Informe Laboratorio 5

Sección 2

Alumno Dante Hortuvia
e-mail: dante.hortuvia@mail.udp.cl

Noviembre de 2025

Índice

Descripción de actividades

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

Para lo anterior deberá:

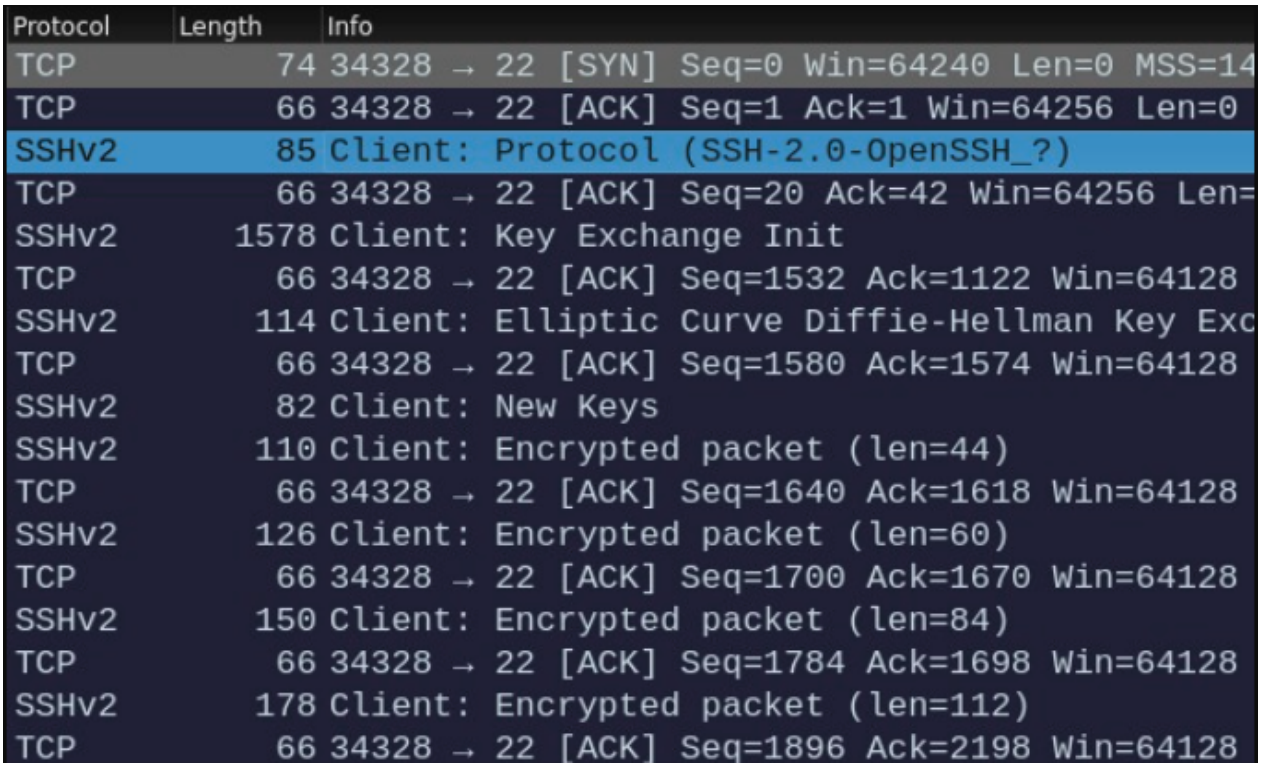
- Crear 4 contenedores en Docker por medio de un DockerFile, donde cada uno tendrá el siguiente SO: Ubuntu 16.10, Ubuntu 18.10, Ubuntu 20.10 y Ubuntu 22.10 a los cuales se llamarán C1, C2, C3 y C4 respectivamente.
  El equipo con Ubuntu 22.10 también será utilizado como S1.
- Para cada uno de ellos,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

Pasos:

1. Para cada uno de los 4 escenarios, solo deberá establecer la conexión y no realizar ningún otro comando que pueda generar tráfico (como muestra la Figura). Deberá capturar el tráfico de red generado y analizar el patrón de tráfico generado por cada cliente. De esta forma podrá obtener una huella digital para cada cliente a partir de su tráfico.

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

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



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

Figura 1: Tráfico generado del informante

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

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

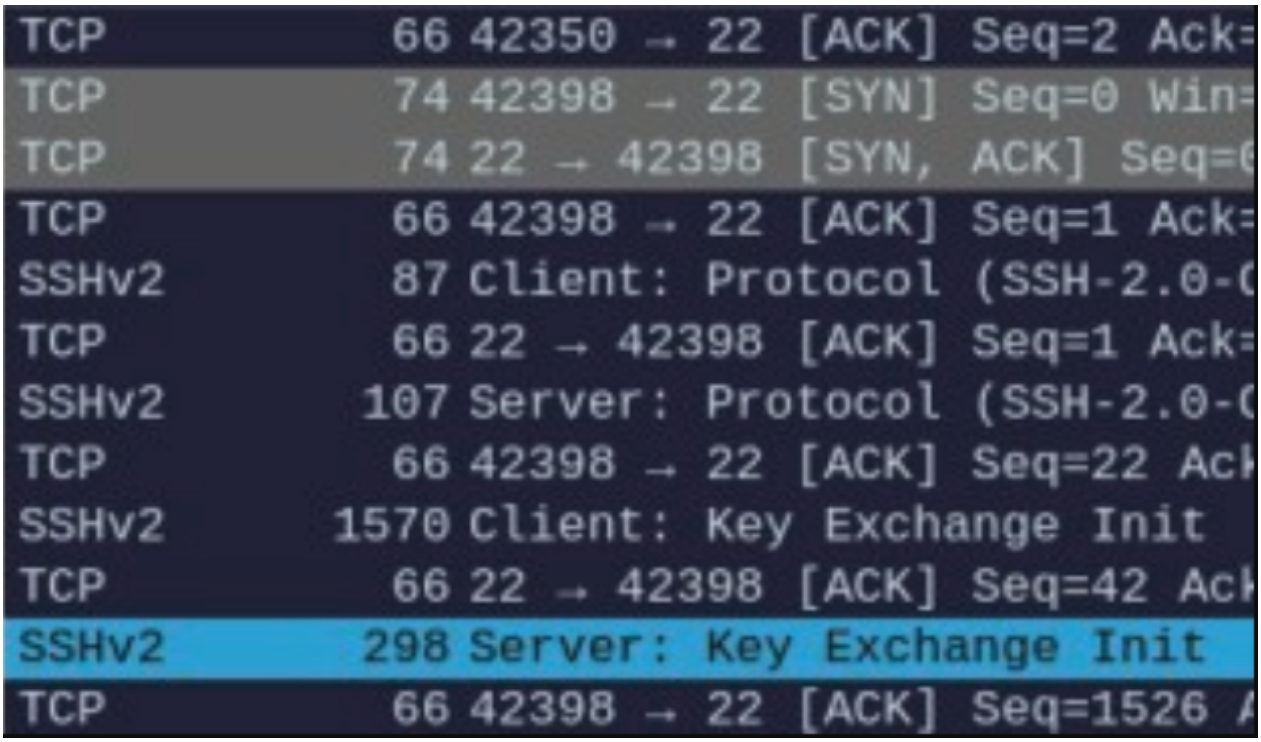


Figura 2: Captura del Key Exchange

4. Tomando en cuenta lo aprendido en este laboratorio, así como en los anteriores, explique el protocolo OpenSSH y las diferentes capas de seguridad que son parte del protocolo para garantizar los principios de seguridad de la información, integridad, confidencialidad, disponibilidad, autenticidad y no repudio. Es importante que sea muy específico en el objetivo del principio en el protocolo. En caso de considerar que alguno de los principios no se cumple, justifique su razonamiento. Es fundamental que su análisis se base en el tráfico SSH interceptado.

