

# Informe Laboratorio 5

## Sección 4

Renato Oscar Benjamin Contreras Carvajal  
e-mail: [renato.contreras@mail.udp.cl](mailto:renato.contreras@mail.udp.cl)

Diciembre de 2024

# Índice

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

## Descripción de actividades

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

Para el presente laboratorio deberá:

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

Pasos:

1. Para cada uno de los 4 escenarios, deberá capturar el tráfico generado por cada conexión con el server. A partir de cada handshake, deberá analizar el patrón de tráfico generado por cada cliente y adicionalmente obtener el HASSH que lo identifique. De esta forma podrá obtener una huella digital para cada cliente a partir de su tráfico.

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.

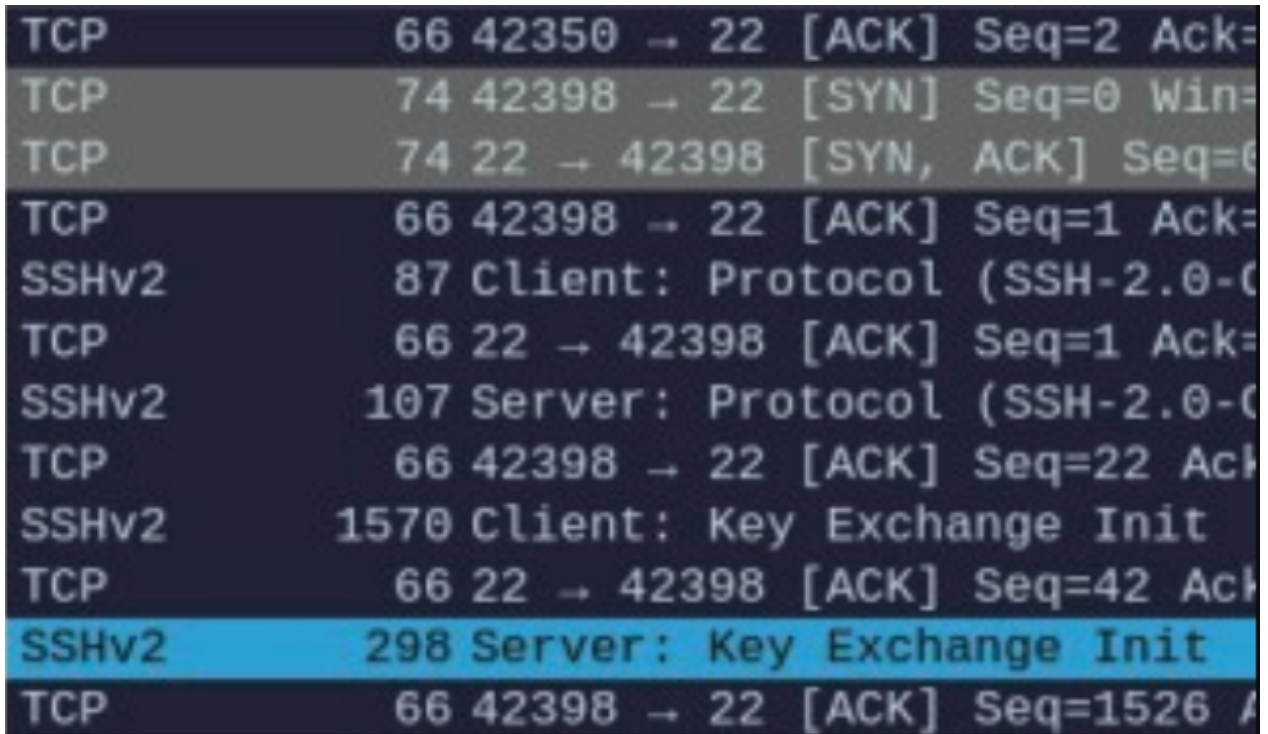
2. Para poder identificar que el usuario efectivamente es el informante, éste utilizará una versión única de cliente. ¿Con qué cliente SSH se habrá generado el siguiente tráfico?

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.



The image shows a network traffic capture with the following entries:

TCP	66	42350 → 22	[ACK]	Seq=2	Ack=
TCP	74	42398 → 22	[SYN]	Seq=0	Win=
TCP	74	22 → 42398	[SYN, ACK]	Seq=0	
TCP	66	42398 → 22	[ACK]	Seq=1	Ack=
SSHv2	87	Client: Protocol (SSH-2.0-C			
TCP	66	22 → 42398	[ACK]	Seq=1	Ack=
SSHv2	107	Server: Protocol (SSH-2.0-C			
TCP	66	42398 → 22	[ACK]	Seq=22	Ack=
SSHv2	1570	Client: Key Exchange Init			
TCP	66	22 → 42398	[ACK]	Seq=42	Ack=
SSHv2	298	Server: Key Exchange Init			
TCP	66	42398 → 22	[ACK]	Seq=1526	Ack=

Figura 2: Captura del Key Exchange

- 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 utilizados para configurar cada uno de los contenedores (C1, C2, C3 y C4/S1) en el laboratorio. Cada archivo ‘Dockerfile’ define la configuración del sistema operativo y las herramientas necesarias para cumplir con los requisitos del laboratorio.

#### 1.1.1. C1

El contenedor C1 utiliza Ubuntu 16.10 y configura el cliente OpenSSH para conexiones SSH.

Listing 1: Dockerfile de C1

```
FROM ubuntu:16.10

ENV DEBIAN_FRONTEND=noninteractive

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|g' /etc/apt/sources.list && \
    sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y openssh-client && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /root
CMD ["bash"]
```

### 1.1.2. C2

El contenedor C2 utiliza Ubuntu 18.10 con una configuración similar para habilitar el cliente OpenSSH.

Listing 2: Dockerfile de C2

```
FROM ubuntu:18.10

ENV DEBIAN_FRONTEND=noninteractive

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|g' /etc/apt/sources.list && \
    sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y openssh-client && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /root
CMD ["bash"]
```

### 1.1.3. C3

El contenedor C3 utiliza Ubuntu 20.10 y también configura el cliente OpenSSH.

Listing 3: Dockerfile de C3

```
FROM ubuntu:20.10

ENV DEBIAN_FRONTEND=noninteractive

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|g' /etc/apt/sources.list && \
    sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y openssh-client && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /root
CMD ["bash"]
```

#### 1.1.4. C4/S1

El contenedor C4/S1 utiliza Ubuntu 22.10 y configura tanto el cliente como el servidor OpenSSH. Además, se crea un usuario llamado **prueba** con contraseña **prueba** para conexiones SSH.

Listing 4: Dockerfile de C4/S1

```
FROM ubuntu:22.10

ENV DEBIAN_FRONTEND=noninteractive

RUN sed -i 's|http://archive.ubuntu.com/ubuntu/|http://old-releases.ubuntu.com/ubuntu/|g' /etc/apt/sources.list && \
    sed -i 's|http://security.ubuntu.com/ubuntu|http://old-releases.ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y openssh-client openssh-server && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

RUN useradd -ms /bin/bash prueba && \
    echo "prueba:prueba" | chpasswd && \
    mkdir -p /var/run/sshd

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

## 1.2. Creación de las credenciales para S1

En el contenedor C4/S1, que actúa como servidor SSH, se configuraron las credenciales necesarias para permitir la conexión remota mediante el protocolo SSH. Esto incluye la creación del usuario `prueba` con la contraseña `prueba`. A continuación, se muestra el fragmento del código utilizado en el `Dockerfile` de C4/S1 para esta configuración:

Listing 5: Creación de las credenciales en C4/S1

