

Informe Laboratorio 5

Sección 4

José Martín Berrios Piña
e-mail: jose.berrios1@mail_udp.cl

Diciembre de 2024

Índice

Descripción de actividades	3
1. Desarrollo (Parte 1)	5
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 paquetes del flujo y el HASSH respectivo (detallado)	13
1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	16
1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	19
1.7. Compara la versión de HASSH obtenida con la base de datos para validar si el cliente corresponde al mismo	22
1.8. Tipo de información contenida en cada uno de los paquetes generados en texto plano	22
1.8.1. C1	22
1.8.2. C2	23
1.8.3. C3	23
1.8.4. C4/S1	23
1.9. Diferencia entre C1 y C2	23
1.10. Diferencia entre C2 y C3	24
1.11. Diferencia entre C3 y C4	24
2. Desarrollo (Parte 2)	26
2.1. Identificación del cliente SSH con versión “?”	26
2.2. Replicación de tráfico al servidor (paso por paso)	27
3. Desarrollo (Parte 3)	28
3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso)	28
4. Desarrollo (Parte 4)	30
4.1. Explicación OpenSSH en general	30
4.2. Capas de Seguridad en OpenSSH	31
4.3. Identificación de que protocolos no se cumplen	31

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.

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-2.0-0)
TCP	66 22 → 42398 [ACK] Seq=1 Ack=
SSHv2	107 Server: Protocol (SSH-2.0-0)
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

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

1.1.1. C1

```

1 # Dockerfile para C1 (Ubuntu 16.10)
2 FROM ubuntu:16.10
3
4 # Cambiar a repositorio antiguo y eliminar referencias a security.
5   ubuntu.com
RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.
  ubuntu.com/ubuntu|g' /etc/apt/sources.list && \

```

```

6      sed -i '/security.ubuntu.com/d' /etc/apt/sources.list && \
7      apt-get update && \
8      apt-get install -y openssh-client && \
9      apt-get clean
10
11 # Copiar script de configuraci n
12 COPY setup.sh /setup.sh
13 RUN chmod +x /setup.sh
14
15 # Ejecutar el script durante la construcci n
16 RUN /setup.sh
17
18 # En caso de querer que se pare CMD ['/bin/bash']
19 CMD ['tail', '-f', '/dev/null']

```

Bloque 1: Dockerfile para C1 (Ubuntu 16.10)

1.1.2. C2

```

1 # Dockerfile para C2 (Ubuntu 18.10)
2 FROM ubuntu:18.10
3
4 # Cambiar a repositorio antiguo y eliminar referencias a security.
5   ubuntu.com
6 RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.
7   ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
8     sed -i '/security.ubuntu.com/d' /etc/apt/sources.list && \
9     apt-get update && \
10    apt-get install -y openssh-client && \
11    apt-get clean
12
13 # Copiar script de configuraci n
14 COPY setup.sh /setup.sh
15 RUN chmod +x /setup.sh
16
17 # Ejecutar el script durante la construcci n
18 RUN /setup.sh
19
20 # En caso de querer que se pare CMD ['/bin/bash']
21 CMD ['tail', '-f', '/dev/null']

```

Bloque 2: Dockerfile para C2 (Ubuntu 18.10)

1.1.3. C3

```

1 # Dockerfile para C3 (Ubuntu 20.10)
2 FROM ubuntu:20.10
3
4 # Cambiar a repositorio antiguo y eliminar referencias a security.
5   ubuntu.com
6 RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.
7     ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
8     sed -i '/security.ubuntu.com/d' /etc/apt/sources.list && \
9     apt-get update && \
10    apt-get install -y openssh-client && \
11    apt-get clean
12
13 # Copiar script de configuraci n
14 COPY setup.sh /setup.sh
15 RUN chmod +x /setup.sh
16
17 # Ejecutar el script durante la construcci n
18 RUN /setup.sh
19
20 # En caso de querer que se pare CMD ['/bin/bash']
21 CMD ['tail', '-f', '/dev/null']

```

Bloque 3: Dockerfile para C3 (Ubuntu 20.10)

1.1.4. C4/S1

```

1 # Dockerfile para C4 (Ubuntu 22.10)
2 FROM ubuntu:22.10
3
4 # Cambiar repositorio y actualizar
5 RUN sed -i 's|http://archive.ubuntu.com/ubuntu|http://old-releases.
6   ubuntu.com/ubuntu|g' /etc/apt/sources.list && \
7     sed -i '/security.ubuntu.com/d' /etc/apt/sources.list && \
8     apt-get update && \
9     apt-get install -y openssh-client openssh-server && \
10    apt-get clean
11
12 # Copiar y ejecutar el script de configuraci n
13 COPY setup.sh /setup.sh
14 RUN chmod +x /setup.sh
15 RUN /setup.sh
16
17 # Comando para mantener el contenedor activo y el servicio SSH
18   corriendo
19 CMD ['/usr/sbin/sshd', '-D']

```

Bloque 4: Dockerfile para C4 (Ubuntu 22.10)

1.2. Creación de las credenciales para S1

```

1 #!/bin/bash
2
3 # Crear el usuario 'prueba' con contraseña 'prueba'
4 useradd -m prueba
5 echo 'prueba:prueba' | chpasswd
6
7 # Configurar el servidor SSH
8 mkdir -p /var/run/sshd
9 echo 'PermitRootLogin yes' >> /etc/ssh/sshd_config
10 echo 'PasswordAuthentication yes' >> /etc/ssh/sshd_config
11
12 # Iniciar el servicio SSH
13 service ssh start
14
15 # Mensaje de finalización
16 echo 'Configuración completada en C4/S1 (Ubuntu 22.10) con servidor
      SSH listo'
```

Bloque 5: Archivo setup.sh encargado de creación de las credenciales para el contenedor que tiene S1 (Ubuntu 22.10)

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

Una vez creados los contenedores Docker, se procede a su conexión ejecutando el siguiente comando:

```
1 docker-compose up --build
```

Para obtener la dirección IP del servidor al que deseamos conectarnos, utilizamos el comando:

```
1 docker network inspect lab5_cripto_lab5_net
```

- En este caso, la dirección IP del servidor S1 obtenida es: **172.19.0.44**.

Posteriormente, se ingresa al contenedor que funciona como cliente 1 (Ubuntu 16.10) mediante los siguientes pasos:

1. Ingresar al contenedor utilizando:

```
1 docker exec -it C1 bash
```

2. Establecer conexión SSH con el servidor S1:

```
1 ssh prueba@172.19.0.5
```

Al establecer la conexión, la terminal nos solicita confirmar si confiamos en el servidor S1 y en la clave pública que este ha enviado. En este caso, y en los posteriores, responderemos 'yes', lo que permite que el fingerprint del servidor (clave pública) se almacene en el archivo ~/.ssh/known_hosts. Esto asegura que, en conexiones futuras, SSH pueda validar automáticamente la autenticidad del servidor.

Antes de continuar con la iniciacion de las credenciales y realización del handshake debemos buscar la interfaz a capturar, por lo tanto se utiliza:

```
1 ifconfig
```

Ahora, se busca la interfaz de red que Docker configuro para esta red Bridge de contenedores, las cuales contienen las ip entre **172.19.0.0 - 172.19.255.255**.

Las ip configuradas para nuestros contenedores son las siguientes:

- 172.19.0.11
- 172.19.0.22
- 172.19.0.33
- 172.19.0.44

Por ello la interfaz que se utilizara para escuchar en Wireshark sera la **br-7c7cc58c231e**.

```
br-7c7cc58c231e: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 172.19.0.1 netmask 255.255.0.0 broadcast 172.19.255.255
      inet6 fe80::42:b4ff:fe7f:3d1f prefixlen 64 scopeid 0x20<link>
        ether 02:42:b4:7f:3d:1f txqueuelen 0 (Ethernet)
          RX packets 0 bytes 0 (0.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 29 bytes 4098 (4.0 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 3: Obtención de la interfaz con 'ifconfig'.

Para la autenticación, se ingresan las credenciales previamente configuradas:

- Usuario: **prueba**
- Contraseña: **prueba**

```
jb@jb-pc1:~/Desktop/lab5_cripto$ docker exec -it C1 bash
root@0245bd0973a2:/# ssh prueba@172.19.0.44
The authenticity of host '172.19.0.44 (172.19.0.44)' can't be established.
ECDSA key fingerprint is SHA256:I65vNUxPg8emCkdkTSh2E8oGksPYVZM7iODFPu21Xko.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.19.0.44' (ECDSA) to the list of known hosts.
prueba@172.19.0.44's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.8.0-49-generic x86_64)