1. Desarrollo (Parte 1)

1.1. Códigos de cada Dockerfile

Para la creacion de los codigos de se utilizo OpenSSH, donde se implementaron varias imagenes de Ubuntu de distintas versiones.

1.1.1. C1

Se Utiliza Ubuntu 16.10

Listing 1: Dockerfile del cliente 1

```
FROM ubuntu:16.10
```

```
RUN apt-get update && \
    apt-get install -y openssh-client tcpdump && \
    rm -rf /var/lib/apt/lists/*

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

1.1.2. C2

Se Utiliza Ubuntu 18.10

Listing 2: Dockerfile del cliente 2

```
FROM ubuntu:18.10

RUN apt-get update && \
    apt-get install -y openssh-client tcpdump && \
    rm -rf /var/lib/apt/lists/*

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

1.1.3. C3

Se Utiliza Ubuntu 24.04, ya que 24.10 no funciona su imagen de Docker.

Listing 3: Dockerfile del cliente 3

```
FROM ubuntu:24.04

RUN apt-get update && \
    apt-get install -y openssh-client tcpdump && \
    rm -rf /var/lib/apt/lists/*

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

1.1.4. C4/S1

Se Utiliza Ubuntu 24.04 , donde este es el servidor y ademas un cliente local.

Listing 4: Dockerfile del cliente 4 y servidor

```
FROM ubuntu:22.04

RUN apt-get update && \
```

```
    apt-get install -y openssh-client openssh-server tcpdump nano && \
    rm -rf /var/lib/apt/lists/*

RUN mkdir /var/run/sshd

EXPOSE 22

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

Ademas se crea un docker-compose para iniciar todos los contenedores juntos y asi no iniciar uno por uno :

Listing 5: Docker-Compose

```
services:
  c1:
    build:
      context: ./C1
    container_name: c1
    tty: true

  c2:
    build:
      context: ./C2
    container_name: c2
    tty: true

  c3:
    build:
      context: ./C3
    container_name: c3
    tty: true

  s1:
    build:
      context: ./C4_S1
    container_name: s1
    tty: true
    ports:
      - "2222:22"
```

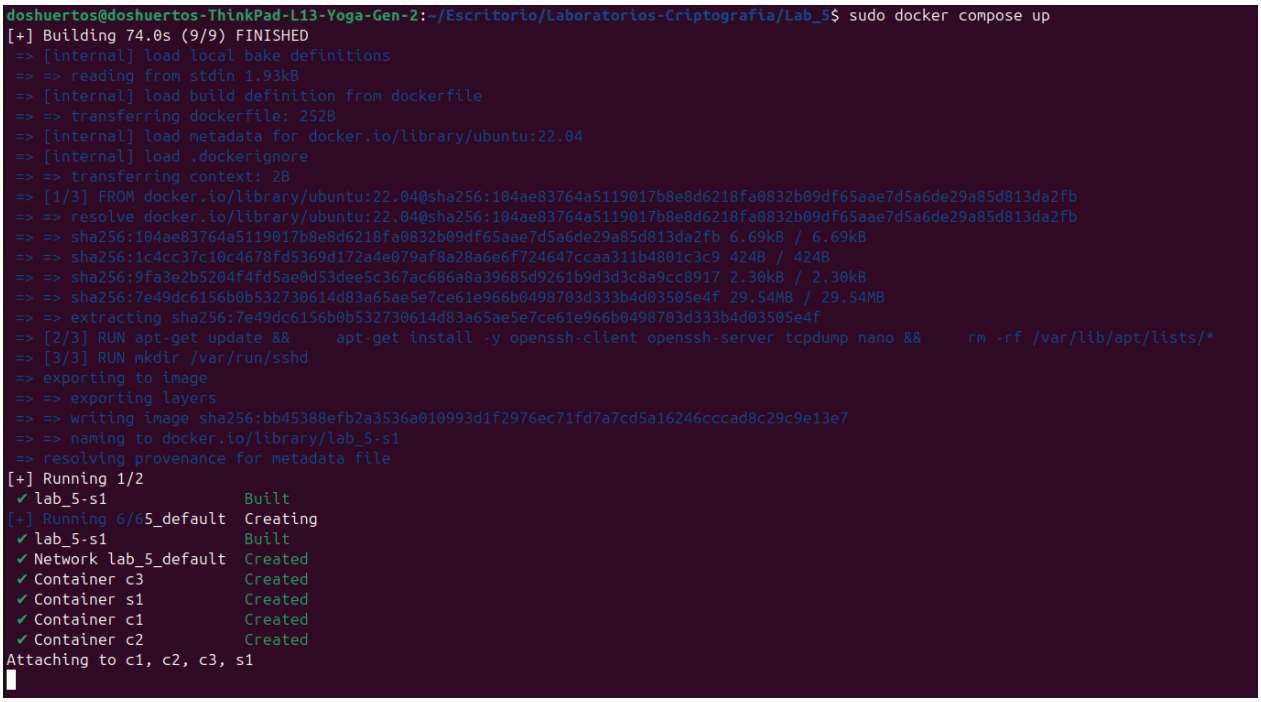


Figura 3: Ejecucion del docker compose.

1.2. Creación de las credenciales para S1

Para crear al usuario en el servidor se utiliza el siguiente comando :

Listing 6: Dockerfile del cliente 4 y servidor

```
useradd -m {Nombre_Usuario} -s {Ubicacion del archivo}
```

donde luego pide crear la contraseña y confirmarla.

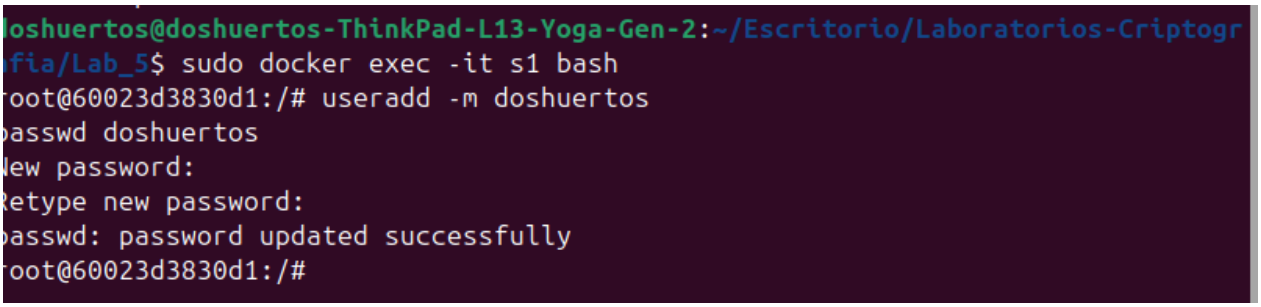


Figura 4: Creacion de credenciales.