```
RUN useradd -ms /bin/bash prueba && \
    echo "prueba:prueba" | chpasswd
```

La descripción del comando es:

- `useradd -ms /bin/bash prueba`: Este comando crea un nuevo usuario llamado `prueba`, asignándole el intérprete de comandos `/bin/bash` como shell predeterminado y un directorio de inicio en `/home/prueba`.
- `echo "prueba:prueba" | chpasswd`: Este comando asigna la contraseña `prueba` al usuario `prueba`.

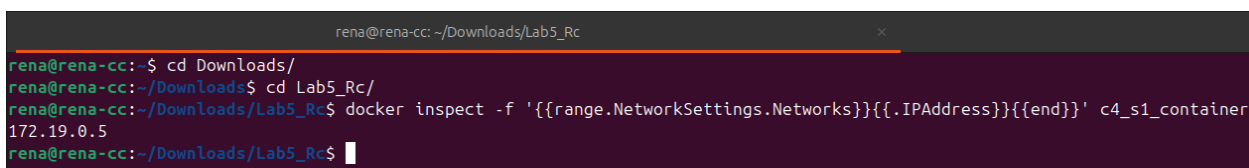
Estas credenciales permiten que los clientes C1, C2 y C3 se conecten al servidor SSH en C4/S1 para las pruebas de conexión y captura de tráfico.

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

Para conectarse a C4.S1, es necesario conocer su IP (Esta fue variando en el ultimo dígito durante los días que se realizó el laboratorio). Para esto, se utiliza el siguiente comando:

Listing 6: Creación de las credenciales en C4/S1

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' c4\_s1\_container
```



The screenshot shows a terminal window with the following commands and output:

```
rena@rena-cc: ~/Downloads/Lab5_Rc
rena@rena-cc:~$ cd Downloads/
rena@rena-cc:~/Downloads$ cd Lab5_Rc/
rena@rena-cc:~/Downloads/Lab5_Rc$ docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' c4_s1_container
172.19.0.5
rena@rena-cc:~/Downloads/Lab5_Rc$
```

Figura 3: Obtención de la IP del contenedor C4.S1.

Luego, se utiliza el siguiente comando:

Listing 7: Conexión SSH desde C1 a C4/S1

```
ssh prueba@172.19.0.3
```



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

#### 1 DESARROLLO (PARTE 1)

Antes de conectarse, se abre Wireshark para observar el tráfico generado:

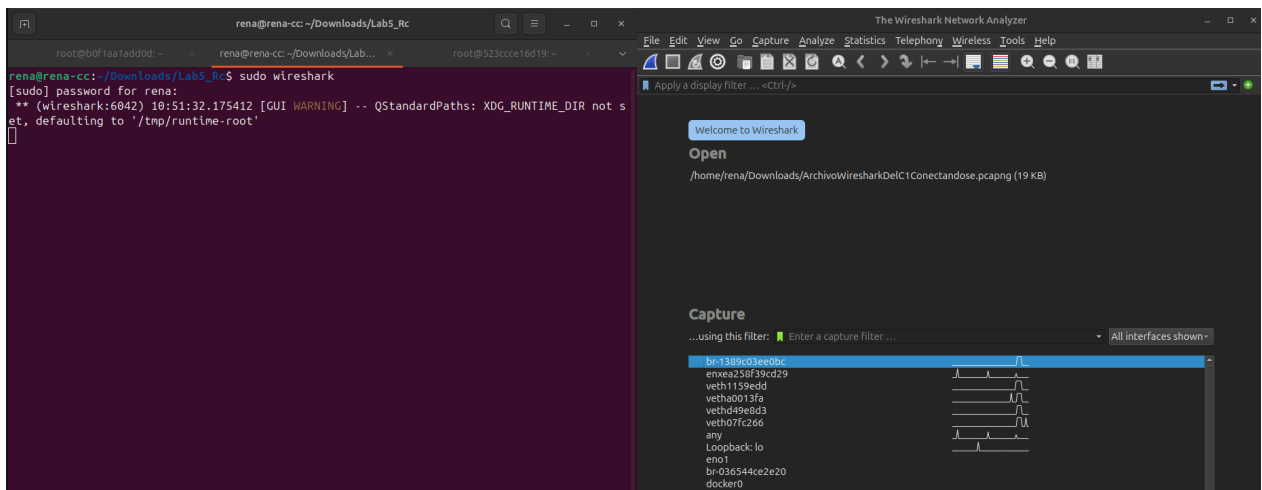


Figura 4: Abriendo Wireshark.

Con esto listo, se realiza la conexión y se guarda la captura del tráfico en 'Tráfico-C1-to-C4', obteniendo lo siguiente:

Source	Destination	Protocol	Length	Info
172.19.0.2	172.19.0.3	TCP	74	33430 → 22 [SYN] Seq=0 Win=
172.19.0.3	172.19.0.2	TCP	74	22 → 33430 [SYN, ACK] Seq=0
172.19.0.2	172.19.0.3	TCP	66	33430 → 22 [ACK] Seq=1 Ack=
172.19.0.2	172.19.0.3	SSHv2	107	Client: Protocol (SSH-2.0-0
172.19.0.3	172.19.0.2	TCP	66	22 → 33430 [ACK] Seq=1 Ack=
172.19.0.3	172.19.0.2	SSHv2	107	Server: Protocol (SSH-2.0-0
172.19.0.2	172.19.0.3	TCP	66	33430 → 22 [ACK] Seq=42 Ack=
172.19.0.2	172.19.0.3	SSHv2	1498	Client: Key Exchange Init
172.19.0.3	172.19.0.2	SSHv2	1146	Server: Key Exchange Init
172.19.0.2	172.19.0.3	SSHv2	114	Client: Elliptic Curve Diff
172.19.0.3	172.19.0.2	SSHv2	662	Server: Elliptic Curve Diff
172.19.0.2	172.19.0.3	SSHv2	82	Client: New Keys
172.19.0.3	172.19.0.2	TCP	66	22 → 33430 [ACK] Seq=1718 A
172.19.0.2	172.19.0.3	SSHv2	110	Client:

Figura 5: Tráfico capturado entre C1 y C4/S1.

A continuación, se inspeccionan los paquetes del cliente y del servidor, específicamente el *Key Exchange Init*, para obtener el *HASSH* y la huella digital del cliente, como se observa en las siguientes imágenes:

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

#### 1 DESARROLLO (PARTE 1)

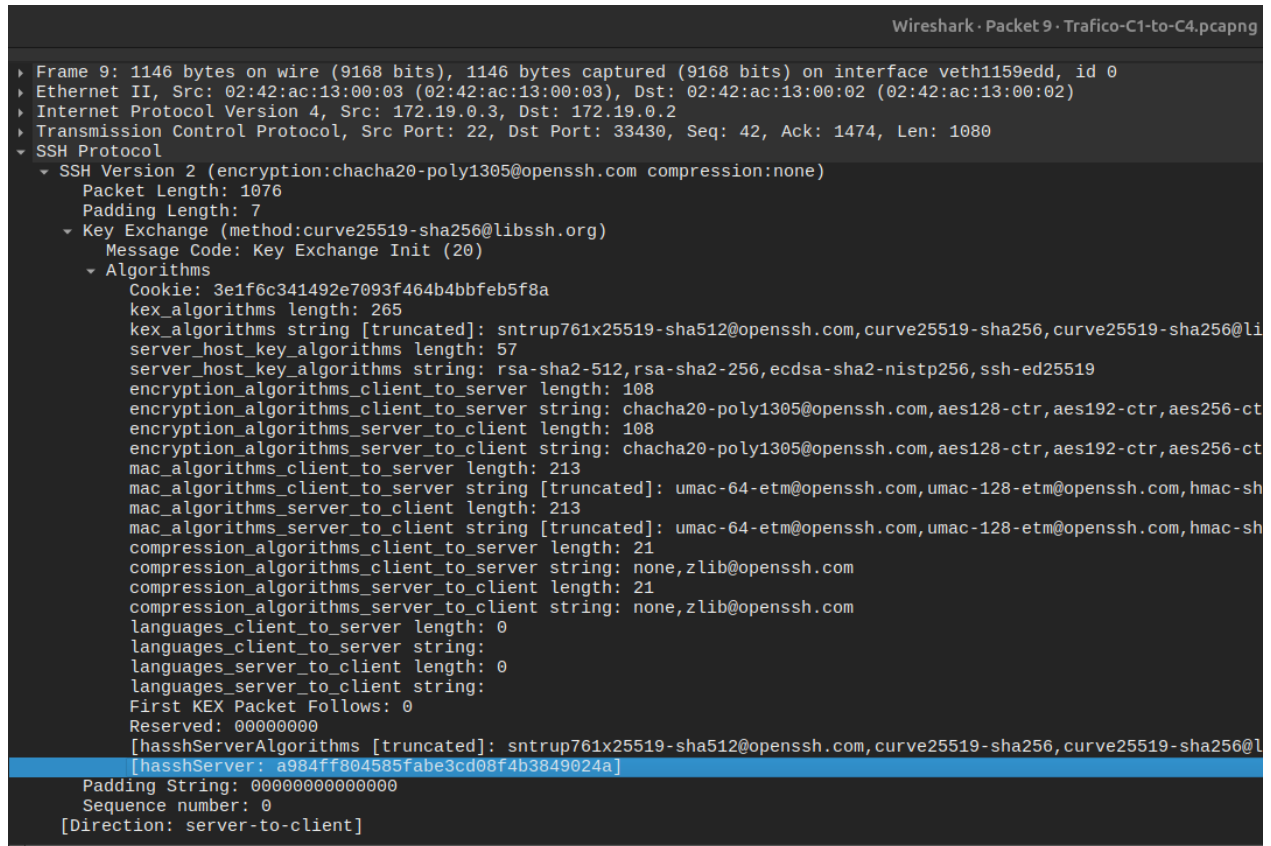
```
Wireshark · Packet 8 · Trafico-C1-to-C4.pcapng
> Frame 8: 1498 bytes on wire (11984 bits), 1498 bytes captured (11984 bits) on interface veth1159edd, id 0
> Ethernet II, Src: 02:42:ac:13:00:02 (02:42:ac:13:00:02), Dst: 02:42:ac:13:00:03 (02:42:ac:13:00:03)
> Internet Protocol Version 4, Src: 172.19.0.2, Dst: 172.19.0.3
> Transmission Control Protocol, Src Port: 33430, Dst Port: 22, Seq: 42, Ack: 42, Len: 1432
> SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1428
    Padding Length: 11
    Key Exchange (method:curve25519-sha256@libssh.org)
      Message Code: Key Exchange Init (20)
      Algorithms
        Cookie: 239219a8f6a898b8dd2fdf57b9000029d
        kex_algorithms length: 286
        kex_algorithms string [truncated]: curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-s
        server_host_key_algorithms length: 290
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-
        encryption_algorithms_client_to_server length: 150
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ct
        encryption_algorithms_server_to_client length: 150
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ct
        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-sh
        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-sh
        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@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-n
        [hassh: 0e4584cb9f2dd077dbf8ba0df8112d8e]
      Padding String: 00000000000000000000000000000000
      Sequence number: 0
      [Direction: client-to-server]
```

Figura 6: En celeste se observa el HASSH obtenido.

El HASSH obtenido tiene un valor de: 0e4584cb9f2dd077dbf8ba0df8112d8e y como se en el octavo paquete de la Figura 5, el largo del mensaje tiene un valor de 1498.

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

### 1 DESARROLLO (PARTE 1)



```
Wireshark · Packet 9 · Trafico-C1-to-C4.pcapng
> Frame 9: 1146 bytes on wire (9168 bits), 1146 bytes captured (9168 bits) on interface veth1159edd, id 0
> Ethernet II, Src: 02:42:ac:13:00:03 (02:42:ac:13:00:03), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
> Internet Protocol Version 4, Src: 172.19.0.3, Dst: 172.19.0.2
> Transmission Control Protocol, Src Port: 22, Dst Port: 33430, Seq: 42, Ack: 1474, Len: 1080
> SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
    Key Exchange (method:curve25519-sha256@libssh.org)
      Message Code: Key Exchange Init (20)
      Algorithms
        Cookie: 3e1f6c341492e7093f464b4bbfeb5f8a
        kex_algorithms length: 265
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@li
        server_host_key_algorithms length: 57
        server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed25519
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ct
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ct
        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-sh
        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-sh
        compression_algorithms_client_to_server length: 21
        compression_algorithms_client_to_server string: none,zlib@openssh.com
        compression_algorithms_server_to_client length: 21
        compression_algorithms_server_to_client string: none,zlib@openssh.com
        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
        [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@l
        [hasshServer: a984ff804585fabe3cd08f4b3849024a]
      Padding String: 0000000000000000
      Sequence number: 0
      [Direction: server-to-client]
```

Figura 7: En celeste se observa el HASSHServer obtenido.

En consecuencia, el HasshServer obtenido es: a984ff804585fabe3cd08f4b3849024a y como se en el noveno paquete de la Figura 5, el largo del mensaje tiene un valor de 1146.

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

Repitiendo lo anterior, se realiza la conexión, ahora con C2, y se guarda la captura del tráfico en 'Trafico-C2-to-C4', obteniendo lo siguiente:

Source	Destination	Protocol	Length	Info
172.19.0.4	172.19.0.3	TCP	74	54066 → 22 [SYN] Seq=0 Win=64240
172.19.0.3	172.19.0.4	TCP	74	22 → 54066 [SYN, ACK] Seq=0 Ack=
172.19.0.4	172.19.0.3	TCP	66	54066 → 22 [ACK] Seq=1 Ack=1 Win
172.19.0.4	172.19.0.3	SSHv2	107	Client: Protocol (SSH-2.0-OpenSS
172.19.0.3	172.19.0.4	TCP	66	22 → 54066 [ACK] Seq=1 Ack=42 Wi
172.19.0.3	172.19.0.4	SSHv2	107	Server: Protocol (SSH-2.0-OpenSS
172.19.0.4	172.19.0.3	TCP	66	54066 → 22 [ACK] Seq=42 Ack=42 W
172.19.0.4	172.19.0.3	SSHv2	1426	Client: Key Exchange Init
172.19.0.3	172.19.0.4	SSHv2	1146	Server: Key Exchange Init
172.19.0.4	172.19.0.3	SSHv2	114	Client: Elliptic Curve Diffie-He
172.19.0.3	172.19.0.4	SSHv2	662	Server: Elliptic Curve Diffie-He

Figura 8: Tráfico capturado entre C2 y C4/S1.

De nuevo, se inspeccionan los paquetes del cliente y del servidor, específicamente el *Key Exchange Init*, para obtener el *HASSH* y la huella digital del cliente, como se observa en las siguientes imágenes:

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

### 1 DESARROLLO (PARTE 1)

```

> Frame 8: 1426 bytes on wire (11408 bits), 1426 bytes captured (11408 bits) on interface vethd49e8d3, id 0
> Ethernet II, Src: 02:42:ac:13:00:04 (02:42:ac:13:00:04), Dst: 02:42:ac:13:00:03 (02:42:ac:13:00:03)
> Internet Protocol Version 4, Src: 172.19.0.4, Dst: 172.19.0.3
> Transmission Control Protocol, Src Port: 54066, Dst Port: 22, Seq: 42, Ack: 42, Len: 1360
> SSH Protocol
  > SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1356
    Padding Length: 5
    > Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      > Algorithms
        Cookie: fad3fa1577a52e0dfaaca9c89cec32e
        kex_algorithms length: 304
        kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp2
        server_host_key_algorithms length: 290
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.c
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.c
        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,ec
        [hassh: 06046964c022c6407d15a27b12a6a4fb]
      Padding String: 0000000000
      Sequence number: 0
      [Direction: client-to-server]
04d0 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61 63 openssh. com,hmac
04e0 2d 73 68 61 31 2d 65 74 6d 40 6f 70 65 6e 73 73 -sha1-et m@openss

```

Figura 9: En celeste se observa el HASSH obtenido.