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

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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ x
```

Figura 4: Ejemplo conexión de C1->S1.

Una vez establecida la conexión correctamente por primera vez, se genera una salida que confirma el éxito del procedimiento. A continuación, se muestra una imagen representativa del entorno conectado.

1.3 Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) 1 DESARROLLO (PARTE 1)

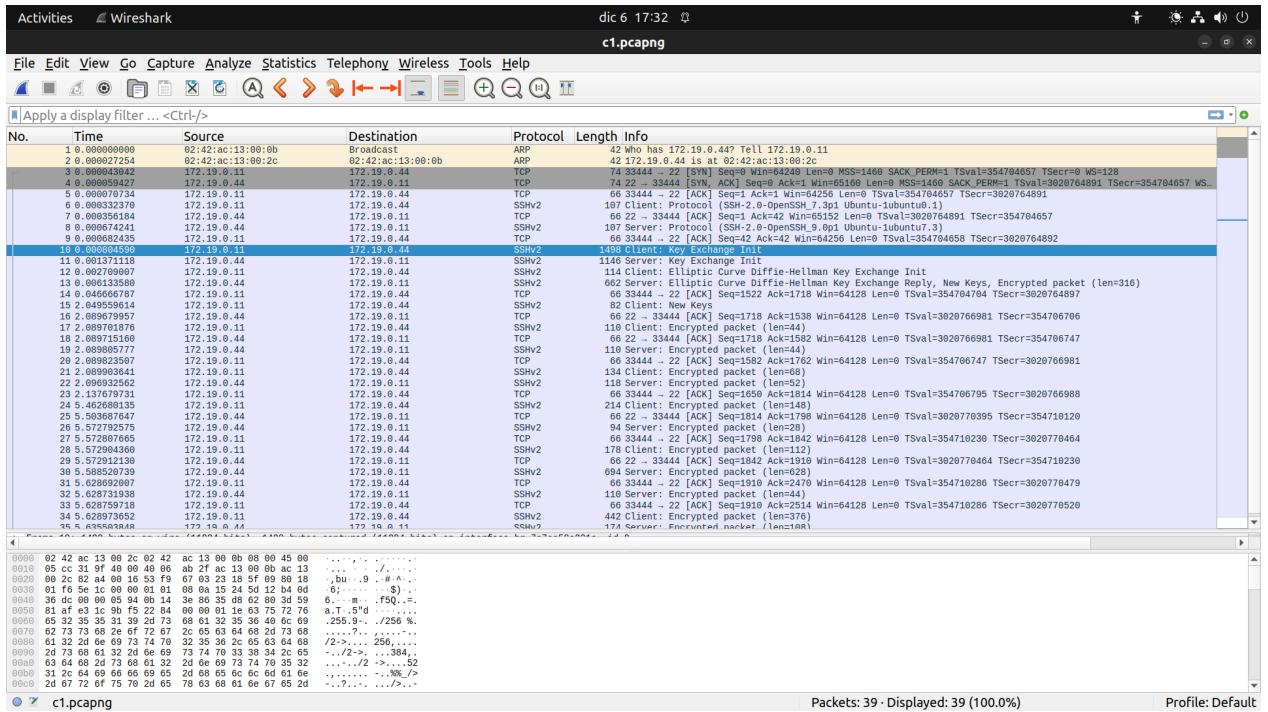


Figura 5: Captura del trafico generado C1->S1.

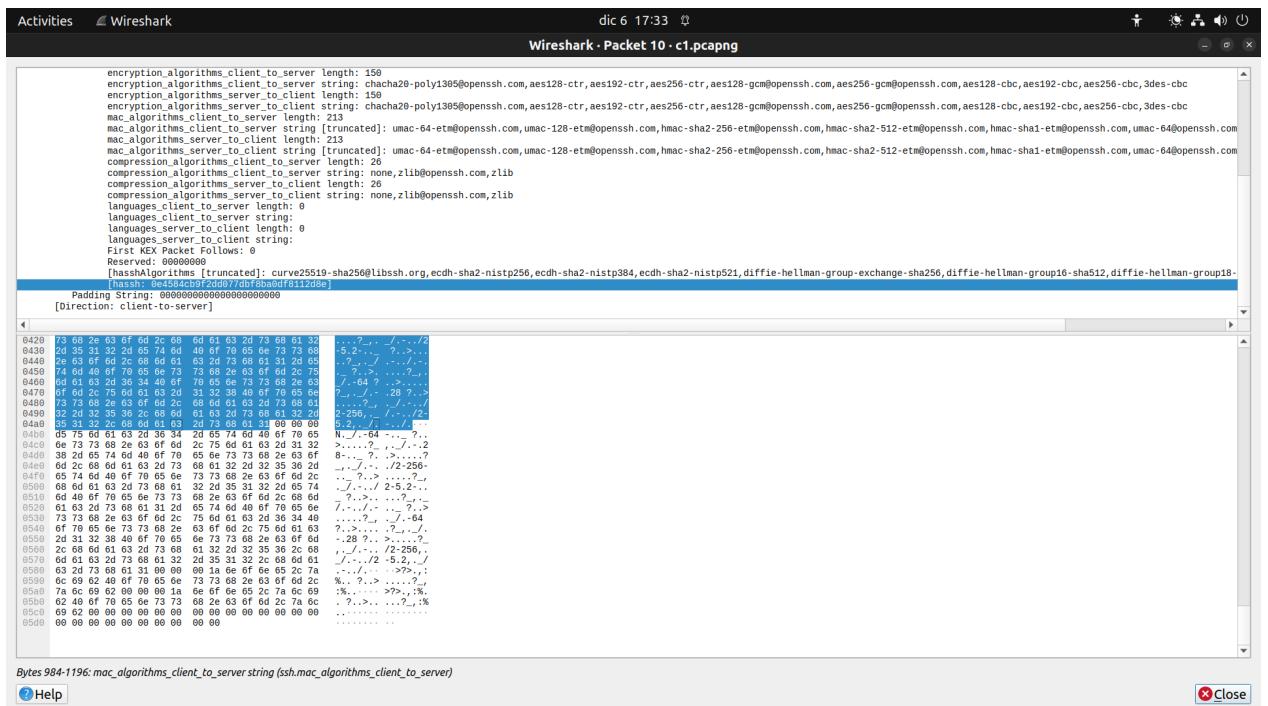


Figura 6: Obtención del HASSH de C1->S1.

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

1 DESARROLLO (PARTE 1)

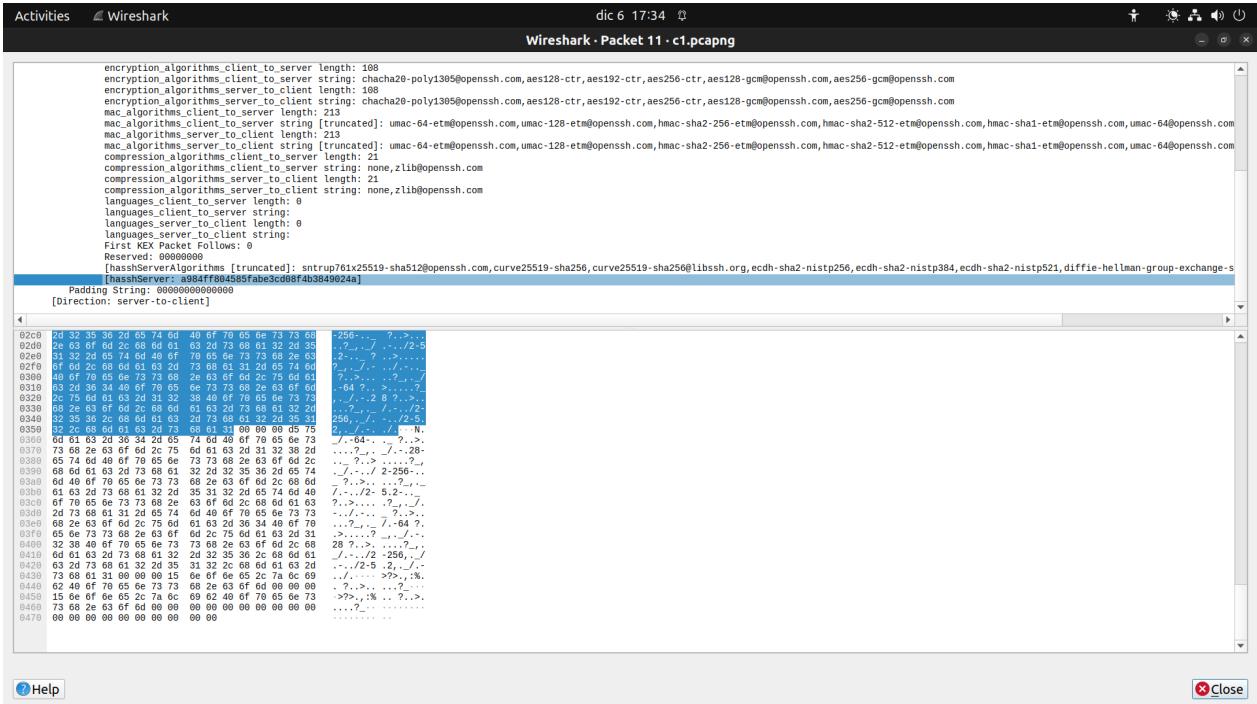


Figura 7: Obtención del HASSHSERVER de C1->S1.

Las últimas dos imágenes nos permiten obtener el hash y el hash del servidor (cada una aparece seleccionada con su respectivo fondo celeste). Los valores obtenidos son:

- [hassh: 0e4584cb9f2dd077dbf8ba0df8112d8e]
- [hasshServer: a984ff804585fabe3cd08f4b3849024a]

La explicación de cada paquete o conjunto de paquetes solo se llevara a cabo en la siguiente lista itemizadora, pues lo único que principalmente varia entre los casos de comunicación con los clientes C1, C2, C3 y C4 son los tamaños del paquete, no su funcionamiento.

Análisis:

- **Paquete 1: ARP Request/Reply:** Tamaño de 42 bytes. Estos paquetes resuelven la dirección MAC del servidor en la red local.
- **Paquete 2 al 4: TCP (Handshake Inicial):** Tamaño promedio de 74 bytes. Estos paquetes establecen la conexión TCP entre el cliente y el servidor (incluyen SYN y ACK).
- **Paquete 5: Client Key Exchange Init:** Tamaño de 1496 bytes. Parámetros de cifrado iniciales relacionados al intercambio de claves del cliente.

- **Paquete 6: Server Key Exchange Reply:** Tamaño de 1146 bytes. Incluye las claves públicas y otros datos del servidor para el intercambio de claves.
- **Paquete 7 al 9: Acknowledgment (ACK):** Tamaño promedio de 66 bytes. Confirman la recepción de los datos intercambiados entre cliente y servidor.
- **Paquete 10 en adelante: Datos Cifrados:** Tamaño que varia entre 52 y 174 bytes. Transmiten información cifrada entre el cliente y el servidor, como autenticación y comandos.

El único procedimiento realizado en la captura fue el establecimiento de la conexión, por ello los paquetes cifrados a partir del paquete 10 incluyen la verificación de autenticación y datos básicos para asegurar la continuidad de la conexión segura.

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