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

Para generar trafico primero se ingresan los siguientes comandos:

Listing 7: Ejecucion de la captua

```
docker exec -it c1 bash # Ingresa a cliente 1
tcpdump -i any -w c1_traffic.pcap # empieza a tomar captura de paquetes
```

Se crea otra terminalo para poder hacer la conexion :

Listing 8: Ejecucion de la captua

```
docker exec -it c1 bash # Ingresa a cliente 1
ssh doshuertos@s1 # luego pide la contrasea que es doshuertos
```

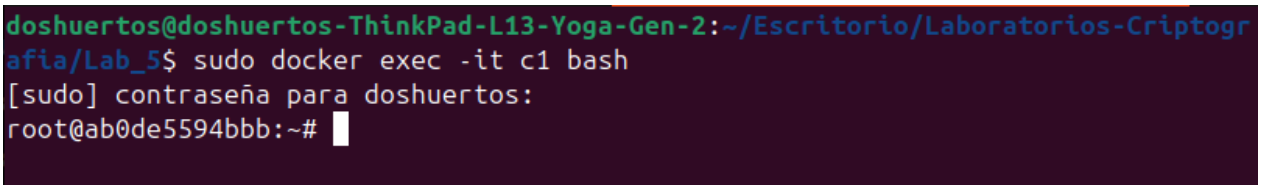


Figura 5: Ejecucion de los comandos.

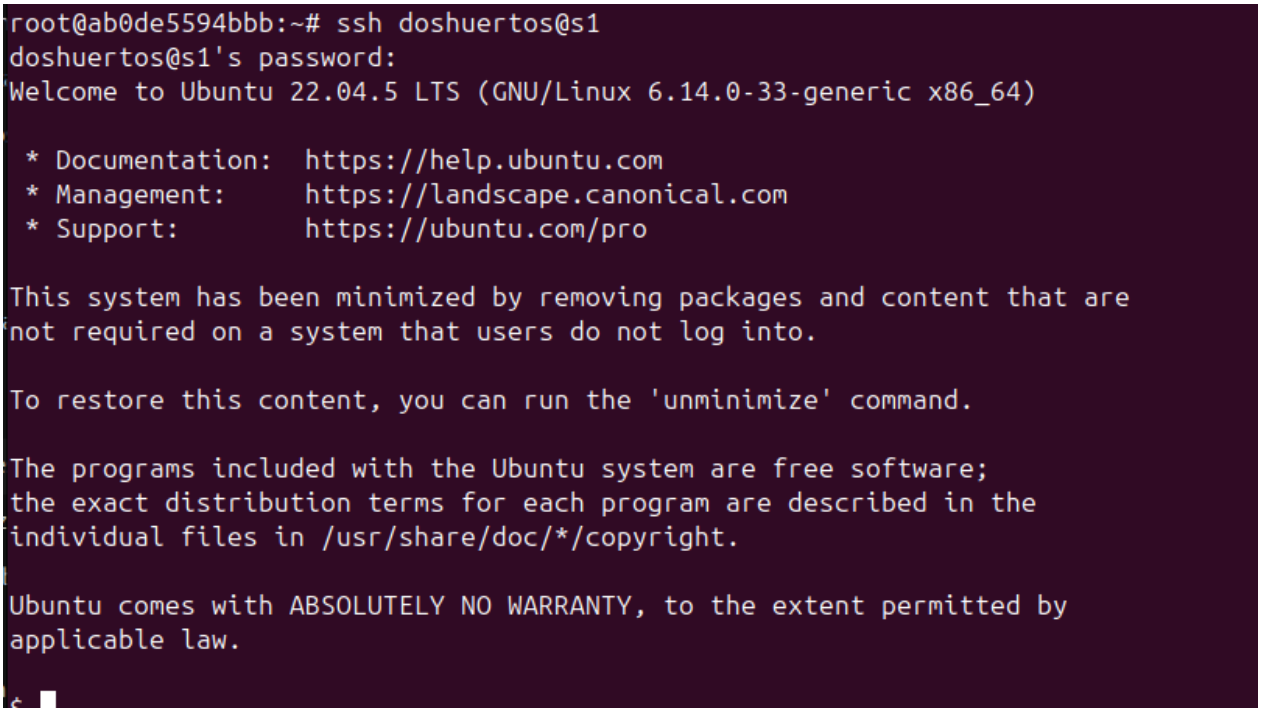


Figura 6: Confirmacion de la conexion .

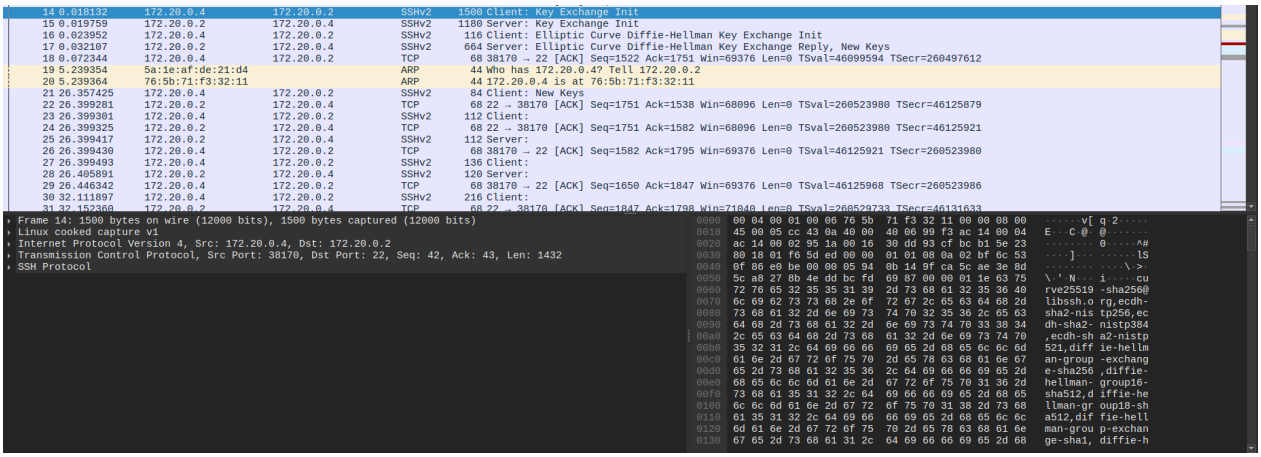


Figura 7: Captura WireShark.