El *HASSH* obtenido tiene un valor de: 06046964c022c6407d15a27b12a6a4fb y, como se muestra en el octavo paquete de la Figura 8, el largo del mensaje tiene un valor de 1426.

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

### 1 DESARROLLO (PARTE 1)

```
> Frame 9: 1146 bytes on wire (9168 bits), 1146 bytes captured (9168 bits) on interface vethd49e8d3, id 0
> Ethernet II, Src: 02:42:ac:13:00:03 (02:42:ac:13:00:03), Dst: 02:42:ac:13:00:04 (02:42:ac:13:00:04)
> Internet Protocol Version 4, Src: 172.19.0.3, Dst: 172.19.0.4
> Transmission Control Protocol, Src Port: 22, Dst Port: 54066, Seq: 42, Ack: 1402, Len: 1080
> SSH Protocol
  > SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
    > Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      > Algorithms
        Cookie: d9b48f8fd85ca3387a2f866f27d06eaf
        kex_algorithms length: 265
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519
        server_host_key_algorithms length: 57
        server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed25519
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.com
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@openssh.com
        compression_algorithms_client_to_server length: 21
        compression_algorithms_client_to_server string: none,zlib@openssh.com
        compression_algorithms_server_to_client length: 21
        compression_algorithms_server_to_client string: none,zlib@openssh.com
        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
        [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519
        [hasshServer: a984ff804585fabe3cd08f4b3849024a]
        Padding String: 00000000000000
        Sequence number: 0
        [Direction: server-to-client]
```

Figura 10: En celeste se observa el HASSHServer obtenido.

En consecuencia, el HasshServer obtenido es: a984ff804585fabe3cd08f4b3849024a y, como se muestra en el noveno paquete de la Figura 8, el largo del mensaje tiene un valor de 1146.

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

Nuevamente, se realiza la conexión, ahora con C3, y se guarda la captura del tráfico en 'Trafico-C3-to-C4', obteniendo lo siguiente:

# 1.5 Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo 1 DESARROLLO (PARTE 1)

Source	Destination	Protocol	Length	Info
172.19.0.5	172.19.0.3	TCP	74	48806 → 22 [SYN] Seq=0 Win=642
172.19.0.3	172.19.0.5	TCP	74	22 → 48806 [SYN, ACK] Seq=0 Ac
172.19.0.5	172.19.0.3	TCP	66	48806 → 22 [ACK] Seq=1 Ack=1 W
172.19.0.5	172.19.0.3	SSHv2	107	Client: Protocol (SSH-2.0-Open
172.19.0.3	172.19.0.5	TCP	66	22 → 48806 [ACK] Seq=1 Ack=42
172.19.0.3	172.19.0.5	SSHv2	107	Server: Protocol (SSH-2.0-Open
172.19.0.5	172.19.0.3	TCP	66	48806 → 22 [ACK] Seq=42 Ack=42
172.19.0.5	172.19.0.3	SSHv2	1578	Client: Key Exchange Init
172.19.0.3	172.19.0.5	SSHv2	1146	Server: Key Exchange Init
172.19.0.5	172.19.0.3	SSHv2	114	Client: Elliptic Curve Diffie-

Figura 11: Tráfico capturado entre C3 y C4/S1.

Otra vez, se inspeccionan los paquetes del cliente y del servidor, específicamente el *Key Exchange Init*, para obtener el *HASSH* y la huella digital del cliente, como se observa en las siguientes imágenes:



1.5 Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo  
(detallado) 1 DESARROLLO (PARTE 1)

```

▶ Frame 8: 1578 bytes on wire (12624 bits), 1578 bytes captured (12624 bits) on interface
▶ Ethernet II, Src: 02:42:ac:13:00:05 (02:42:ac:13:00:05), Dst: 02:42:ac:13:00:03 (02:42:ac:13:00:03)
▶ Internet Protocol Version 4, Src: 172.19.0.5, Dst: 172.19.0.3
▶ Transmission Control Protocol, Src Port: 48806, Dst Port: 22, Seq: 42, Ack: 42, Len: 1578
▼ SSH Protocol
  ▼ SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1508
    Padding Length: 10
    ▼ Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      ▼ Algorithms
        Cookie: 0f2173345ff48406de2b346f841137c5
        kex_algorithms length: 241
        kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha256@libssh.org
        server_host_key_algorithms length: 500
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp256@libssh.org
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,chacha20-poly1305@openssh.com
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,chacha20-poly1305@openssh.com
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-64-etm@openssh.com
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-64-etm@openssh.com
        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]
        [hassh: ae8bd7dd09970555aa4c6ed22adbbf56]
        Padding String: 000000000000000000000000
        Sequence number: 0
        [Direction: client-to-server]
0560 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 tm@opensh.com,h
0570 6d 61 63 2d 73 68 61 31 2d 65 74 6d 40 6f 70 65 mac-sha1-etm@one

```

Figura 12: En celeste se observa el HASSH obtenido.

El *HASSH* obtenido tiene un valor de: ae8bd7dd09970555aa4c6ed22adbbf56 y, como se muestra en el octavo paquete de la Figura 11, el largo del mensaje tiene un valor de 1578.



```

▶ Frame 9: 1146 bytes on wire (9168 bits), 1146 bytes captured (9168 bits) on interface
▶ Ethernet II, Src: 02:42:ac:13:00:03 (02:42:ac:13:00:03), Dst: 02:42:ac:13:00:05 (02:4
▶ Internet Protocol Version 4, Src: 172.19.0.3, Dst: 172.19.0.5
▶ Transmission Control Protocol, Src Port: 22, Dst Port: 48806, Seq: 42, Ack: 1554, Len
▼ SSH Protocol
  ▼ SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
    ▼ Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      ▼ Algorithms
        Cookie: 7f73a003b508dc40488cd36a462bc463
        kex_algorithms length: 265
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.com,curve2
        server_host_key_algorithms length: 57
        server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nist
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,
        compression_algorithms_client_to_server length: 21
        compression_algorithms_client_to_server string: none,zlib@openssh.com
        compression_algorithms_server_to_client length: 21
        compression_algorithms_server_to_client string: none,zlib@openssh.com
        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
        [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.com,curve
        [hasshServer: a984ff804585fabe3cd08f4b3849024a]
        Padding String: 0000000000000000
        Sequence number: 0
        [Direction: server-to-client]
03c0 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61 63 openssh. com,hmac

```

Figura 13: En celeste se observa el HASSHServer obtenido.

En consecuencia, el HasshServer obtenido es: a984ff804585fabe3cd08f4b3849024a y, como se muestra en el noveno paquete de la Figura 11, el largo del mensaje tiene un valor de 1146.

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

En este caso, se analizó la conexión entre C4 y S1, ambos ejecutándose dentro del mismo contenedor. Esto presentó un desafío adicional, ya que Wireshark no lograba capturar el tráfico generado en la interfaz loopback (lo). Para solucionar este inconveniente, se utilizó la herramienta `tcpdump`, configurada para escuchar específicamente en la interfaz lo:

```
root@eb6cd9b86cb1:/# tcpdump -i lo -w /tmp/traffic.pcap
tcpdump: listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C37 packets captured
74 packets received by filter
0 packets dropped by kernel
root@eb6cd9b86cb1:/#
```

Figura 14: Captura del tráfico utilizando tcpdump en la interfaz loopback.

