

Informe Laboratorio 5

Sección L02

Sebastian Gulfo Serna
sebastian.gulfo@mail_udp.cl

Junio de 2024

Índice

Descripción de actividades	3
1. Desarrollo (Parte 1)	5
1.1. Códigos de cada Dockerfile y su Construcción	5
1.1.1. C1 (Ubuntu 16.10)	5
1.1.2. C2 (Ubuntu 18.10)	6
1.1.3. C3 (Ubuntu 20.10)	6
1.1.4. C4/S1 (Ubuntu 22.10)	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)	9
1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	12
1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	15
1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	18
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	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)	26
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 y Principios de la Información	30
4.3. Identificación de qué principios no se cumplen (o se cumplen limitadamente)	31

Descripción de actividades

Para este último laboratorio, se solicita trabajar con Docker y el protocolo SSH, a fin de poder entender el concepto de criptografía asimétrica y firmas digitales. El propósito es adentrarse en cómo diferentes versiones de clientes SSH interactúan con un servidor, cómo pueden ser identificados a través de sus huellas de tráfico (HASSH) y cómo se puede modificar tanto el comportamiento del cliente como del servidor para cumplir con requerimientos específicos de comunicación segura.

Para lo anterior se 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 (servidor).
- Para cada uno de ellos, se deberá instalar el cliente OpenSSH disponible en los repositorios de apt, y para el equipo S1 se deberá también instalar el servidor OpenSSH.
- En S1 se deberá crear el usuario “**prueba**” con contraseña “**prueba**”, para acceder a él desde los clientes por el protocolo SSH.
- Se trabajarán 4 escenarios de conexión cliente-servidor: C1 → S1, C2 → S1, C3 → S1, C4 → S1 (conexión a localhost en la interfaz loopback).

Los pasos específicos a desarrollar son:

1. Para cada uno de los 4 escenarios, capturar y analizar el tráfico SSH. Obtener el HASSH para identificar cada cliente, comparar con bases de datos de referencia y detallar el tamaño y contenido de los paquetes relevantes del handshake. Se identificarán las diferencias incrementales entre las versiones de SSH.
2. Identificar un cliente SSH específico cuya versión de protocolo se presenta como **SSH-2.0-OpenSSH_?** a partir de una imagen de tráfico proporcionada (Figura 1). Replicar este tráfico, generando una conexión con un cliente que emule las características observadas.
3. Modificar la configuración del servidor S1 para que su paquete **Key Exchange Init** tenga un tamaño inferior a 300 bytes, documentando los pasos realizados. La imagen de referencia para este punto es la Figura 2.
4. Explicar el protocolo OpenSSH y cómo sus mecanismos garantizan los principios de seguridad de la información (confidencialidad, integridad, autenticidad, disponibilidad y no repudio), basando el análisis en el tráfico interceptado durante el laboratorio.

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 del informante (Referencia Enunciado Lab. Parte 2).

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: Key Exchange del servidor objetivo (Referencia Enunciado Lab. Parte 3).

1. Desarrollo (Parte 1)

Esta sección detalla la configuración del entorno Docker y el análisis del tráfico SSH para los clientes C1, C2, C3 y C4 conectándose al servidor S1.

1.1. Códigos de cada Dockerfile y su Construcción

Se emplearon cuatro Dockerfiles para construir las imágenes. Fue necesario redirigir APT a `old-releases.ubuntu.com` para las versiones de Ubuntu más antiguas.