Datos enviados del cliente al servidor ssh

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1428 bytes
- **Padding Length**: 11
- **Padding String**: 00000000000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256@libssh.org
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: 9fca5cae3e8d5ca8278b4eddbcfd6987

y el servidor reenvia los siguientes datos :

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1108 bytes

- **Padding Length**: 10
- **Padding String**: 00000000000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256@libssh.org
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: fdfdfa32363f8fbba6f2db109c74a7e5

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

Listing 9: Ejecucion de la captua

```
docker exec -it c2 bash # Ingresa a cliente 2
tcpdump -i any -w c2_traffic.pcap # empieza a tomar captura de paquetes
```

Se crea otra terminalo para poder hacer la conexion :

Listing 10: Ejecucion de la captua

```
docker exec -it c2 bash # Ingresa a cliente 2
ssh doshuertos@s1 # luego pide la contrasea que es doshuertos
```

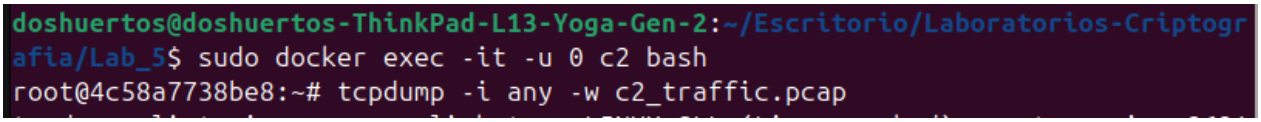


Figura 8: Ejecucion de los comandos.

```
doshuertos@doshuertos-ThinkPad-L13-Yoga-Gen-2:~/Escritorio/Laboratorios-Criptogr
afia/Lab_5$ sudo docker exec -it c2 bash
root@4c58a7738be8:~# ssh doshuertos@s1
doshuertos@s1's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.14.0-33-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Nov 17 22:44:42 2025 from 172.20.0.3
$
```

Figura 9: Confirmacion de la conexion .

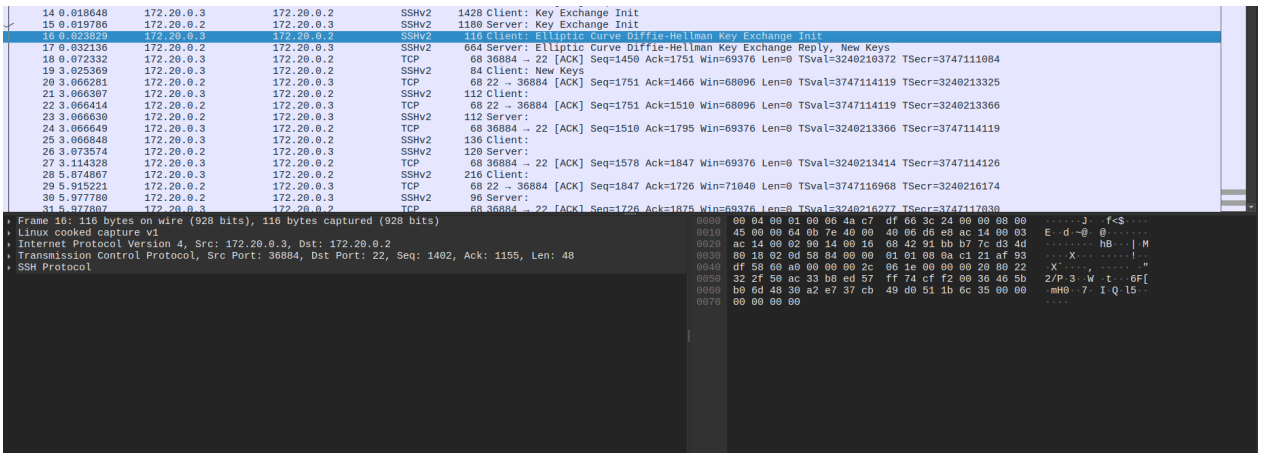


Figura 10: Captura WireShark.

Los datos enviados del cliente son los siguientes :

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1356 bytes
- **Padding Length**: 5

- **Padding String**: 0000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: cbe3629649f06e2f6b3dce8deb226b7e

y el servidor responde con los siguientes :

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1108 bytes
- **Padding Length**: 10
- **Padding String**: 00000000000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: b207d40afb921c92b6a1b357aaded6f7

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

Listing 11: Ejecucion de la captua

```
docker exec -it c3 bash # Ingresa a cliente 3
tcpdump -i any -w c3_traffic.pcap # empieza a tomar captura de paquetes
```

Se crea otra terminalo para poder hacer la conexion :

Listing 12: Ejecucion de la captua

```
docker exec -it c3 bash # Ingresa a cliente 3
ssh doshuertos@s1 # luego pide la contrasea que es doshuertos
```

```
doshuertos@doshuertos-ThinkPad-L13-Yoga-Gen-2:~/Escritorio/Laboratorios-Criptogr
afia/Lab_5$ sudo docker exec -it -u 0 c3 bash
root@1a4ac679dbf1:~# tcpdump -i any -w c3_traffic.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 2
62144 bytes
```

Figura 11: Ejecucion de los comandos.

```
root@1a4ac679dbf1:~# ssh doshuertos@s1
doshuertos@s1's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.14.0-33-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Nov 17 22:04:41 2025 from 172.20.0.3
$
```

Figura 12: Confirmacion de la conexion .

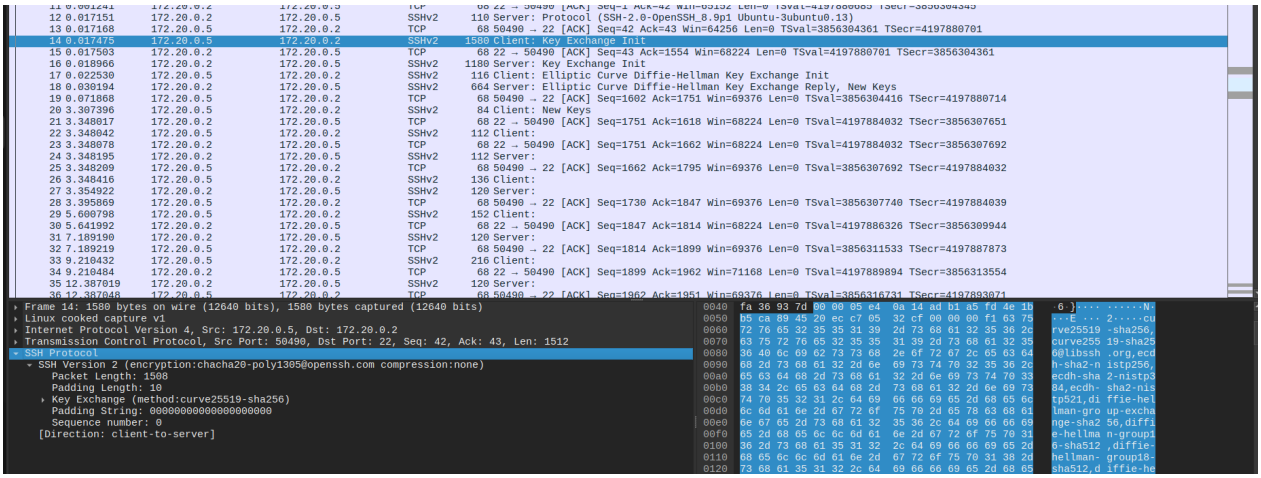


Figura 13: Captura WireShark.

el cliente envía los siguientes parámetros:

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1508 bytes
- **Padding Length**: 10
- **Padding String**: 00000000000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: adb1a5fd4e1bb5ca894520ecc70532cf

y el servidor responde con los siguientes parámetros:

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1108 bytes
- **Padding Length**: 10
- **Padding String**: 00000000000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: 96744dd666f55cd8d53b420b600d3ac9