```
jb@jb-pc1:~/Desktop/lab5_cripto$ docker exec -it C2 bash
root@cbc3f35841e3:/# ssh prueba@172.19.0.44
The authenticity of host '172.19.0.44 (172.19.0.44)' can't be established.
ECDSA key fingerprint is SHA256:I65vNUxPg8emCkdkTSh2E8oGksPYVZM7iODFPu21Xko.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.19.0.44' (ECDSA) to the list of known hosts.
prueba@172.19.0.44's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.8.0-49-generic x86_64)

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

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: Fri Dec  6 19:24:58 2024 from 172.19.0.11
$ 
```

Figura 8: Ejemplo conexión de C2->S1.

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

1 DESARROLLO (PARTE 1)

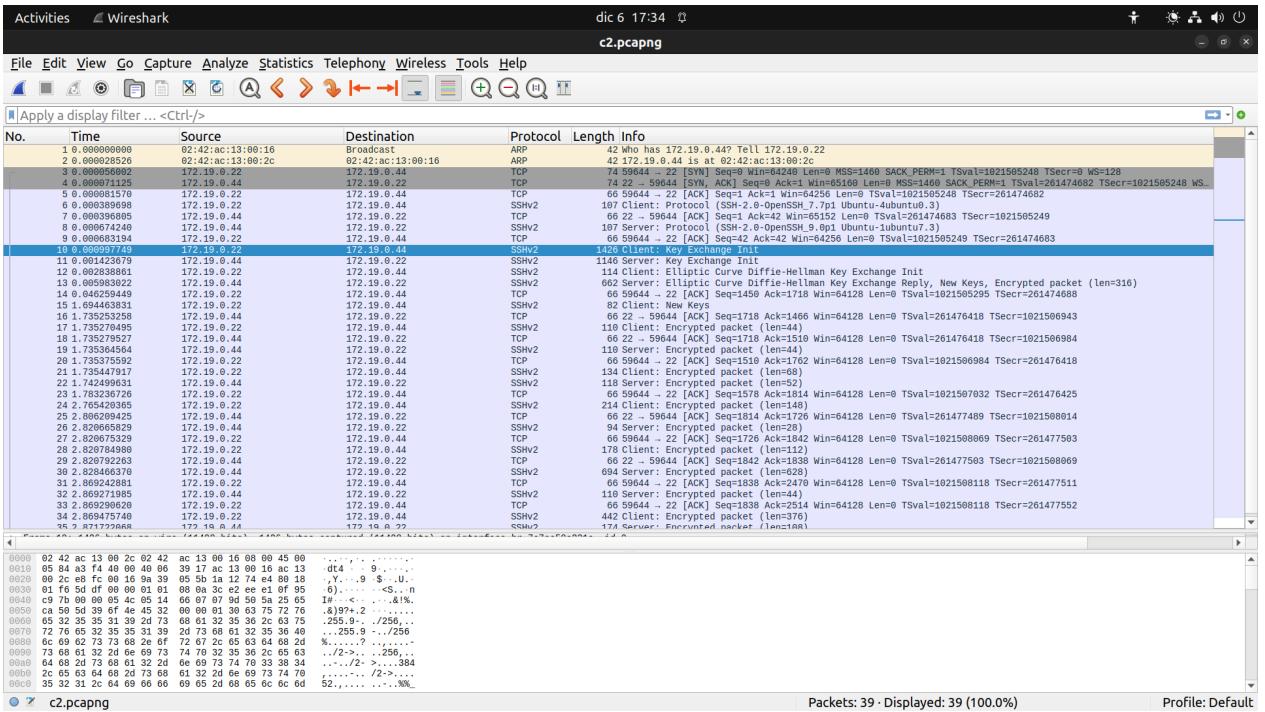


Figura 9: Captura del tráfico generado C2->S1.

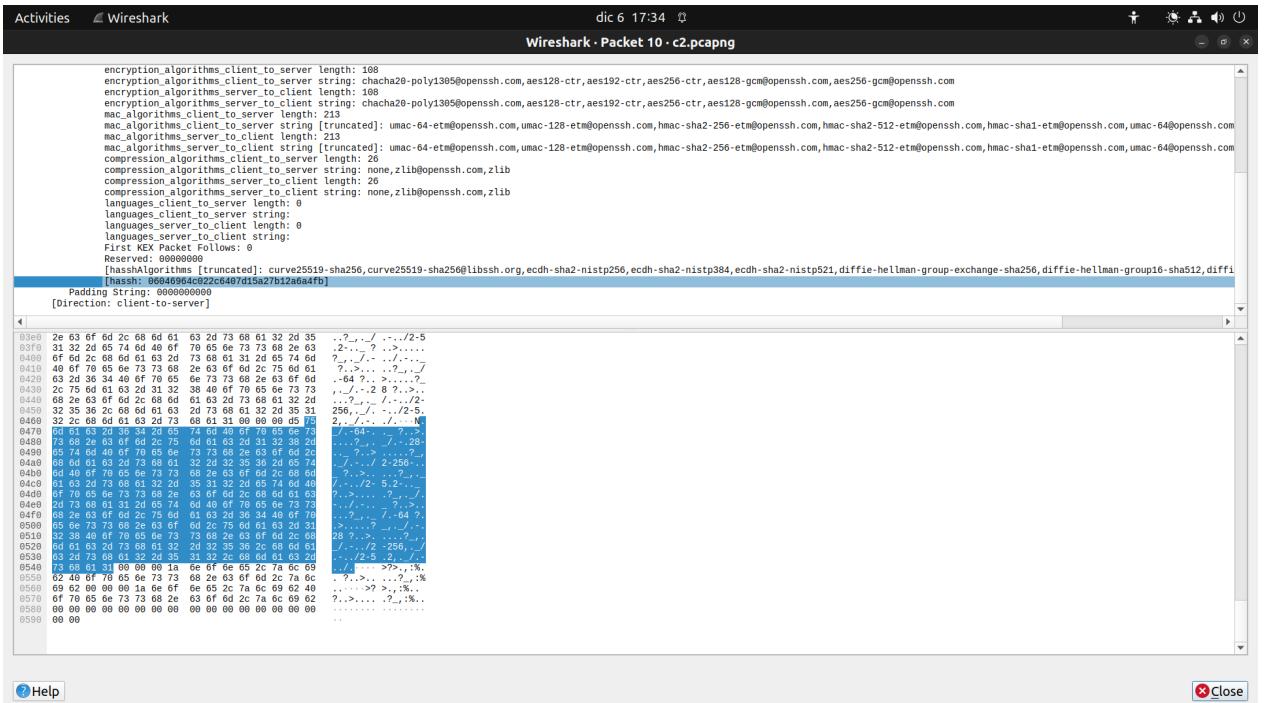


Figura 10: Obtención del HASSH de C2->S1.

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

1 DESARROLLO (PARTE 1)

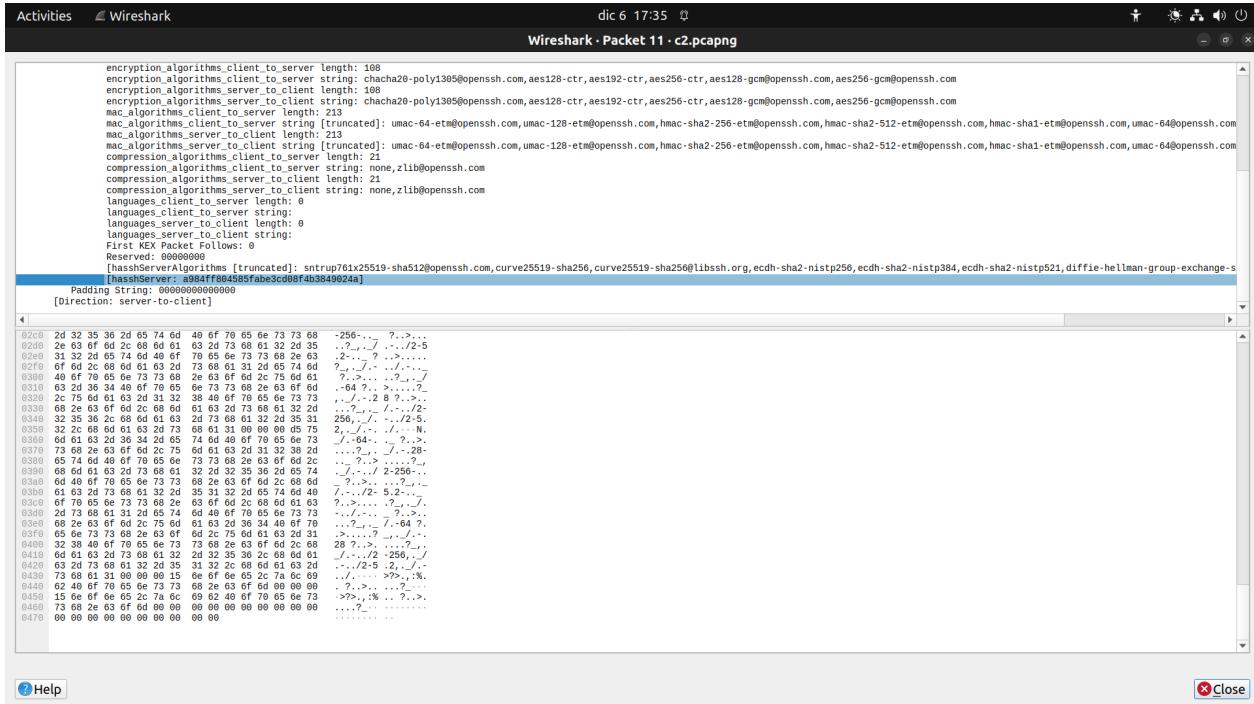


Figura 11: Obtención del HASSHSERVER de C2->S1.

Las últimas dos imágenes nos permiten obtener el hash y el hash del servidor (cada una aparece seleccionada con su respectivo fondo celeste). Los valores obtenidos son:

- [hassh: 06046964c022c6407d15a27b12a6a4fb]
- [hasshServer: a984ff804585fabe3cd08f4b3849024a]

A la hora de analizar el orden de aparición y tamaños se obtiene:

- **Paquete 1: ARP Request/Reply:** Tamaño de 42 bytes.
- **Paquete 2 al 4: TCP (Handshake Inicial):** Tamaño promedio de 74 bytes.
- **Paquete 5: Client Key Exchange Init:** Tamaño de 1426 bytes.
- **Paquete 6: Server Key Exchange Reply:** Tamaño de 1146 bytes.
- **Paquete 7 al 9: Acknowledgment (ACK):** Tamaño promedio de 66 bytes.
- **Paquete 10 en adelante: Datos Cifrados:** Tamaño variable entre 52 y 174 bytes.

El único procedimiento realizado en la captura fue el establecimiento de la conexión, por ello los paquetes cifrados a partir del paquete 10 incluyen la verificación de autenticación y datos básicos para asegurar la continuidad de la conexión segura.

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