El tráfico capturado se guardó en un archivo `Trafico-C4-to-S1`. Posteriormente, este archivo fue exportado al equipo anfitrión y analizado con Wireshark, obteniendo la siguiente visualización:

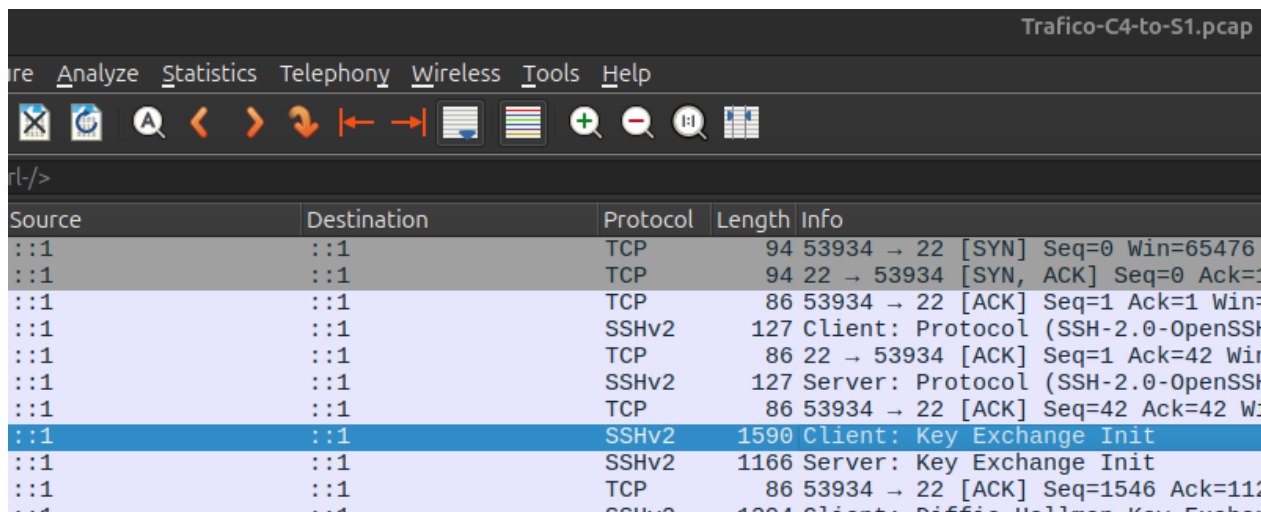


Figura 15: Tráfico capturado entre C4 y S1 visualizado en Wireshark.

A continuación, se inspeccionaron los paquetes clave del cliente y del servidor, con el enfoque en el proceso de *Key Exchange Init*. Esto permitió identificar tanto el *HASSH* del cliente como la huella digital del servidor, como se observa en las siguientes imágenes:

```

> Frame 8: 1590 bytes on wire (12720 bits), 1590 bytes captured (12720 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 53934, Dst Port: 22, Seq: 42, Ack: 42, Len: 1590
> SSH Protocol
  > SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1500
    Padding Length: 4
    > Key Exchange (method:sntrup761x25519-sha512@openssh.com)
      Message Code: Key Exchange Init (20)
      > Algorithms
        Cookie: 20140c9a3cd4aa64d18cdf4f229fc332
        kex_algorithms length: 276
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.com,curve25519-sha512@openssh.com,curve25519-sha256@openssh.com,curve25519-sha256@openssh.com
        server_host_key_algorithms length: 463
        server_host_key_algorithms string [truncated]: ssh-ed25519-cert-v01@openssh.com,ssh-ed25519,ssh-rsa,ssh-dss,ssh-key-exchange
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,curve25519-sha256@openssh.com,curve25519-sha256@openssh.com,curve25519-sha256@openssh.com
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,curve25519-sha256@openssh.com,curve25519-sha256@openssh.com,curve25519-sha256@openssh.com
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-64-etm@openssh.com,umac-64-etm@openssh.com,umac-64-etm@openssh.com
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-64-etm@openssh.com,umac-64-etm@openssh.com,umac-64-etm@openssh.com
        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]: sntrup761x25519-sha512@openssh.com,curve25519-sha512@openssh.com,curve25519-sha256@openssh.com,curve25519-sha256@openssh.com]
        [hassh: 78c05d999799066a2b4554ce7b1585a6]
      Padding String: 00000000
      Sequence number: 0
      [Direction: client-to-server]
    0480 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61 63 2d 73 enssh.co m,hmac-s
    0490 68 61 32 2d 35 31 32 2d 65 74 6d 40 6f 70 65 6e ba2-512-etm@open
    
```

Figura 16: *HASSH* del cliente destacado en celeste.

El valor del *HASSH* obtenido es 78c05d999799066a2b4554ce7b1585a6. Según se muestra en el octavo paquete de la Figura 15, el tamaño del mensaje correspondiente es de 1590 bytes.

```

> Frame 9: 1166 bytes on wire (9328 bits), 1166 bytes captured (9328 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 22, Dst Port: 53934, Seq: 42, Ack: 1546, Len:
> SSH Protocol
  > SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
    > Key Exchange (method:sntrup761x25519-sha512@openssh.com)
      Message Code: Key Exchange Init (20)
      > Algorithms
        Cookie: 80a8e92801e9ee6c9b783e2633a32dc9
        kex_algorithms length: 265
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.com,curve25519
        server_host_key_algorithms length: 57
        server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes128-cbc@openssh.com
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes128-cbc@openssh.com
        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,hmac-sha2-512,hmac-sha1
        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,hmac-sha2-512,hmac-sha1
        compression_algorithms_client_to_server length: 21
        compression_algorithms_client_to_server string: none,zlib@openssh.com
        compression_algorithms_server_to_client length: 21
        compression_algorithms_server_to_client string: none,zlib@openssh.com
        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
        [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.com,curve25519
        [hasshServer: a984ff804585fabe3cd08f4b3849024a]
      Padding String: 0000000000000000
      Sequence number: 0
      [Direction: server-to-client]
    03d0 65 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c etm@open ssh.com.
    
```

Figura 17: HASSH del servidor destacado en celeste.

Por otro lado, la huella digital del servidor es a984ff804585fabe3cd08f4b3849024a, con un tamaño de mensaje de 1166 bytes, como se detalla en el noveno paquete de la Figura 15.

Se puede observar que en todas las conexiones el patrón que sigue el protocolo es el siguiente:

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

Protocol	Length	Info
SSHv2	127	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
SSHv2	127	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
SSHv2	1590	Client: Key Exchange Init
SSHv2	1166	Server: Key Exchange Init
SSHv2	1294	Client: Diffie-Hellman Key Exchange Init
SSHv2	1650	Server: Diffie-Hellman Key Exchange Reply, New Keys
SSHv2	102	Client: New Keys
SSHv2	130	Client:
SSHv2	130	Server:
SSHv2	154	Client:

Figura 18: Patrón observado en las conexiones.

Por lo que el proceso de handshake, en el protocolo SSH, sigue un patrón estructurado que se repite independientemente de las versiones o configuraciones específicas utilizadas. Inicialmente, el cliente envía su versión de protocolo, un mensaje que incluye detalles del software que utiliza (como OpenSSH) y su versión específica. Este paso inicial establece un marco común para la negociación, a lo que el servidor responde enviando su propia versión del protocolo. Este sirve para garantizar que ambas partes estén alineadas en términos de compatibilidad.

A continuación, se inicia la negociación de algoritmos, donde cliente y servidor intercambian listas de algoritmos compatibles para las distintas fases de la conexión (se explica un poco mas en '1.7. Tipo de información contenida en cada uno de los paquetes generados en texto plano'). Estas listas suelen incluyen opciones de cifrado, autenticación, compresión e intercambio de claves. Este paso garantiza que el protocolo seleccione los algoritmos más robustos y compatibles según las configuraciones específicas de cliente y servidor.