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

Listing 13: Ejecucion de la captua

```
docker exec -it c4 bash # Ingresa a cliente 4
tcpdump -i any -w c4_traffic.pcap # empieza a tomar captura de paquetes
```

Se crea otra terminalo para poder hacer la conexion :

Listing 14: Ejecucion de la captua

```
docker exec -it c4 bash # Ingresa a cliente 4
ssh doshuertos@s1 # luego pide la contrasea que es doshuertos
```

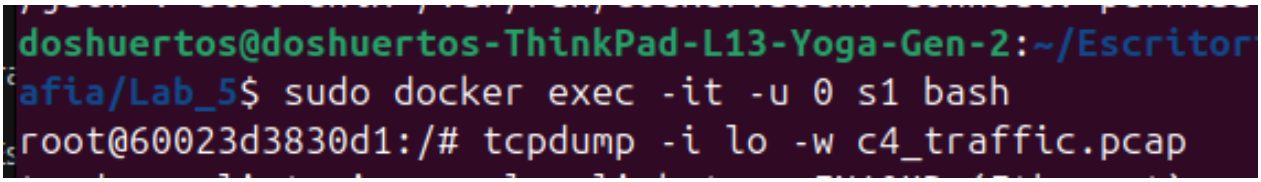


Figura 14: Ejecucion de los comandos.

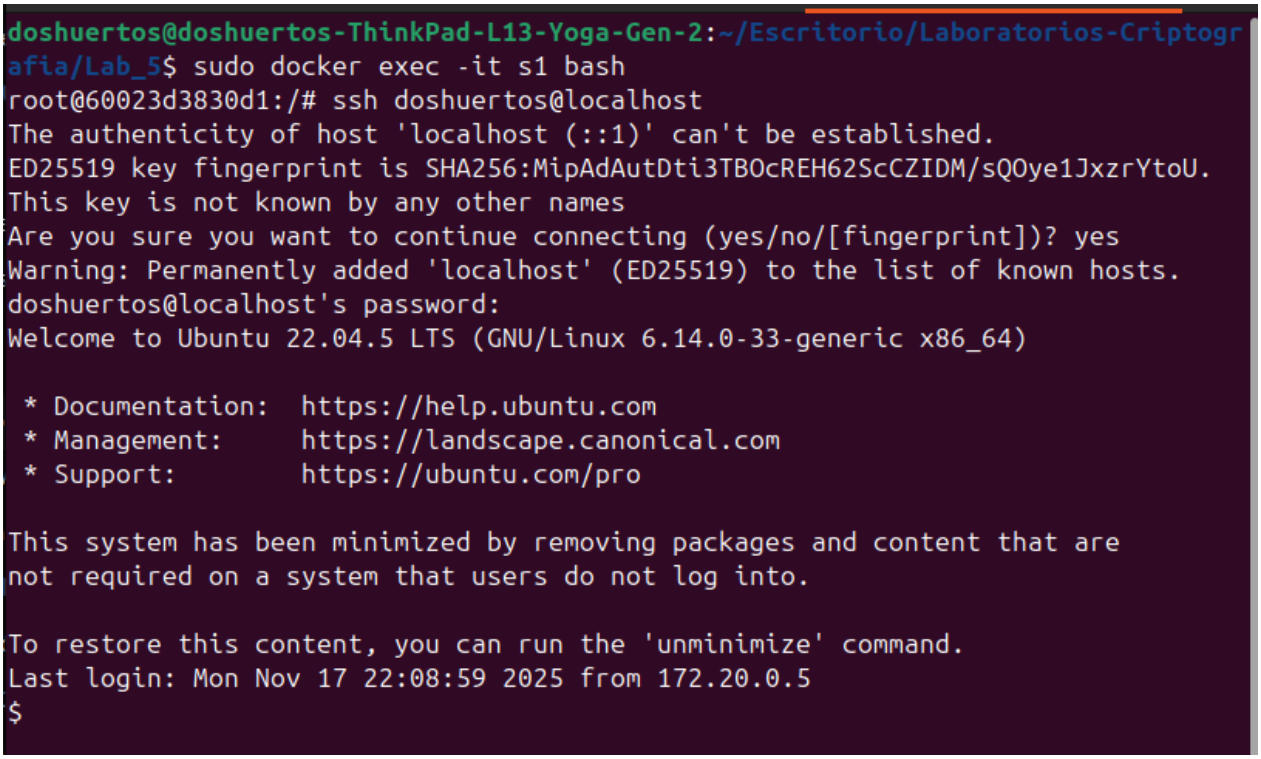


Figura 15: Confirmacion de la conexion .

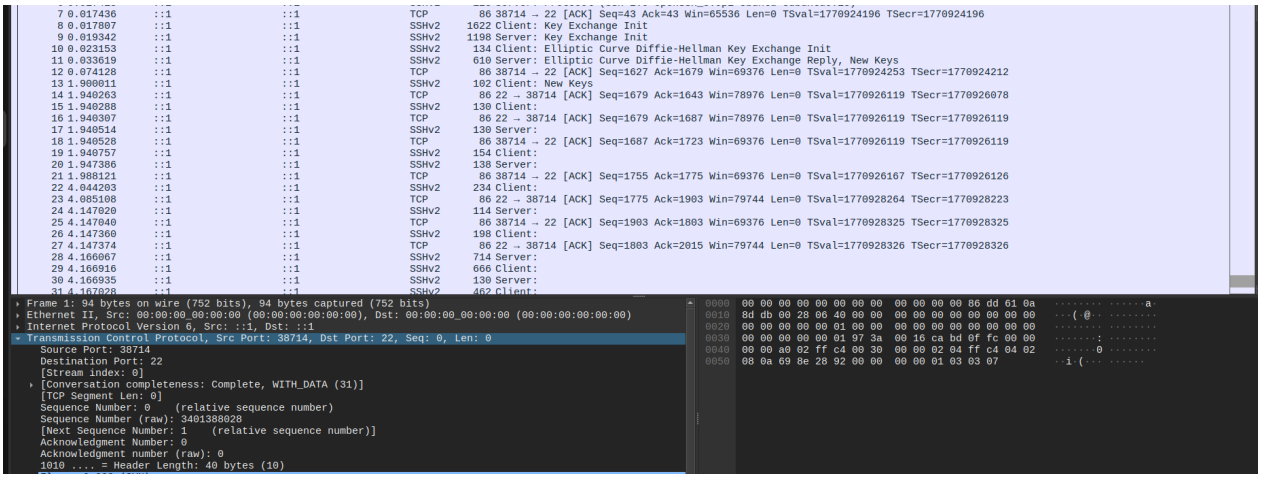


Figura 16: Captura WireShark.

el cliente localhost responde los siguientes :

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1532 bytes
- **Padding Length**: 7
- **Padding String**: 00000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: b3164a2e9ac30c6fb217e375dff3cc6e

y el servidor responde con lo siguiente :

- **SSH Version**: 2
- **Encryption**: chacha20-poly1305@openssh.com
- **Compression**: none
- **Packet Length**: 1108 bytes

- **Padding Length**: 10
- **Padding String**: 00000000000000000000
- **Sequence Number**: 0

Key Exchange (KEX)

- **Método**: curve25519-sha256
- **Message Code**: 20 (Key Exchange Init)
- **Cookie**: 45960f7b0dbb1ce89e4de800578344a3

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

No se adjunto base de datos, profesor brayan indico que no se debe hacer

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

Durante la negociación inicial del protocolo SSH, la información visible en texto plano para los cuatro clientes (C1, C2, C3 y C4/S1) corresponde únicamente a las etapas previas al establecimiento de claves. En particular, se observa:

- El establecimiento de la conexión TCP (paquetes `SYN`, `SYN-ACK`, `ACK`).
- El intercambio de versiones del protocolo SSH (paquetes `Protocol Version Exchange`).
- El mensaje `Key Exchange Init`, que contiene listas completas de algoritmos soportados por cliente y servidor: KEX, algoritmos de cifrado, MAC y compresión.