```
jb@jb-pc1:~/Desktop/lab5_cripto$ docker exec -it C3 bash
root@ccccc1f2ee517:/# ssh prueba@172.19.0.44
The authenticity of host '172.19.0.44 (172.19.0.44)' can't be established.
ECDSA key fingerprint is SHA256:I65vNUxPg8emCkdkTSh2E8oGksPYVZM7iODFPu21Xko.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.19.0.44' (ECDSA) to the list of known hosts.
prueba@172.19.0.44's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.8.0-49-generic x86_64)

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

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: Fri Dec  6 19:25:38 2024 from 172.19.0.22
$ 
```

Figura 12: Ejemplo conexión de C3->S1.

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

1 DESARROLLO (PARTE 1)

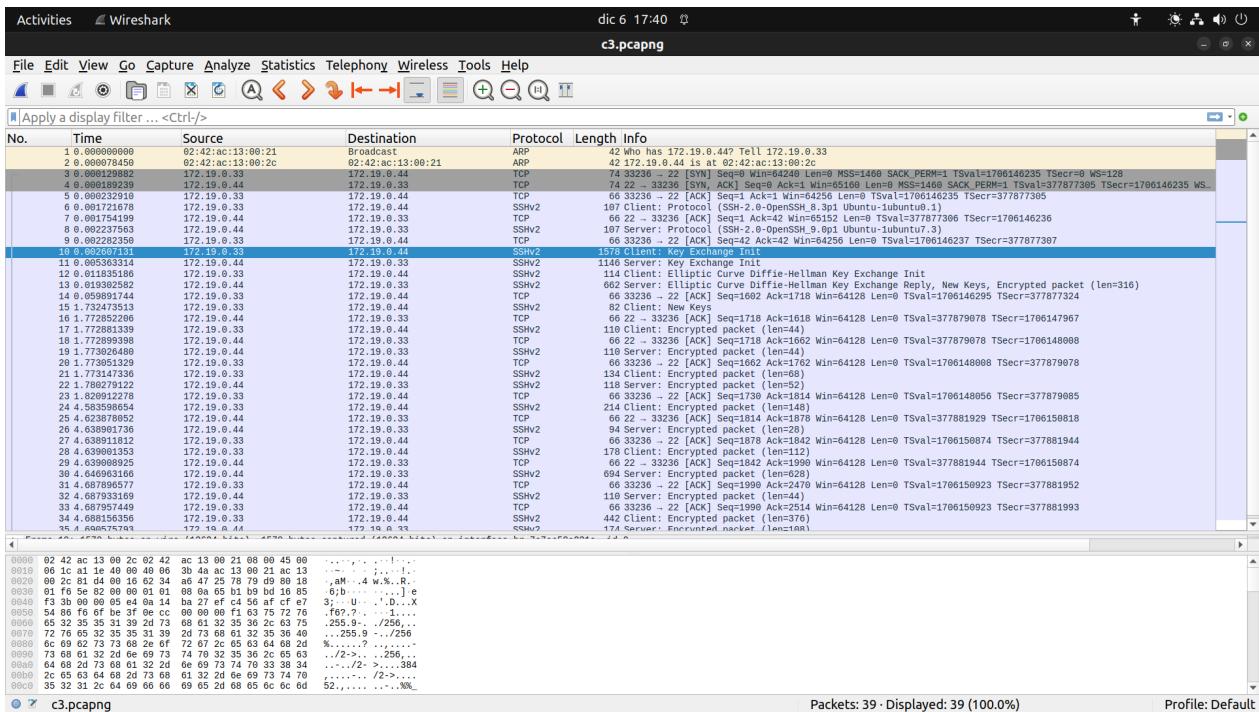


Figura 13: Captura del tráfico generado C3->S1.

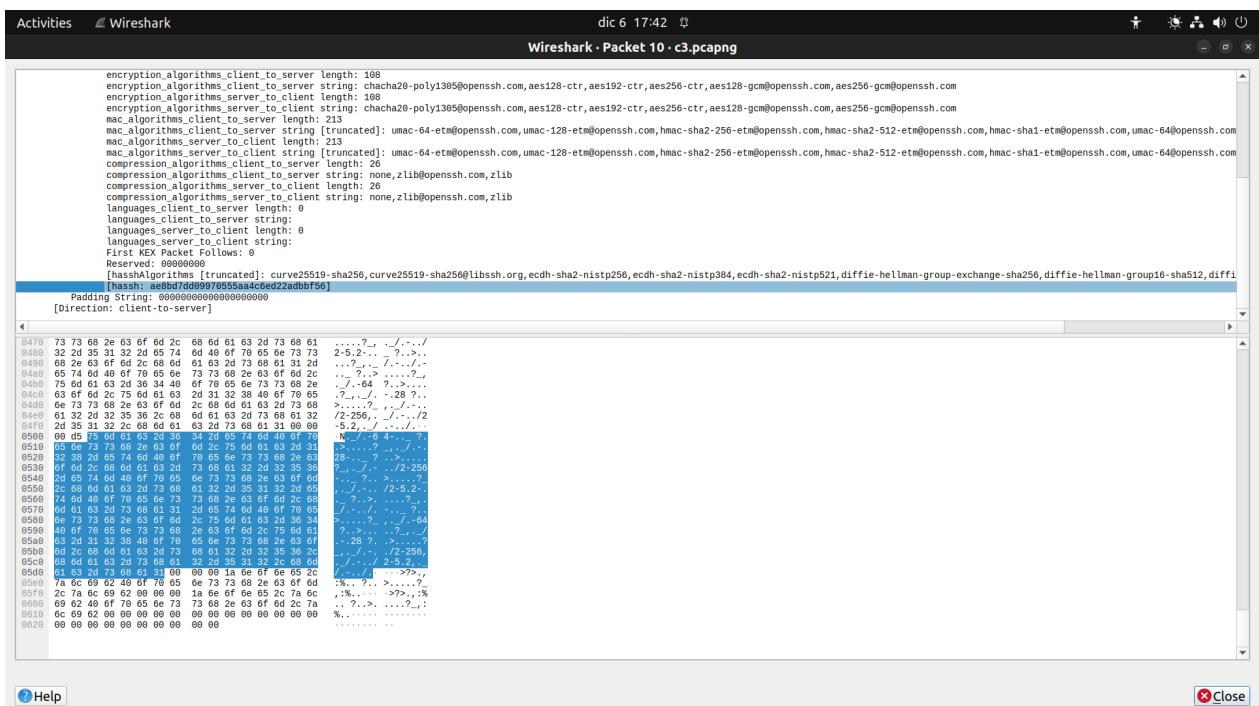


Figura 14: Obtención del HASSH de C3->S1.

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

1 DESARROLLO (PARTE 1)

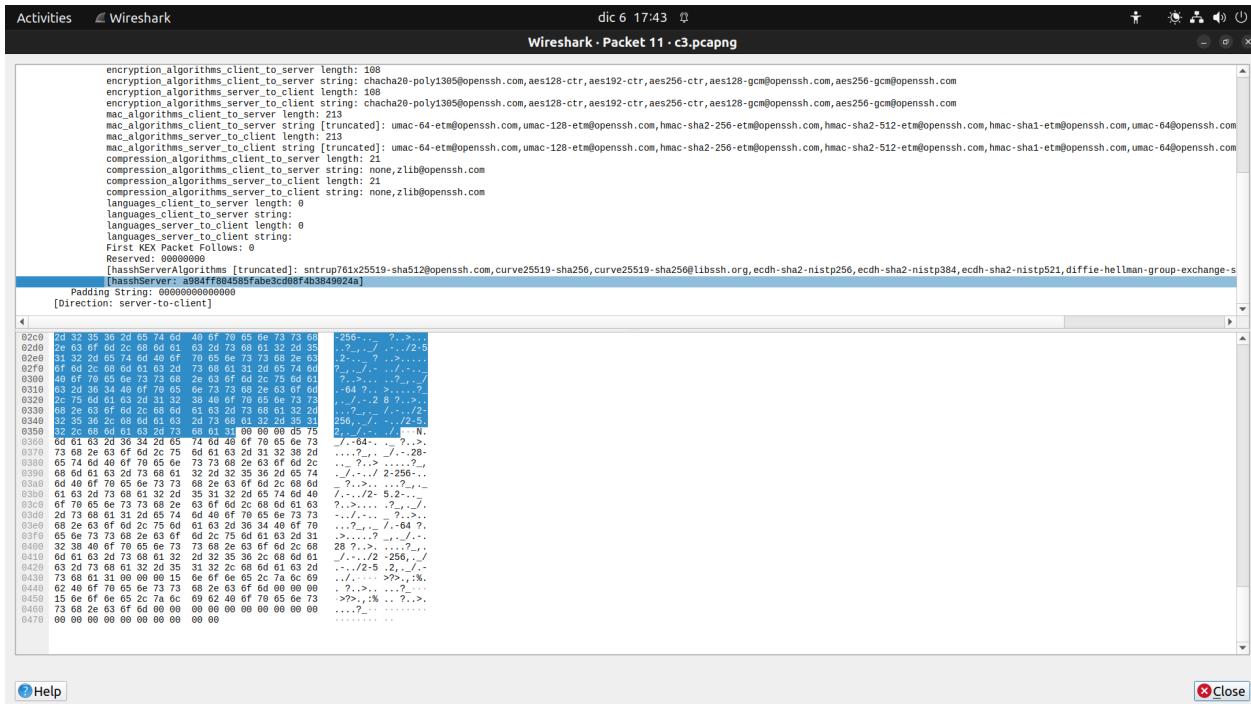


Figura 15: Obtención del HASSHSERVER de C3->S1.

Las últimas dos imágenes nos permiten obtener el hash y el hash del servidor (cada una aparece seleccionada con su respectivo fondo celeste). Los valores obtenidos son:

- [hassh: ae8bd7dd09970555aa4c6ed22adbbf56]
- [hasshServer: a984ff804585fabe3cd08f4b3849024a]

A la hora de analizar el orden de aparición y tamaños se obtiene:

- **Paquete 1: ARP Request/Reply:** Tamaño de 42 bytes.
- **Paquete 2 al 4: TCP (Handshake Inicial):** Tamaño promedio de 74 bytes.
- **Paquete 5: Client Key Exchange Init:** Tamaño de 1578 bytes.
- **Paquete 6: Server Key Exchange Reply:** Tamaño de 1146 bytes.
- **Paquete 7 al 9: Acknowledgment (ACK):** Tamaño promedio de 66 bytes.
- **Paquete 10 en adelante: Datos Cifrados:** Tamaño variable entre 52 y 174 bytes.

El único procedimiento realizado en la captura fue el establecimiento de la conexión, por ello los paquetes cifrados a partir del paquete 10 incluyen la verificación de autenticación y datos básicos para asegurar la continuidad de la conexión segura.

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

Ya que se comprende el funcionamiento de C4->S1, entonces debemos interceptar los paquetes desde el contenedor que funciona como C4, esto debido a que C4 y S1 están dentro del mismo contenedor, lo cual de forma predeterminada Wireshark no captura estos paquetes en la interfaz utilizada previamente(**br-7c7cc58c231e**).