Una vez definidos los algoritmos a utilizar, se procede al intercambio de claves. Durante esta etapa, se implementa un esquema como Diffie-Hellman, que permite a ambas partes establecer una clave compartida de manera segura sin exponerla directamente en la comunicación. Los paquetes en este punto contienen parámetros públicos y las claves parciales generadas, lo que asegura que la conexión resultante sea confidencial.

Finalmente, tanto el cliente como el servidor confirman la activación de las nuevas claves mediante mensajes pequeños que formalizan el inicio de la comunicación segura.

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

El contenido de los paquetes durante cada una de las etapas mencionadas anteriormente varían. Los primeros mensajes contienen versiones de protocolo y metadatos. En la negociación de algoritmos, se incluyen listas de opciones compatibles, mientras que en el intercambio de claves se transmiten los parámetros específicos de ese esquema. Los mensajes finales son más compactos, ya que su propósito es puramente de control y confirmación.

Aunque este patrón es consistente, los escenarios pueden diferir en pequeños detalles,

como la versión de SSH utilizada, los algoritmos negociados o las configuraciones específicas del cliente y el servidor. Por ejemplo, diferentes versiones del protocolo pueden activar o desactivar ciertos algoritmos, y configuraciones personalizadas pueden modificar las etapas del proceso, como priorizar ciertos métodos de autenticación. Estas diferencias incrementales afectan los tamaños y el contenido de los paquetes, proporcionando pistas sobre las características específicas de cada conexión.

### 1.7.1. C1

El paquete incluye información de inicio de sesión mediante el protocolo SSH versión 2, con el cifrado `chacha20-poly1305@openssh.com`. El mensaje contiene los algoritmos de intercambio de claves (`curve25519-sha256@libssh.org`), autenticación y cifrado, además de la longitud del paquete (1428 bytes) y padding (11 bytes).

### 1.7.2. C2

Incluye información similar a C1, pero con algunas actualizaciones. Utiliza la misma versión de SSH y cifrado, pero el método de intercambio de claves cambia a `curve25519-sha256`. La longitud del paquete es menor (1356 bytes) con un padding de 5 bytes.

### 1.7.3. C3

Se observa una estructura similar, con un intercambio de claves basado en `curve25519-sha256`. El paquete aumenta de tamaño (1508 bytes) con un padding de 10 bytes. Los algoritmos de cifrado, autenticación y compresión permanecen consistentes.

### 1.7.4. C4/S1

Este paquete introduce una mejora significativa con el método de intercambio de claves `sntrup761x25519-sha512@openssh.com`, lo que refleja un mayor nivel de seguridad. La longitud del paquete es de 1500 bytes y el padding de 4 bytes.

## 1.8. Diferencia entre C1 y C2

C2 utiliza el algoritmo `curve25519-sha256` para el intercambio de claves, mientras que C1 emplea `curve25519-sha256@libssh.org`. Además, la longitud del paquete disminuye de 1428 bytes (C1) a 1356 bytes (C2), lo que indica una optimización en la transmisión de datos.

## 1.9. Diferencia entre C2 y C3

En C3, el tamaño del paquete aumenta a 1508 bytes, lo que puede deberse a un aumento en la cantidad de información de los algoritmos admitidos. Sin embargo, el método de intercambio de claves permanece igual (`curve25519-sha256`), con un cambio notable en el padding, de 5 bytes (C2) a 10 bytes (C3).



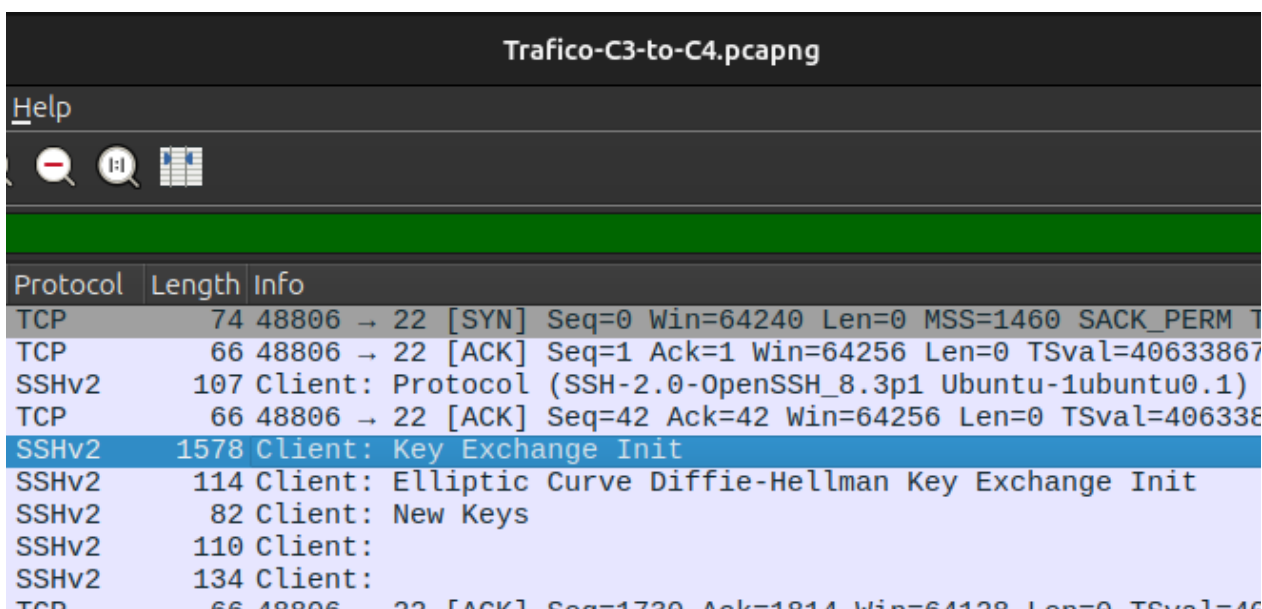
## 1.10. Diferencia entre C3 y C4

C4 introduce un nuevo método de intercambio de claves, `sntrup761x25519-sha512@openssh.com`, que proporciona mayor seguridad al combinar criptografía de clave pública clásica con post-cuántica. A pesar de esto, el tamaño del paquete disminuye ligeramente a 1500 bytes, probablemente debido a una mejor optimización del mensaje.

## 2. Desarrollo (Parte 2)

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

Para identificar la versión del cliente SSH, se analizó el tráfico asociado al protocolo SSHv2 generado por este. Particularmente, se tomó como referencia el tamaño del paquete correspondiente al mensaje Key Exchange Init, el cual tiene un valor de 1578 bytes. Comparando este tamaño con los resultados obtenidos, se determinó que coincide con el cliente C3, cuya versión de sistema operativo es Ubuntu 20.10 y utiliza OpenSSH 8.3p1. Este análisis se observa en la figura siguiente:



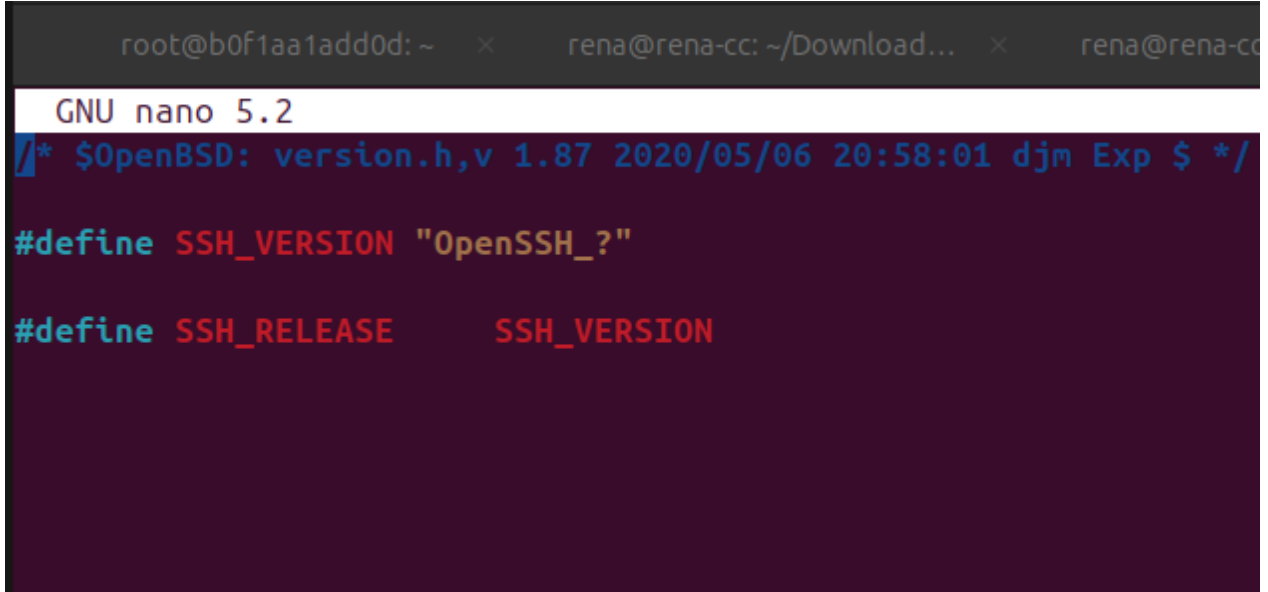
Protocol	Length	Info
TCP	74	48806 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM T
TCP	66	48806 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=40633867
SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
TCP	66	48806 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=406338
SSHv2	1578	Client: Key Exchange Init
SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
SSHv2	82	Client: New Keys
SSHv2	110	Client:
SSHv2	134	Client:
TCP	66	48806 → 22 [ACK] Seq=1730 Ack=1814 Win=64128 Len=0 TSval=40

Figura 19: Tráfico obtenido que es similar al del informante.

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

Para replicar el tráfico generado hacia el servidor utilizando una versión personalizada del cliente SSH (`OpenSSH_?`), se realizaron los siguientes pasos:

1. Se ingresó al contenedor correspondiente al cliente SSH (C3) y se instalaron las herramientas necesarias para compilar el código fuente de OpenSSH. Esto incluyó las dependencias principales como `wget`, `build-essential`, `zlib1g-dev`, `libssl-dev`, y otras:



```
root@b0f1aa1add0d: ~ x rena@rena-cc: ~/Download... x rena@rena-cc: ~
GNU nano 5.2
/* $OpenBSD: version.h,v 1.87 2020/05/06 20:58:01 djm Exp $ */
#define SSH_VERSION "OpenSSH_?"
#define SSH_RELEASE SSH_VERSION
```

Figura 20: Configuración realizada.

```
apt-get update
apt-get install -y wget build-essential zlib1g-dev libssl-dev
libpam0g-dev libselinux1-dev libwrap0-dev ca-certificates
```

2. Se descargó el código fuente de OpenSSH 8.3p1 desde el repositorio oficial y se extrajo su contenido:

```
wget https://cdn.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh
-8.3p1.tar.gz
tar xvfz openssh-8.3p1.tar.gz
cd openssh-8.3p1
```

3. Se modificó el archivo `version.h` dentro del código fuente para cambiar la versión pre-determinada de `OpenSSH-8.3p1` a `OpenSSH-?`. Esto se logró editando la línea correspondiente y eliminando el sufijo `p1`:

```
#define SSH_VERSION "OpenSSH_?"
#define SSH_RELEASE SSH_VERSION
```

4. Una vez realizados los cambios, se configuró, compiló e instaló el cliente SSH modificado:

```
./configure --prefix=/usr --sysconfdir=/etc/ssh
make
make install
```

5. Se verificó que el cliente SSH mostrara la versión personalizada mediante el comando:

```
ssh -V
```



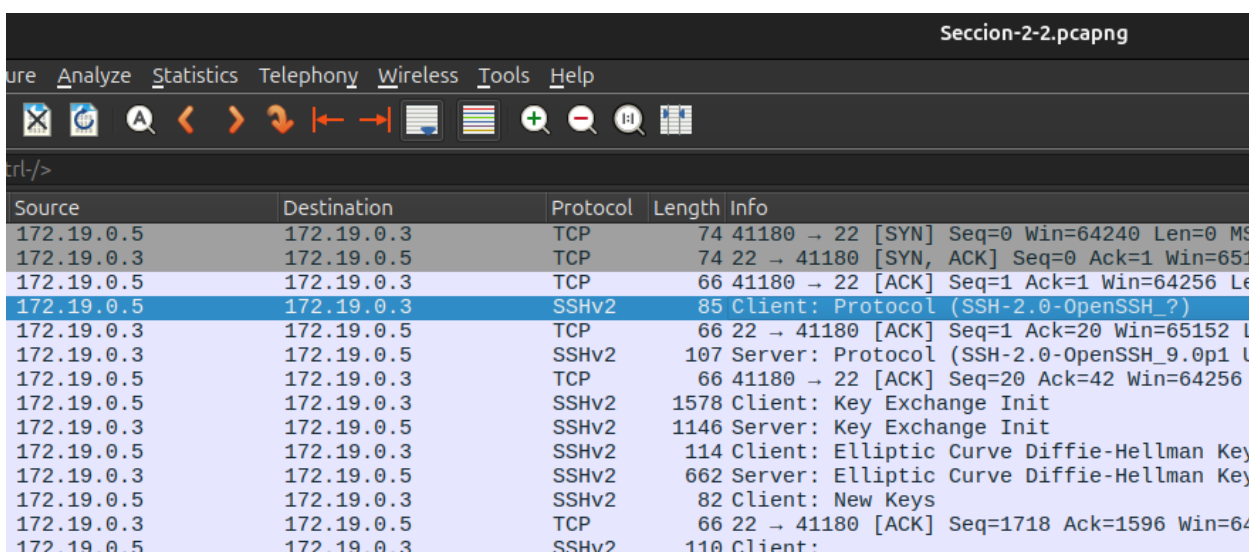
```

root@a61c1a872c3b:~/openssh-8.3p1# nano version.h
root@a61c1a872c3b:~/openssh-8.3p1# ssh -V
OpenSSH_?, OpenSSL 1.1.1f 31 Mar 2020

```

Figura 21: Versión personalizada.

6. Finalmente, se estableció una conexión SSH hacia el servidor (C4/S1) utilizando el cliente modificado, y se capturó el tráfico generado con **wireshark**.



Source	Destination	Protocol	Length	Info
172.19.0.5	172.19.0.3	TCP	74	41180 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=
172.19.0.3	172.19.0.5	TCP	74	22 → 41180 [SYN, ACK] Seq=0 Ack=1 Win=6510
172.19.0.5	172.19.0.3	TCP	66	41180 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0
172.19.0.5	172.19.0.3	SSHv2	85	Client: Protocol (SSH-2.0-OpenSSH_?)
172.19.0.3	172.19.0.5	TCP	66	22 → 41180 [ACK] Seq=1 Ack=20 Win=65152 Len=0
172.19.0.3	172.19.0.5	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu
172.19.0.5	172.19.0.3	TCP	66	41180 → 22 [ACK] Seq=20 Ack=42 Win=64256 Len=0
172.19.0.5	172.19.0.3	SSHv2	1578	Client: Key Exchange Init
172.19.0.3	172.19.0.5	SSHv2	1146	Server: Key Exchange Init
172.19.0.5	172.19.0.3	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key
172.19.0.3	172.19.0.5	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key
172.19.0.5	172.19.0.3	SSHv2	82	Client: New Keys
172.19.0.3	172.19.0.5	TCP	66	22 → 41180 [ACK] Seq=1718 Ack=1596 Win=64256
172.19.0.5	172.19.0.3	SSHv2	110	Client:

Figura 22: Tráfico SSH generado mostrando la versión personalizada OpenSSH\_?.

