seq=1602 Ack=1718 Win=64128 Len=0 TSval=2074493276 TSecr=2853375999
35	2.083916	172.17.0.4	172.17.0.5	TCP	68	[TCP Dup ACK 34#1] 32790 → 22 [ACK] Seq=1602 Ack=1718 Win=64128 Len=0 TSval=2074493276 TSecr=2853375999
36	5.986879	172.17.0.4	172.17.0.5	SSHv2	84	Client: New Keys
37	5.986887	172.17.0.4	172.17.0.5	TCP	84	[TCP Retransmission] 32790 → 22 [PSH, ACK] Seq=1602 Ack=1718 Win=64128 Len=16 TSval=2074497179 TSecr=2853375999

Figura 28: Trafico modificado.

3. Desarrollo (Parte 3)

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

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH (Open Secure Shell) es un conjunto de herramientas diseñadas para proporcionar comunicaciones seguras en redes no confiables utilizando el protocolo SSH. Este protocolo se centra en garantizar la protección de los datos transmitidos mediante el cifrado, la autenticación mutua y la integridad de las comunicaciones.

SSH establece una conexión segura entre un cliente y un servidor, permitiendo la administración remota de sistemas, la transferencia de archivos y la ejecución de comandos de manera segura. A diferencia de protocolos inseguros como Telnet o FTP, OpenSSH utiliza cifrado robusto para prevenir la intercepción y manipulación de datos por parte de terceros.

Las características principales de OpenSSH incluyen:

- **Cifrado de datos:** Todas las comunicaciones son cifradas para garantizar la confidencialidad y prevenir el acceso no autorizado.
- **Autenticación segura:** Admite múltiples métodos de autenticación, como contraseñas, claves públicas y certificados digitales.
- **Integridad de las comunicaciones:** Utiliza algoritmos de suma de verificación (MACs) para garantizar que los datos no hayan sido alterados durante la transmisión.
- **Flexibilidad y compatibilidad:** OpenSSH soporta túneles seguros y proxies, permitiendo aplicaciones avanzadas como la redirección de puertos.

En resumen, OpenSSH es una herramienta fundamental para establecer comunicaciones seguras, proteger datos sensibles y asegurar la administración remota de sistemas, cumpliendo con los requisitos modernos de seguridad de la información.

4.2. Capas de Seguridad en OpenSSH

El protocolo OpenSSH implementa múltiples capas de seguridad para garantizar los principios fundamentales de la seguridad de la información: integridad, confidencialidad, disponibilidad, autenticidad y no repudio. A continuación, se detallan estas capas y cómo contribuyen a cada principio:

- **Confidencialidad:** OpenSSH utiliza algoritmos de cifrado avanzados como AES (Advanced Encryption Standard) y ChaCha20 para cifrar los datos transmitidos. Esto asegura que la información no pueda ser interceptada ni leída por terceros durante la comunicación.
- **Integridad:** La integridad de los datos es garantizada mediante el uso de códigos de autenticación de mensajes (MAC, Message Authentication Codes) como HMAC-SHA2. Estos algoritmos verifican que los datos no hayan sido alterados durante la transmisión.
- **Autenticidad:** OpenSSH emplea autenticación mutua basada en claves públicas y privadas mediante el uso de algoritmos como RSA, ECDSA y Ed25519. Esto permite verificar la identidad tanto del cliente como del servidor, asegurando que las partes involucradas en la comunicación sean legítimas.
- **Disponibilidad:** Aunque no está diseñada específicamente para garantizar disponibilidad, OpenSSH incluye mecanismos como la resistencia a ataques de fuerza bruta y el rechazo de intentos de conexión excesivos mediante configuraciones como el límite de sesiones por cliente y el tiempo de espera en la autenticación.
- **No repudio:** Si bien OpenSSH no implementa un mecanismo explícito de no repudio, la autenticación mediante claves públicas y privadas genera registros que pueden usarse para vincular las acciones a una entidad específica. Sin embargo, esto depende de configuraciones adicionales en el sistema para el registro y auditoría.

Estas capas de seguridad trabajan conjuntamente para proteger la comunicación en redes no confiables, asegurando que los datos sean transmitidos de forma segura y verificable, y cumpliendo con los principios de la seguridad de la información.

4.3. Identificación de que protocolos no se cumplen

Todos los principios de la seguridad de la información son cumplidos por OpenSSH, dado que implementa medidas robustas para garantizar la confidencialidad, integridad, autenticidad, disponibilidad y no repudio. A continuación, se detallan cómo se cumplen cada uno de estos principios:

- **Confidencialidad:** El uso de cifrado fuerte (como AES y ChaCha20) garantiza que la información no pueda ser interceptada y leída por terceros durante su transmisión.
- **Integridad:** Los códigos de autenticación de mensajes (MAC) aseguran que los datos no sean alterados, garantizando la integridad de las comunicaciones.
- **Autenticidad:** El proceso de autenticación mediante claves públicas y privadas, junto con la verificación mutua entre cliente y servidor, asegura que ambas partes sean legítimas.
- **Disponibilidad:** Aunque no es un protocolo específico para garantizar disponibilidad, OpenSSH implementa medidas de protección contra ataques de denegación de servicio (DoS) y fuerza bruta, lo que ayuda a mantener el servicio accesible.
- **No repudio:** La autenticación mediante claves criptográficas y la capacidad de generar registros detallados proporcionan una forma de asegurar que las acciones realizadas durante la sesión puedan ser rastreadas y vinculadas a una entidad específica.

En conclusión, OpenSSH cumple con los principios fundamentales de seguridad de la información, lo que lo convierte en una herramienta eficaz para garantizar comunicaciones seguras en redes no confiables. Ningún principio queda sin cumplir, dado que todas las medidas implementadas en el protocolo están orientadas a proteger los datos y las interacciones de los usuarios de manera integral.

Conclusiones y comentarios

El protocolo OpenSSH es una herramienta clave para garantizar comunicaciones seguras en redes no confiables, proporcionando un conjunto robusto de mecanismos para proteger la confidencialidad, integridad, autenticidad, disponibilidad y no repudio de los datos transmitidos. Gracias a su cifrado avanzado, autenticación mutua y verificación de integridad, OpenSSH asegura que los datos sean enviados de manera segura, sin riesgo de ser interceptados o manipulados. Además, la autenticación mediante claves criptográficas garantiza la autenticidad de las partes involucradas en la comunicación, mientras que las medidas contra ataques de fuerza bruta contribuyen a la disponibilidad del servicio.

A través del análisis de las capas de seguridad implementadas en OpenSSH, se puede concluir que el protocolo cumple de manera efectiva con los principios fundamentales de la seguridad de la información. Esto lo convierte en una herramienta indispensable para la administración remota de sistemas, la transferencia segura de archivos y la ejecución de comandos en entornos vulnerables. OpenSSH no solo protege los datos en tránsito, sino que también proporciona un medio para garantizar la autenticidad de las entidades involucradas y prevenir cualquier alteración de los datos transmitidos. En resumen, OpenSSH es una solución integral para la seguridad en comunicaciones remotas.