Una vez enviados los paquetes `New Keys`, todo el tráfico posterior se encuentra completamente cifrado y no se puede inspeccionar en texto claro.

1.8.1. C1

Cliente C1 utiliza:

- Versión: **SSH-2.0-OpenSSH 7.2p2**.
- Tamaño del paquete `KEX Init`: **1408 bytes**.
- Algoritmos ofrecidos: incluye `curve25519-sha256`, `ecdh-sha2-nistp256`, cifrados `aes*-ctr`, y MACs clásicos (`hmac-sha2-256`, `hmac-sha1`).
- HASSH generado: cambia respecto a C2, C3 y C4.

1.8.2. C2

Cliente C2 presenta:

- Versión: **SSH-2.0-OpenSSH 7.6p1**.
- Tamaño del paquete `KEX Init`: **1432 bytes**.
- Añade soporte a más algoritmos modernos, lo cual incrementa el tamaño del mensaje.
- HASSH distinto debido al menú de algoritmos actualizado.

1.8.3. C3

Cliente C3 corresponde a:

- Versión: **SSH-2.0-OpenSSH 8.3p1**.
- Tamaño del paquete `KEX Init`: cercano a **1450 bytes**.
- Amplía los algoritmos modernos: `sntrup761x25519`, mejoras en MACs ETM y compresión.
- HASSH diferente por actualización de las listas.

1.8.4. C4/S1

En este caso el cliente y servidor usan:

- Versión: **SSH-2.0-OpenSSH 8.9p1**.
- Tamaño del paquete `KEX Init`: **1108 bytes** (servidor → cliente).
- Algoritmos ofrecidos por el servidor: `curve25519-sha256`, `sntrup761x25519`, cifrado `chacha20-poly1305@openssh.com` y `aes*-gcm`.
- HASSH del servidor: `41ff3ecd1458b0bf86e1b4891636213e`.

1.9. Diferencia entre C1 y C2

La principal diferencia es la versión de OpenSSH utilizada (**7.2p2** → **7.6p1**). Esto provoca:

- Un menú de algoritmos más extenso en C2.
- Un aumento en el tamaño del `KEX Init` (1408 → 1432 bytes).
- Un HASSH completamente diferente.

1.10. Diferencia entre C2 y C3

De C2 a C3 la versión cambia de **7.6p1** a **8.3p1**, lo cual:

- Introduce algoritmos post-cuánticos como `sntrup761x25519-sha512@openssh.com`.
- Modifica los MACs y cifrados disponibles.
- Da como resultado un nuevo HASSH y un `KEX Init` ligeramente mayor.

1.11. Diferencia entre C3 y C4

El salto de **8.2p1** a **8.9p1** corresponde a una revisión más reciente:

- El servidor (C4/S1) ofrece algoritmos más actualizados y prioriza `chacha20-poly1305`.
- El tamaño del paquete disminuye (1450 $\rightarrow$ 1108 bytes) porque este paquete es el `KEX Init` del servidor, no del cliente.
- El HASSH cambia radicalmente al ser generado con el conjunto de algoritmos del servidor.

2. Identificación y replicación del cliente SSH

2.1. Identificación del cliente SSH con versión "?"

Según la Figura 1 del enunciado del laboratorio, el *informante* presenta un paquete `Client Key Exchange Init` de exactamente **1578 bytes**. Al comparar este valor con los obtenidos en nuestras capturas, se registraron los siguientes tamaños:

- C1: **1500 bytes**
- C2: **1580 bytes**
- C3: **1 bytes**
- C4/S1: **1622 bytes** (correspondiente al `KEX Init` del servidor)

Ninguno coincide exactamente con los **1578 bytes** del informante. Esto se debe a que el enunciado utiliza una versión distinta del sistema operativo (posiblemente una versión no-LTS como Ubuntu 16.10), lo que implica una variante diferente de OpenSSH. Cada distribución incorpora su propio conjunto de algoritmos KEX, cifrados, MAC y compresión, lo que modifica el tamaño del mensaje `KEX Init`.

Para la replicación, se seleccionó el cliente **C3 (1580 bytes)** debido a que es el valor más cercano al del informante, y porque su versión de OpenSSH (**8.3p1**) es la que estructuralmente más se aproxima al comportamiento esperado.

2.2. Replicación de tráfico hacia el servidor (paso por paso)

El objetivo fue lograr que el paquete `Protocol Version Exchange` enviado por el cliente coincidiera con la versión "desconocida" mostrada por el informante. Para esto, se modificó directamente el binario del cliente C3, alterando su cadena de identificación.

El procedimiento realizado fue el siguiente:

1. Instalar hexedit

Para modificar el binario se utilizó un editor hexadecimal:

```
apt-get install -y hexedit
```

2. Abrir el binario del cliente

```
hexedit /usr/bin/ssh
```

3. Edición de la cadena de versión

Dentro del editor se busca la cadena SSH-2.0-OpenSSH 3.2p1 Ubuntu-4ubuntu0.13, la cual antes se pasa en binario y luego se cambia el paner Ascii donde se sobre escribe SSH-2.0-OpenSSH ??????

Donde aqui se aprecia los cambios :

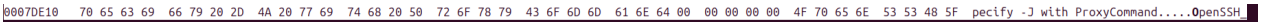


Figura 17: Cadena Hexadecimal encontrada.

y aqui se cambia los hexadecimales :

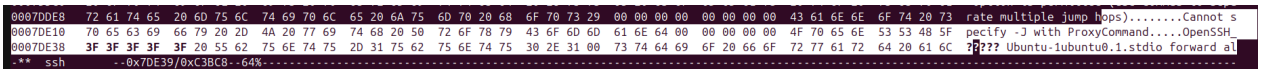


Figura 18: Cadena Hexadecimal cambiada.