### 3. Desarrollo (Parte 3)

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

Para reducir el tamaño del mensaje **Key Exchange Init (KEI)** generado por el servidor OpenSSH a menos de 300 bytes, se realizaron las siguientes configuraciones directamente desde la consola del servidor ubicado en el contenedor C4/S1:

1. **Acceso al contenedor del servidor:** Se ingresó al contenedor C4/S1 utilizando el siguiente comando:

```
docker exec -it c4_s1_container bash
```

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

2. **Edición del archivo de configuración del servidor:** Se abrió el archivo de configuración del servidor OpenSSH, ubicado en `/etc/ssh/sshd_config`, con el editor de texto `nano`:

```
nano /etc/ssh/sshd_config
```

En este archivo, se modificaron los algoritmos utilizados para optimizar el tamaño del mensaje KEI. Las líneas agregadas fueron:

```
KexAlgorithms curve25519-sha256
Ciphers aes128-ctr
MACs umac-64@openssh.com
Compression no
```

3. **Ajuste de permisos:** Para garantizar que el archivo de configuración sea leído correctamente por el servicio SSH, se ajustaron los permisos con el siguiente comando:

```
chmod 600 /etc/ssh/sshd_config
```

4. **Reinicio del servicio SSH:** Se aplicaron los cambios reiniciando el servicio OpenSSH:

```
service ssh restart
```

Con el procedimiento descrito, se capturó el tráfico utilizando Wireshark, obteniendo el siguiente resultado:

Destination	Protocol	Length	Info
172.19.0.3	TCP	74	50636 → 22 [SYN] Seq=0 Win=64240 Le
172.19.0.4	TCP	74	22 → 50636 [SYN, ACK] Seq=0 Ack=1 W
172.19.0.3	TCP	66	50636 → 22 [ACK] Seq=1 Ack=1 Win=64
172.19.0.3	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7
172.19.0.4	TCP	66	22 → 50636 [ACK] Seq=1 Ack=42 Win=6
172.19.0.4	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9
172.19.0.3	TCP	66	50636 → 22 [ACK] Seq=42 Ack=42 Win=
172.19.0.3	SSHv2	570	Client: Key Exchange Init
172.19.0.4	SSHv2	282	Server: Key Exchange Init
172.19.0.3	SSHv2	114	Client: Elliptic Curve Diffie-Hellm
172.19.0.4	SSHv2	658	Server: Elliptic Curve Diffie-Hellm
172.19.0.3	SSHv2	82	Client: New Keys
172.19.0.4	TCP	66	22 → 50636 [ACK] Seq=850 Ack=610 Wi

Figura 23: Captura de Wireshark del mensaje Key Exchange Init modificado en el servidor.

## 4. Desarrollo (Parte 4)

## 5. Desarrollo (Parte 4)

### 5.1. Explicación OpenSSH en general

OpenSSH es una implementación libre y completa del protocolo Secure Shell (SSH), diseñada para garantizar comunicaciones seguras en redes, incluso en entornos no confiables. Se utiliza ampliamente para administrar sistemas de manera remota, transferir archivos y establecer túneles seguros mediante reenvío de puertos. Su funcionalidad principal incluye el cifrado de datos, la autenticación de usuarios y servidores, y la integridad de las transmisiones, protegiendo la información contra accesos no autorizados y modificaciones.

El protocolo SSH opera a través de un proceso de varias etapas: primero, cliente y servidor intercambian sus versiones del protocolo para establecer compatibilidad. Luego, negocian los algoritmos criptográficos a usar en la conexión, lo que incluye métodos de intercambio de claves, cifrado, autenticación de mensajes y compresión. Una vez seleccionados los algoritmos, se procede al intercambio seguro de claves mediante métodos como Diffie-Hellman o Curve25519. Finalmente, se realiza la autenticación del usuario, que puede ser mediante contraseñas, claves públicas u otros métodos soportados. OpenSSH se ha convertido en un estándar por su robustez, flexibilidad y enfoque en la seguridad.

### 5.2. Capas de Seguridad en OpenSSH

OpenSSH incorpora múltiples capas de seguridad que garantizan la confidencialidad, integridad y autenticidad de las comunicaciones. La confidencialidad se logra mediante algoritmos de cifrado como AES y ChaCha20-Poly1305, que aseguran que los datos transmitidos no puedan ser leídos por terceros. La integridad se verifica a través de códigos de autenticación de mensajes (MAC), como HMAC-SHA256 o UMAC, que detectan modificaciones no autorizadas en los datos.

Además, OpenSSH implementa autenticación tanto para el servidor como para el usuario. La autenticación del servidor, realizada mediante claves host, asegura que el cliente se conecta al servidor legítimo. Por su parte, la autenticación del usuario puede realizarse mediante contraseñas, claves públicas u otros métodos, lo que garantiza que solo usuarios autorizados accedan al sistema. Aunque la disponibilidad no es directamente gestionada por OpenSSH, el monitoreo y la configuración adecuada del servidor pueden prevenir ataques que comprometan el servicio, como los ataques de fuerza bruta o denegación de servicio (DoS). Finalmente, el protocolo permite rastrear la autenticidad de las acciones realizadas mediante claves criptográficas, asegurando el principio de no repudio.

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

A través de las pruebas realizadas en diferentes versiones de OpenSSH, se detectaron algunas limitaciones en el cumplimiento de ciertos principios de seguridad del protocolo. En

versiones más antiguas, como las instaladas en contenedores con Ubuntu 16.10 y 18.10, la compatibilidad con algoritmos modernos como Curve25519 puede ser limitada, lo que lleva a la selección de algoritmos menos seguros como Diffie-Hellman en algunas conexiones.

Asimismo, en algunos escenarios el tamaño del mensaje Key Exchange Init (KEI) es mayor al esperado, debido a la negociación de múltiples algoritmos en lugar de una selección optimizada. Esto no solo afecta el rendimiento, sino que también expone la conexión a una mayor superficie de ataque al listar algoritmos obsoletos. Por ejemplo, algoritmos como Diffie-Hellman Group Exchange siguen presentes en estas configuraciones, a pesar de ser menos seguros que alternativas modernas. Estas observaciones resaltan la necesidad de utilizar versiones actualizadas de OpenSSH y de realizar configuraciones específicas para garantizar la seguridad.

## Conclusiones y comentarios

Los resultados obtenidos en este laboratorio resaltan que, aunque OpenSSH es una herramienta confiable y robusta, su desempeño y nivel de seguridad están directamente vinculados a la versión utilizada y a la configuración aplicada. En entornos con versiones antiguas, como las analizadas en este experimento, la presencia de algoritmos criptográficos obsoletos puede comprometer la seguridad, aumentando la superficie de ataque. Por esta razón, ajustar manualmente la configuración para priorizar algoritmos modernos y eficientes, como `curve25519-sha256`, es esencial para reducir vulnerabilidades y mejorar el rendimiento del protocolo.

El monitoreo y análisis del tráfico mediante herramientas como Wireshark nos ayudaron a identificar las discrepancias en la negociación de algoritmos y validar la efectividad de las configuraciones aplicadas. La captura y estudio del tráfico evidenciaron cómo ajustes específicos, como la optimización del mensaje Key Exchange Init (KEI), pueden impactar directamente en la eficiencia de la comunicación, reduciendo el tamaño del mensaje y reforzando la seguridad de las conexiones.

Este laboratorio nos remarca la importancia de mantener configuraciones adecuadas y actualizadas. La capacidad de analizar, ajustar y validar de forma continua permite no solo garantizar comunicaciones seguras en entornos adversos, sino también responder de manera proactiva a los desafíos que plantean tecnologías en constante evolución.