```
jb@jb-pc1:~/Desktop/lab5_cripto$ docker exec -it C4 bash
root@8544280be76e:/# tcpdump -i lo -w /c4.pcap
tcpdump: listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C77 packets captured
154 packets received by filter
154 packets dropped by kernel
```

Figura 16: Metodo utilizado para la intercepción de los paquetes C4->S1.

Una vez comienza la intercepción en otra terminal hacemos la conexión de C4->S1.

```
jb@jb-pc1:~/Desktop/lab5_cripto$ docker exec -it C4 bash
root@8544280be76e:/# ssh prueba@172.19.0.44
The authenticity of host '172.19.0.44 (172.19.0.44)' can't be established.
ED25519 key fingerprint is SHA256:7Zx0dYgY5Z8tlEycBQuBfIbo7CXeUBbhVbR89CA/Imw.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.19.0.44' (ED25519) to the list of known hosts.
prueba@172.19.0.44's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.8.0-49-generic x86_64)

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

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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ [ ]
```

Figura 17: Ejemplo conexión de C4->S1.

1.6 Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) 1 DESARROLLO (PARTE 1)

El tráfico generado se analiza abriendo el archivo **c4.pcap** obtenido con Wireshark

```
jb@jb-pc1:~/Desktop/lab5_cripto$ docker cp C4:/c4.pcap /home/jb/Desktop/lab5_cripto/c4.pcap
Successfully copied 18.4kB to /home/jb/Desktop/lab5_cripto/c4.pcap
```

Figura 18: Reposicionamiento del archivo que contiene la captura de C4->S1.

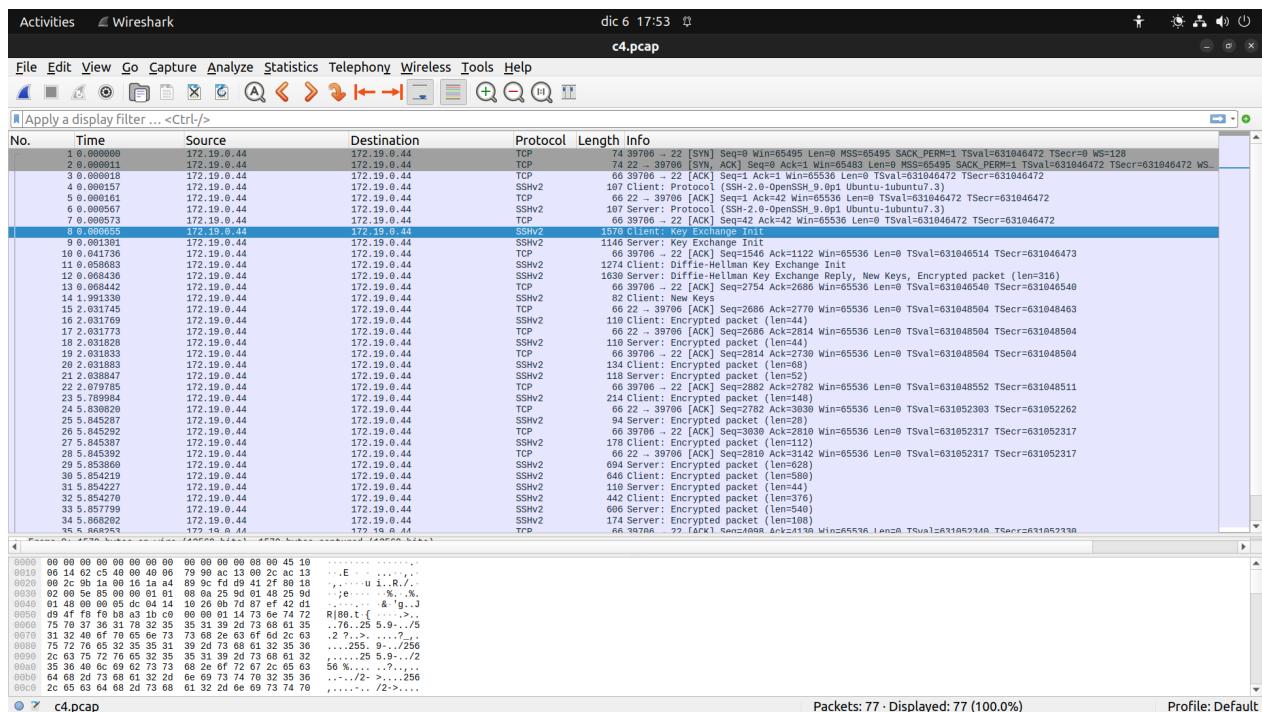


Figura 19: Captura del trafico generado C4->S1.

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

1 DESARROLLO (PARTE 1)

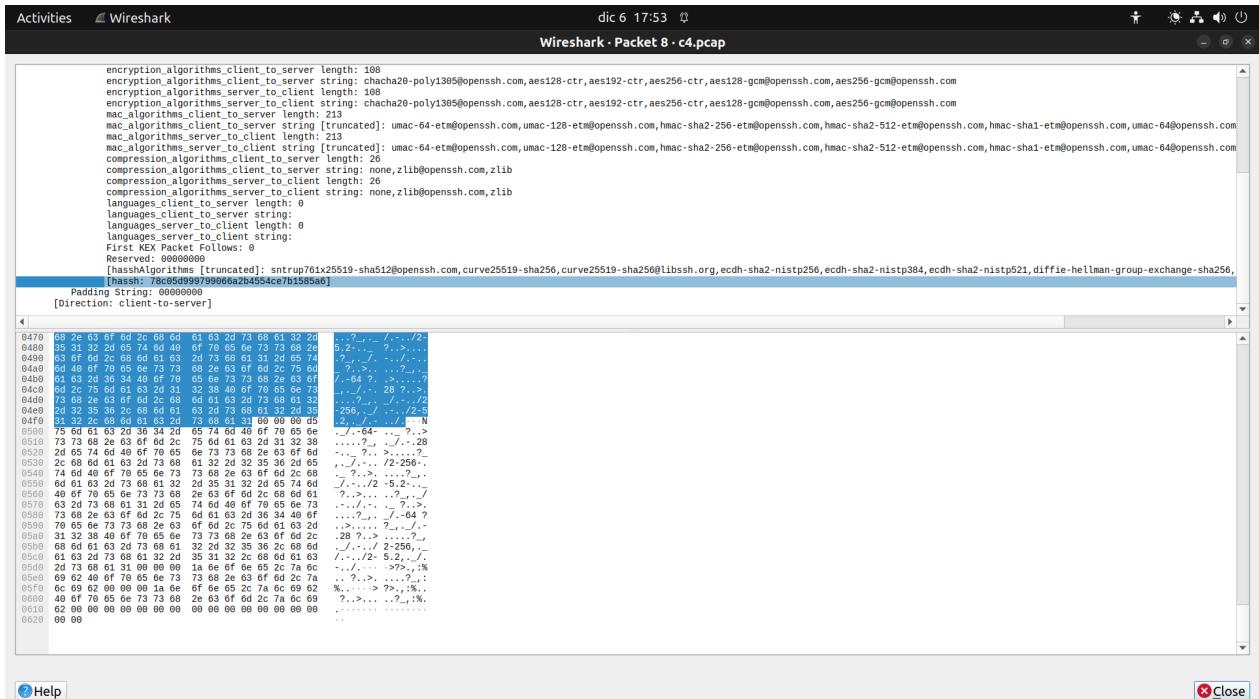


Figura 20: Obtención del HASSH de C4->S1.

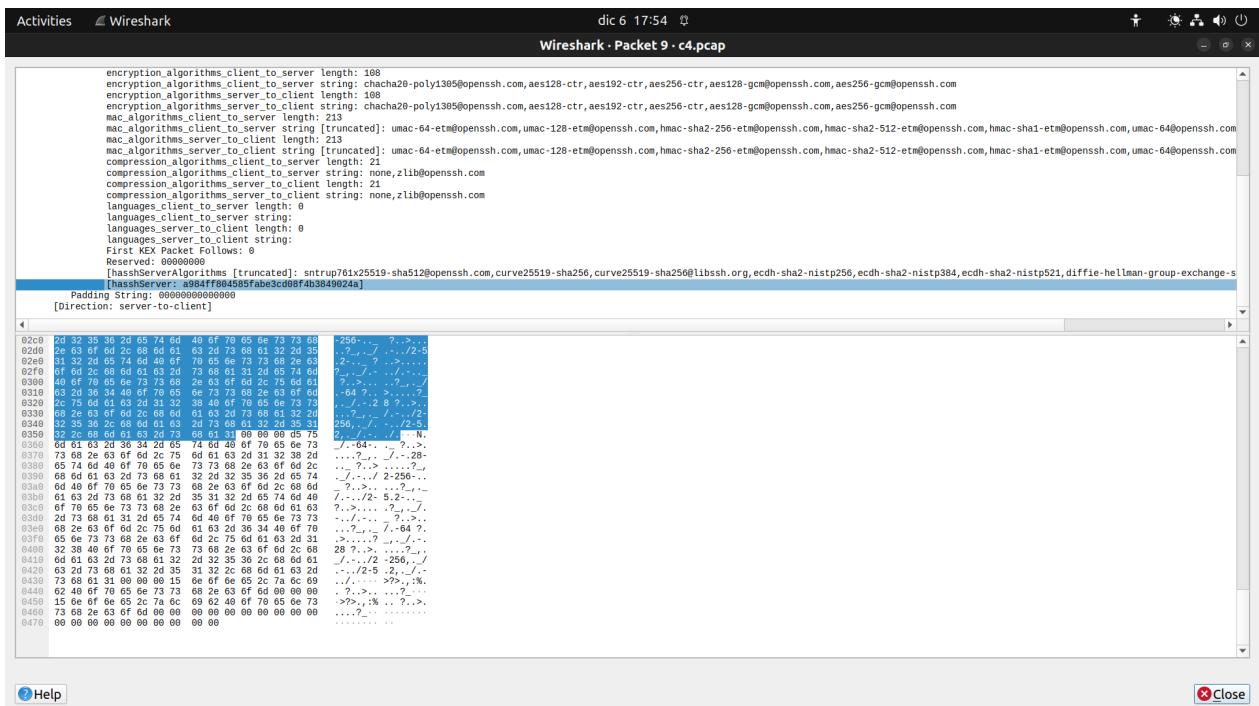


Figura 21: Obtención del HASSHSERVER de C4->S1.

Las últimas dos imágenes nos permiten obtener el hash y el hash del servidor (cada una aparece seleccionada con su respectivo fondo celeste). Los valores obtenidos son:

- [hassh: 78c05d999799066a2b4554ce7b1585a6]
- [hasshServer: a984ff804585fabe3cd08f4b3849024a]

A la hora de analizar el orden de aparición y tamaños se obtiene:

- **Paquete 1 al 3: TCP (Handshake Inicial):** Tamaño promedio de 74 bytes.
- **Paquete 4: Client Key Exchange Init:** Tamaño de 1570 bytes.
- **Paquete 5: Server Key Exchange Reply:** Tamaño de 1274 bytes.
- **Paquete 6 al 8: Acknowledgment (ACK):** Tamaño promedio de 66 bytes.
- **Paquete 9 en adelante: Datos Cifrados:** Tamaño variable entre 52 y 244 bytes.

El único procedimiento realizado en la captura fue el establecimiento de la conexión, por ello los paquetes cifrados a partir del paquete 9 incluyen la verificación de autenticación y datos básicos para asegurar la continuidad de la conexión segura.

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

NULL

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

1.8.1. C1

- **Handshake Inicial (TCP):** Igual que en cualquier conexión TCP. Se ven flags SYN, SYN-ACK y ACK. Por ser una versión más vieja del sistema, se tienen valores por defecto más simples en el tamaño de ventana y no tantas opciones avanzadas de TCP.
- **Key Exchange Init (SSH2):** Se aprecian listados de algoritmos tradicionales, incluyendo cifrados antiguos como CBC y 3DES.
- **Key Exchange Reply (SSH2):** El servidor devuelve su llave pública y datos para completar el intercambio de claves. Debido a la antigüedad, podría no haber soporte para llaves especiales (FIDO/U2F).

1.8.2. C2

- **Handshake Inicial (TCP):** Muy similar a C1, pero dada la versión más reciente, el stack TCP incluye opciones más refinadas por defecto, como mayor ventana inicial.
- **Key Exchange Init (SSH2):** Lista de algoritmos más moderna, eliminando cifrados obsoletos. Aquí aparecen algoritmos como `curve25519-sha256`, eliminando los viejos CBC.
- **Key Exchange Reply (SSH2):** El servidor retorna su clave pública y parámetros DH.