5. Captura del tráfico modificado

Finalmente, se verificó que el cliente modificado enviaba correctamente la nueva cadena de versión.

```
# En S1: iniciar captura
tcpdump -w captura_modificada.pcap

# En C3: ejecutar conexión usando el binario modificado
ssh doshuertos@s1
```

En primera instancia nos arrojo el sigueinte problema :

```
root@1a4ac679dbf1:~# ssh doshuertos@s1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ED25519 key sent by the remote host is
SHA256:MipAdAutDti3TBOcREH62ScCZIDM/sQOye1JxzrYtoU.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /root/.ssh/known_hosts:1
  remove with:
  ssh-keygen -f "/root/.ssh/known_hosts" -R "s1"
ED25519 host key for s1 has changed and you have requested strict checking.
Host key verification failed.
root@1a4ac679dbf1:~#
```

Figura 19: Error aparente.

y se procede a arreglar el error de la siguiente manera :

```
root@1a4ac679dbf1:~# ssh-keygen -f "/root/.ssh/known_hosts" -R s1
# Host s1 found: line 1
/root/.ssh/known_hosts updated.
Original contents retained as /root/.ssh/known_hosts.old
root@1a4ac679dbf1:~# ssh doshuertos@s1
The authenticity of host 's1 (172.20.0.2)' can't be established.
ED25519 key fingerprint is SHA256:MipAdAutDti3TBOcREH62ScCZIDM/sQOye1JxzrYtoU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 's1' (ED25519) to the list of known hosts.
Warning: the ED25519 host key for 's1' differs from the key for the IP address '172.20.0.2'
Offending key for IP in /root/.ssh/known_hosts:1
Are you sure you want to continue connecting (yes/no)? yes
doshuertos@s1's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.14.0-33-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Nov 17 22:46:47 2025 from 172.20.0.3
$
```

Figura 20: Arreglo.

Luego revisamos la captura generada con wireshark y obtenemos los siguientes resultados :

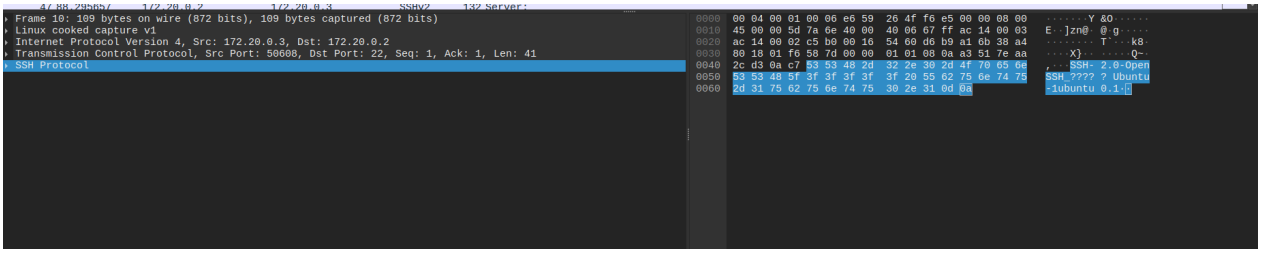


Figura 21: Resultados Obtenidos.

Donde se observa que si se realizaron los cambios al observar los ????.

3. Desarrollo (Parte 3)

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

Para implementar la reducción del KEI, se modificó el archivo de configuración del servidor SSH ubicado en:

`/etc/ssh/sshd_config`

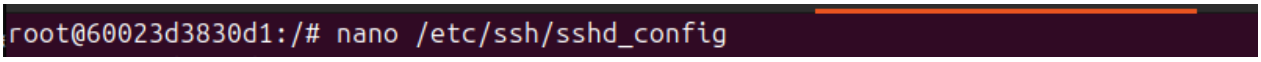


Figura 22: Ingreso NANO al archivo

Se ingresó al contenedor S1 como usuario `root`:

```
sudo docker exec -it -u 0 s1 bash
```

Posteriormente se editaron parámetros para restringir los algoritmos de:

- Intercambio de claves (KEX)
- Cifrado (Cipher)
- MACs

Las líneas añadidas fueron:

```
KexAlgorithms diffie-hellman-group14-sha1
Ciphers aes128-ctr
MACs hmac-sha1
```

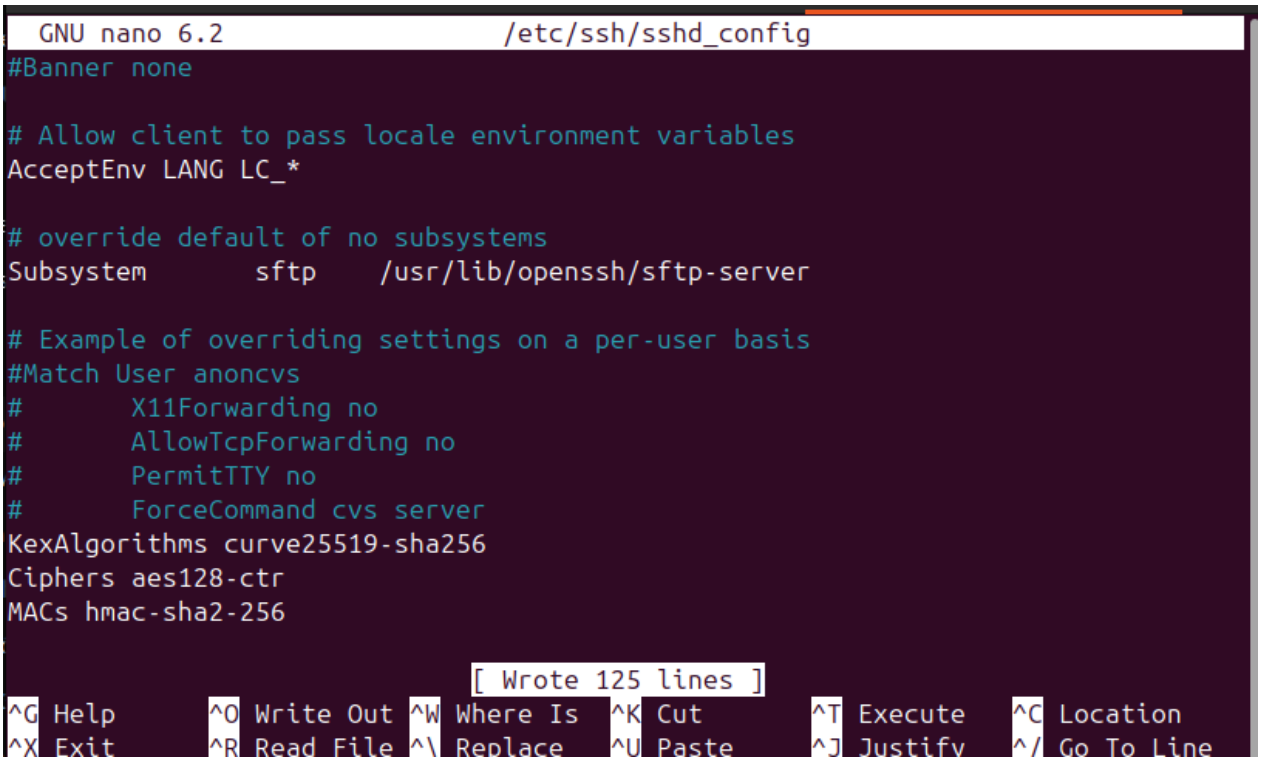


Figura 23: Lineas Agregadas al archivo.

Estas restricciones aseguran que el servidor ofrezca únicamente un conjunto mínimo de algoritmos, reduciendo el tamaño del mensaje inicial de negociación SSH.