1.8.3. C3

- **Handshake Inicial (TCP):** Igual que C2, con variaciones mínimas.
- **Key Exchange Init (SSH2):** Se incluyen algoritmos aún más actualizados y soporte para llaves especiales (FIDO/U2F).
- **Key Exchange Reply (SSH2):** El servidor responde con sus claves, ahora incluyendo tipos de llaves más modernas. Esto se aprecia en el contenido del paquete y la selección de KEX y host keys.

1.8.4. C4/S1

- **Handshake Inicial:** Igual a todos los anteriores.
- **Key Exchange Init:** Parámetros iniciales optimizados para la versión más reciente del cliente SSH.
- **Key Exchange Reply:** Claves públicas del servidor y datos específicos del algoritmo de intercambio de claves seleccionado.

1.9. Diferencia entre C1 y C2

- **Tamaños de los paquetes:** En C1, el paquete `Client Key Exchange Init` tiene un tamaño de 1496 bytes, mientras que en C2 este paquete es más pequeño, con un tamaño de 1426 bytes.
- **HASSH del cliente:** Los valores de HASSH muestran configuraciones distintas para la negociación de claves iniciales:
 - C1: `0e4584cb9f2dd077dbf8ba0df8112d8e`
 - C2: `06046964c022c6407d15a27b12a6a4fb`
- **Sistema operativo y versión:** C1 utiliza Ubuntu 16.10, mientras que C2 emplea Ubuntu 18.10.

A continuación se seleccionan los principales de cada área, los cuales también serán utilizados para análisis posteriores de los demás clientes.

C1 (Ubuntu 16.10):

- KEX: curve25519-sha256@libssh.org, ecdh-sha2-nistp, diffie-hellman-
- Encryption: incluye modos ctr, gcm@openssh.com, y también cbc y 3des-cbc

C2 (Ubuntu 18.10) vs C1:

- KEX: se agrega curve25519-sha256
- Encryption: se eliminan las variantes cbc y 3des-cbc

1.10. Diferencia entre C2 y C3

- **Tamaños de los paquetes:** En C2, el paquete Client Key Exchange Init tiene un tamaño de 1426 bytes, mientras que en C3 este paquete es más grande, con un tamaño de 1578 bytes. Esto indica un aumento en los datos enviados durante la negociación inicial.
- **HASSH del cliente:** Los valores de HASSH también muestran diferencias claras:
 - C2: 06046964c022c6407d15a27b12a6a4fb
 - C3: 157d7cc6f4620d081a7e32a5b12eabf5
- **Sistema operativo y versión:** C2 utiliza Ubuntu 18.10, mientras que C3 emplea Ubuntu 20.04.

C3 (Ubuntu 20.10) vs C2:

- Host Keys: se agregan llaves sk-ecdsa-, sk-ssh-ed25519-

1.11. Diferencia entre C3 y C4

- **Tamaños de los paquetes:** En C3, el paquete Client Key Exchange Init tiene un tamaño de 1578 bytes, mientras que en C4 este paquete es ligeramente más pequeño, con un tamaño de 1570 bytes. Por ultimo pero importante notar, los paquetes cifrados en C4 presentan un rango más amplio, variando entre 52 y 244 bytes, en comparación con C3.
- **HASSH del cliente:** Los valores de HASSH indican configuraciones diferentes en los clientes SSH:
 - C3: 157d7cc6f4620d081a7e32a5b12eabf5

- C4: 29ac456b87f44bda35e1d6b9a6bfc8c1
- **Sistema operativo y versión:** C3 utiliza Ubuntu 20.04, mientras que C4 emplea Ubuntu 22.04.

C4 (Ubuntu 22.10) vs C3:

- KEX: se agrega sntrup761x25519-sha512@openssh.com

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión “?”

El cliente identificado mas probable en este caso es C3, es decir el que utiliza la versión Ubuntu 20.10, debido a que tanto la imagen que se dio como ejemplo al compararse con la obtenida anteriormente poseen un Key Exchange Init del cliente de tamaño 1578 bytes.

Se realiza modificación al **setup.sh** del contenedor que corresponde al cliente C3, de forma que salga **SSH2.0OpenSSH_?**

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 22: Tráfico generado del informante(EJEMPLO)

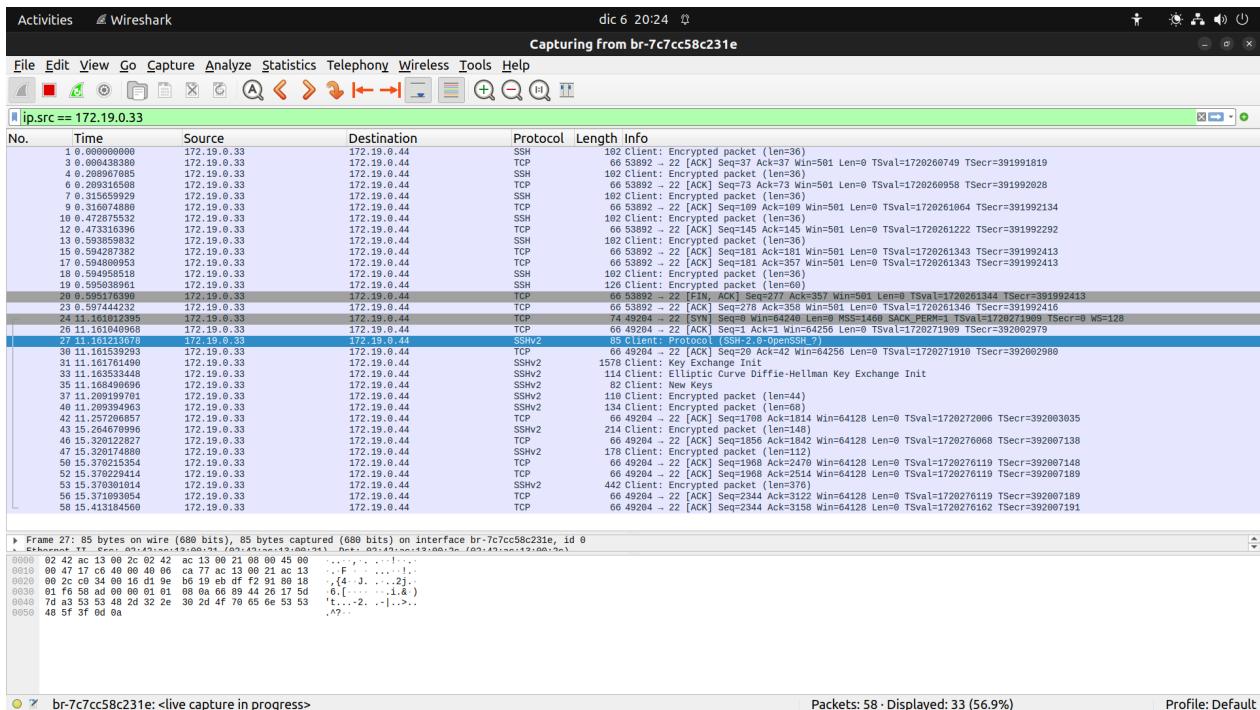


Figura 23: Captura del tráfico generado C3->S1 con modificación de la versión SSH.

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

Agregamos un **setup.sh** del contenedor que corresponde al cliente C3, de forma que salga **SSH2.0OpenSSH_?**

```
1 #!/bin/bash
2 set -e
3
4
5 # Instalar dependencias de compilaci n
6 apt-get update
7 apt-get install -y wget build-essential zlib1g-dev libssl-dev
     libpam0g-dev libselinux1-dev libwrap0-dev ca-certificates
8
9 # Descargar el c digo fuente de OpenSSH 8.3p1
10 # URL alternativa: https://cdn.openbsd.org/pub/OpenBSD/OpenSSH/
     portable/openssh-8.3p1.tar.gz
11 wget https://cdn.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-
     -8.3p1.tar.gz
12 tar xvfz openssh-8.3p1.tar.gz
13 cd openssh-8.3p1
14
15 # Modificar la cadena de versi n en el c digo fuente
```

```

16 sed -i 's/^#define SSH_VERSION.*/#define SSH_VERSION "OpenSSH_?"/'
  version.h
17
18 # Configurar
19 ./configure --prefix=/usr --sysconfdir=/etc/ssh
20 make
21 make install
22
23 echo "Configuración completada en C3 (Ubuntu 20.10) con OpenSSH_?
  personalizado"

```

Bloque 6: Archivo setup.sh encargado de modificación de la versión para el contenedor del cliente C3.

Además debemos tratar de replicar el mismo flujo de tráfico en este caso de C3->S1, donde el método utilizado es exactamente el mismo de todas las otras secciones, la única diferencia es que debemos filtrar en este caso por la ip del cliente C3 de forma que solo visualicemos los paquetes del cliente(`ip.src == 172.19.0.33`) y no del servidor.

La única complejidad de este paso era identificar el cliente y forzar la versión de SSH a `SSH2.0OpenSSH_?`.

24 11.161012395	172.19.0.33	172.19.0.44	TCP	74 49284 - 22 [SYN] Seq=0 Win=44240 Len=0 MSS=1460 SACK_PERM=1 TStamp=1720271909 TSectr=0 V
26 11.161040968	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TStamp=1720271909 TSectr=392002979
27 11.16123678	172.19.0.33	172.19.0.44	SShv2	85 Client: Protocol (SSH-2.0-OpenSSH_?)
30 11.161539293	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=20 Ack=42 Win=64256 Len=0 TStamp=1720271910 TSectr=392002980
31 11.161761149	172.19.0.33	172.19.0.44	SShv2	1578 Client: Key Exchange Init
33 11.1617611446	172.19.0.33	172.19.0.44	SShv2	1602 Client: New Keys Curve Diffie-Hellman Key Exchange Init
35 11.168499696	172.19.0.33	172.19.0.44	SShv2	82 Client: New Keys
37 11.209139761	172.19.0.33	172.19.0.44	SShv2	110 Client: Encrypted packet (len=44)
40 11.209334963	172.19.0.33	172.19.0.44	SShv2	134 Client: Encrypted packet (len=68)
42 11.257206857	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=1708 Ack=1814 Win=64128 Len=0 TStamp=1720272006 TSectr=392003035
43 15.264670996	172.19.0.33	172.19.0.44	SShv2	214 Client: Encrypted packet (len=148)
46 15.320122827	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=1854 Ack=1842 Win=64128 Len=0 TStamp=1720276668 TSectr=392007138
47 15.320174888	172.19.0.33	172.19.0.44	SShv2	178 Client: Encrypted packet (len=112)
50 15.370215354	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=1969 Ack=2478 Win=64128 Len=0 TStamp=1720276119 TSectr=392007148
52 15.370229414	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=1968 Ack=2514 Win=64128 Len=0 TStamp=1720276119 TSectr=392007189
53 15.370301014	172.19.0.33	172.19.0.44	SShv2	442 Client: Encrypted packet (len=376)
56 15.371093054	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=2344 Ack=3122 Win=64128 Len=0 TStamp=1720276119 TSectr=392007189
58 15.413184560	172.19.0.33	172.19.0.44	TCP	66 49284 - 22 [ACK] Seq=2344 Ack=3158 Win=64128 Len=0 TStamp=1720276162 TSectr=392007191

Figura 24: Replicación al tráfico del ejemplo logrando `SSH2.0OpenSSH_?`.

3. Desarrollo (Parte 3)

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

Sabemos que el cliente dispone de los siguientes algoritmos:

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