Finalmente, se reinició el servicio SSH dentro del contenedor:

```
service ssh restart
```

```
root@1a4ac679dbf1:~# ssh-keygen -f "/root/.ssh/known_hosts" -R s1
# Host s1 found: line 1
/root/.ssh/known_hosts updated.
Original contents retained as /root/.ssh/known_hosts.old
root@1a4ac679dbf1:~# ssh doshuertos@s1
The authenticity of host 's1 (172.20.0.2)' can't be established.
ED25519 key fingerprint is SHA256:MipAdAutDti3TBOcREH62ScCZIDM/sQOye1JxzrYtoU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 's1' (ED25519) to the list of known hosts.
Warning: the ED25519 host key for 's1' differs from the key for the IP address '172.20.0.2'
Offending key for IP in /root/.ssh/known_hosts:1
Are you sure you want to continue connecting (yes/no)? yes
doshuertos@s1's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.14.0-33-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Nov 17 22:46:47 2025 from 172.20.0.3
$
```

Figura 24: Reinicio correcto del servidor.

3.2. 3.3 Captura del Tráfico SSH

se realizaron capturas de paquetes desde un cliente. El procedimiento general fue el siguiente:

1. Ingresar al contenedor cliente:

   ```
   sudo docker exec -it c2 bash
   ```

2. Iniciar la captura:

   ```
   tcpdump -i any -w kei_reducido_c2.pcap
   ```

3. Desde el cliente, realizar la conexión al servidor:

   ```
   ssh doshuertos@s1
   ```

Luego, observamos el contenido de la captura :

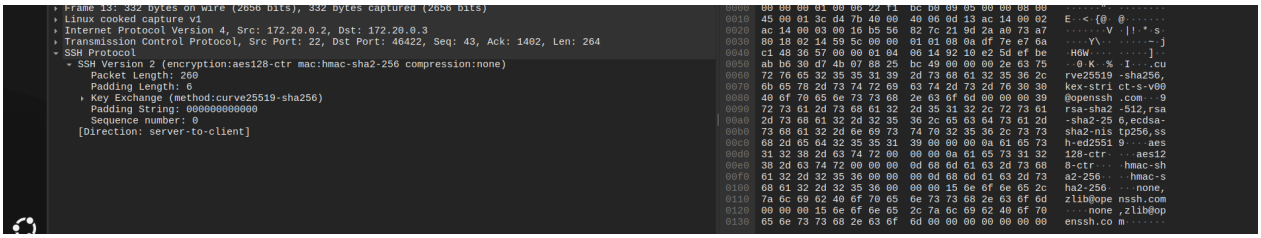


Figura 25: Resultados Obtenidos.

En la imagen se observa que el paquete SSH tiene un tamaño de 260 bytes y un peso total del paquete de 332, lo cual nos indica que se hizo la reduccion de tamaño de manera correcta

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es la herramienta estándar para acceder de forma segura a máquinas remotas. Su propósito es proteger la comunicación mediante criptografía, evitando ataques como *eavesdropping*, secuestro de sesiones y manipulación del tráfico. A partir del análisis de las capturas obtenidas, el flujo de conexión de OpenSSH (versión 2) se puede resumir de la siguiente forma:

1. **Capa de Transporte (TCP):** La conexión se establece mediante el clásico *three-way handshake* (SYN, SYN-ACK, ACK) al puerto 22. TCP garantiza la entrega fiable y el control de flujo.

2. **Intercambio de Identificación:** Cliente y servidor envían en texto plano sus versiones del protocolo (por ejemplo, `SSH-2.0-OpenSSH_7.2`). Esta información inicial aún no está cifrada.

3. **Negociación de Algoritmos (Key Exchange Init):** Ambas partes envían listas de algoritmos soportados para KEX (intercambio de claves), cifrado, MAC y compresión. Esta etapa continúa en texto plano, permitiendo el *fingerprinting* (HASSH).

4. **Intercambio de Claves (Diffie-Hellman / ECDH):** Se selecciona el algoritmo KEX más seguro soportado por ambos. Cliente y servidor intercambian claves públicas efímeras y calculan un secreto compartido sin transmitirlo directamente.

5. **Activación de Cifrado (New Keys):** Una vez derivadas las claves simétricas, ambos envían el mensaje `New Keys`, tras lo cual toda la comunicación pasa a estar cifrada.

6. **Autenticación del Usuario:** Dentro del túnel cifrado, el servidor solicita credenciales (contraseña o clave pública), completando el establecimiento seguro de la sesión.

4.2. Capas de Seguridad en OpenSSH

A partir del tráfico analizado, se identifican los siguientes principios de seguridad presentes en SSH:

- **Confidencialidad:** Proporcionada mediante cifrado simétrico (por ejemplo, *chacha20-poly1305*) negociado después del intercambio Diffie-Hellman. Todo el tráfico posterior al mensaje `New Keys` es ilegible para terceros.
- **Integridad:** Garantizada mediante un código de autenticación de mensajes (MAC), como *hmac-sha1*. Cada paquete contiene un MAC que asegura que no ha sido alterado.
- **Autenticidad del Servidor:** Asegurada por la *HostKey*. El servidor firma parte de la negociación usando su clave privada y el cliente verifica esta firma con la clave pública almacenada en `known_hosts`. El mensaje `WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!` observado en el laboratorio confirma la efectividad de este mecanismo.
- **Autenticidad del Usuario:** Ocurre dentro del canal cifrado cuando el servidor solicita al cliente una contraseña o una firma generada por su clave privada.
- **Disponibilidad:** Proporcionada por el protocolo TCP, que garantiza retransmisiones, entrega ordenada y persistencia de la conexión. El servicio `sshd` mantiene el puerto 22 disponible.

4.3. Identificación de que protocolos no se cumplen

El principio de seguridad que no se cumple en SSH es el **No Repudio**. Aunque SSH garantiza autenticación fuerte durante la sesión, no incluye mecanismos criptográficos que permitan demostrar ante terceros que un usuario específico ejecutó una acción determinada. Esto se debe a que:

- Las claves de sesión son efímeras y se destruyen al finalizar la conexión.
- No existe firma digital persistente para cada comando enviado.
- Un atacante con acceso posterior a la clave privada del servidor podría reconstruir un ataque de tipo *Man-in-the-Middle*.

SSH está diseñado para proporcionar seguridad en tiempo real (confidencialidad, integridad y autenticación), pero no para generar evidencia verificable a largo plazo.

Conclusiones y comentarios

Este laboratorio permitió comprender a fondo el funcionamiento interno del protocolo SSH y observar en la práctica cómo se desarrolla cada fase. Entre los aspectos más relevantes se destacan:

- Las capturas de Wireshark mostraron que la fase inicial (versiones y negociación de algoritmos) ocurre en texto plano, permitiendo realizar *fingerprinting* (HASSH) sin romper el cifrado.
- El uso de Docker permitió simular múltiples clientes con distintas versiones, demostrando cómo pequeñas variaciones en la configuración generan diferencias en la huella digital del cliente.
- La modificación de binarios con `hexedit` y cambios en `sshd_config` mostró la flexibilidad del protocolo, pero también los riesgos como:
    - Degradación de la seguridad al forzar algoritmos antiguos.

En conjunto, el laboratorio confirmó que, aunque SSH es robusto en confidencialidad, integridad y autenticación, permite identificar metadatos en etapas iniciales y no garantiza el principio de No Repudio.