```

▼ SSH Protocol
  ▼ SSH Version 2
    ▼ Packet Length: 1588
    ▼ Padding Length: 10
  ▼ Key Exchange
    ▼ Message Code: Key Exchange Init (20)
      ▼ Algo:
        ▼ Cookie: 77fdaa541e20773df7325b86ec0ef8b
          kex_algorithms length: 241
          kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,
          server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256-cert-v01@open
          encryption_algorithms_client_to_server length: 108
          encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com
          encryption_algorithms_server_to_client length: 108
          encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@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-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@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,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@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
          [hashalgos [truncated]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffi
          [hash]: ae8bd7dd09970555aa4c6ed22adbfb56]

```

Figura 25: Algoritmos del cliente.

Para esta parte necesitamos disminuir el tamaño de la Key Exchange Init del servidor, por ello se modifica el setup.sh obteniendo el siguiente:

```

1 #!/bin/bash
2 set -e
3
4 # Crear usuario y contraseña
5 useradd -m prueba
6 echo "prueba:prueba" | chpasswd
7
8 # Instalar SSH server y tcpdump
9 apt-get update && apt-get install -y openssh-server tcpdump
10
11 # Ajustar configuración SSH
12 mkdir -p /var/run/sshd
13 cp /etc/ssh/sshd_config /etc/ssh/sshd_config.bak
14
15 # Eliminar GSSAPIAuthentication del ssh_config (cliente) si existe
16 sed -i '/GSSAPIAuthentication/d' /etc/ssh/ssh_config || true
17
18 # Configuración mínima para KEI reducido
19 # Se elige un solo algoritmo compatible:
20 # - KEX: ecdh-sha2-nistp256 (el cliente lo soporta)
21 # - Cipher: aes128-ctr (breve y soportado)
22 # - MAC: hmac-sha1 (breve)
23 echo "KexAlgorithms ecdh-sha2-nistp256
24 Ciphers aes128-ctr
25 MACs hmac-sha1" >> /etc/ssh/sshd_config
26
27 # Reiniciar SSH
28 service ssh restart
29
30 echo "KEI reducido y sin GSSAPIAuthentication"

```

Bloque 7: Archivo setup.sh modificado para obtener menos de 300 bytes de KEI del servidor.

Una vez modificando el `setup.sh` y haciendo nuevamente `docker-compose up --build` y iniciando la captura haciendo el handshake inicial se obtiene la siguiente captura de Wireshark.

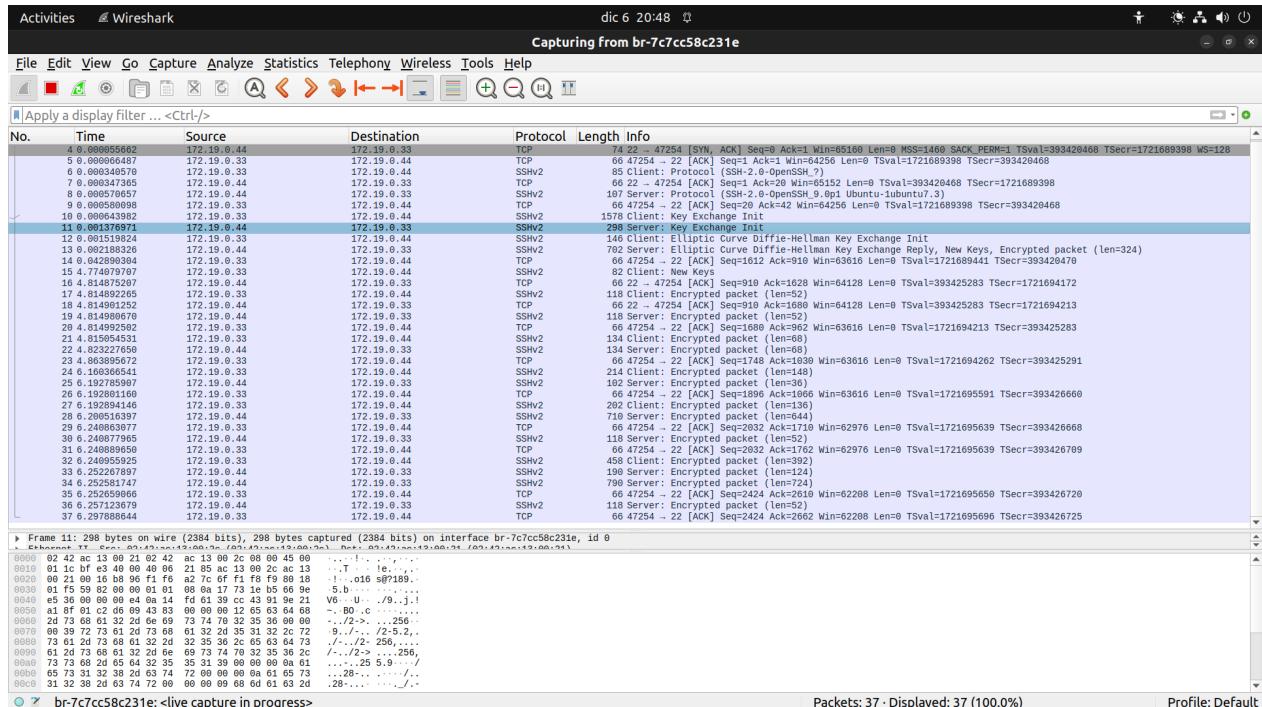


Figura 26: Captura con KEI del server de 289 bytes.

Como se pudo presenciar en la captura anterior, se obtiene un KEI del server con un tamaño de 298 bytes, lo cual es menor a 300 bytes.

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es un protocolo que permite conectar un computador con otro de forma segura a través de internet. Sirve para que dos computadoras puedan comunicarse usando cifrado, lo que significa que todo lo que viaja entre ellas está protegido y no puede ser leído por otros [1].

4.2. Capas de Seguridad en OpenSSH

- **Confidencialidad:** Usa cifrado fuerte para que nadie pueda leer los mensajes mientras viajan.
- **Integridad:** Revisa que los mensajes no sean modificados en el camino usando códigos especiales.
- **Autenticidad:** Verifica que las computadoras que se conectan son las correctas usando claves únicas.
- **No repudio:** No tan común en OpenSSH, pero existen algunos métodos avanzados para saber quién envió los mensajes sin dudas.
- **Disponibilidad:** Permite reconnectar automáticamente si algo interrumpe la comunicación.

4.3. Identificación de que protocolos no se cumplen

Al interceptar el tráfico SSH, se puede observar lo siguiente:

- La **confidencialidad** y **autenticidad** están bien protegidas gracias al cifrado y las claves públicas.
- La **integridad** está garantizada con algoritmos de verificación.
- La **disponibilidad** depende de la configuración de red, pero en general está bien manejada.
- El **no repudio** no siempre se cumple porque OpenSSH no guarda evidencia permanente de quién envió qué.

Conclusiones y comentarios

El laboratorio permitió profundizar en el análisis del protocolo OpenSSH y sus capas de seguridad, logrando identificar cómo se aplican los principios de seguridad de la información. Sin embargo, se observó que el no repudio no está garantizado en este protocolo, ya que no registra evidencia permanente del origen de los mensajes.

El trabajo con contenedores Docker facilitó el análisis en diferentes escenarios, demostrando que las versiones más recientes del cliente OpenSSH ofrecen mejores algoritmos de cifrado y mayor compatibilidad con estándares modernos.

Link github: https://github.com/martintintinnn/LAB5_Cripto_JB

Referencias

- [1] Inc. Red Hat. *Uso de comunicaciones seguras entre dos sistemas con OpenSSH*. Accedido: 6 de diciembre de 2024. 2024. URL: https://docs.redhat.com/es/documentation/red_hat_enterprise_linux/8/html/configuring_basic_system_settings/using-secure-communications-between-two-systems-with-openssh_configuring-basic-system-settings.