1.1.1. C1 (Ubuntu 16.10)

```
FROM ubuntu:16.10
ENV DEBIAN_FRONTEND=noninteractive
RUN sed -i -re 's/([a-z]{2}\.).?archive.ubuntu.com|security.ubuntu.com/old-
    releases.ubuntu.com/g' /etc/apt/sources.list
RUN apt-get update && \
    apt-get install -y openssh-client iproute2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
CMD ["tail", "-f", "/dev/null"]
```

Listing 1: Dockerfile para cliente C1 (Ubuntu 16.10).



```
[+] Building 108.8s (7/7) FINISHED
⇒ [internal] load build definition from Dockerfile                               docker:default
⇒ ⇒ transferring dockerfile: 378B                                              0.0s
⇒ [internal] load metadata for docker.io/library/ubuntu:16.10                      0.05s
⇒ ⇒ [internal] load .dockerignore                                                 1.8s
⇒ ⇒ transferring context: 2B                                                       0.0s
⇒ ⇒ [1/3] FROM docker.io/library/ubuntu:16.10@sha256:8dc9652808dc091400d7d5983949043a9f9c7132b15c14814275d25f 9.6s
⇒ ⇒ ⇒ resolve docker.io/library/ubuntu:16.10@sha256:8dc9652808dc091400d7d5983949043a9f9c7132b15c14814275d25f 0.0s
⇒ ⇒ ⇒ sha256:dca7be20e546564ad2c985dae3c8b0a259454f5637e98b59a3ca6509432cc01 42.79MB / 42.79MB   8.1s
⇒ ⇒ ⇒ sha256:40bca54f5968c2bdb0d8516e2ca4d8f181326a06ff6feee8b4f5e1a36826b8 816B / 816B      0.4s
⇒ ⇒ ⇒ sha256:61464f23390e7d30cffd10a22f27ae6f8f69cc4c1662af2c775f9d657266016 515B / 515B      0.2s
⇒ ⇒ ⇒ sha256:8dc9652808dc091400d7d5983949043a9f9c7132b15c14814275d25f94bca18a 1.36kB / 1.36kB   0.0s
⇒ ⇒ ⇒ sha256:7d3f705d307c7c225398e0d4c4f8512f64eb3a65959a1fb4514dfde18a047e7 3.62kB / 3.62kB   0.0s
⇒ ⇒ ⇒ sha256:d9f0bcd5dc8b557254a1a18c6b78866b9bf460ab1bf2c73cc6aca210408dc67 854B / 854B      0.5s
⇒ ⇒ ⇒ sha256:120db6f90955814bab93a8ca1f19cbcac473fc22833f52f4d29d066135fd10b6 163B / 163B      0.6s
⇒ ⇒ ⇒ extracting sha256:dca7be20e546564ad2c985dae3c8b0a259454f5637e98b59a3ca6509432cc001 1.4s
⇒ ⇒ ⇒ extracting sha256:40bca54f5968c2bdb0d8516e6c2ca4d8f181326a06ff6feee8b4f5e1a36826b8 0.0s
⇒ ⇒ ⇒ extracting sha256:61464f23390e7d30cffd10a22f27ae6f8f69cc4c1662af2c775f9d657266016 0.0s
⇒ ⇒ ⇒ extracting sha256:d99f0bcd5dc8b557254a1a18c6b78866b9bf460ab1bf2c73cc6aca210408dc67 0.0s
⇒ ⇒ ⇒ extracting sha256:120db6f90955814bab93a8ca1f19cbcac473fc22833f52f4d29d066135fd10b6 0.0s
⇒ [2/3] RUN sed -i -re 's/([a-z]{2}\.).?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /e 0.3s
⇒ [3/3] RUN apt-get update && apt-get install -y openssh-client iproute2 && apt-get clean && r 96.9s
⇒ ⇒ exporting to image
⇒ ⇒ writing image sha256:b2ca77fb02f41e94489a2cab76e4f751f1d1ddab7cad0efae220da16c26f244 0.05s
⇒ ⇒ naming to docker.io/library/ubuntu1610_ssh_client 0.0s
```

Figura 3: Resultado de la construcción de la imagen Docker para el cliente C1.

Explicación y Construcción C1: El Listado 1 presenta el Dockerfile para el cliente C1, basado en Ubuntu 16.10. La directiva `RUN sed...` modifica `sources.list` para utilizar los repositorios de versiones antiguas. Posteriormente, se actualiza la lista de paquetes y se instalan `openssh-client` e `iproute2`. La Figura 3 muestra la salida de la terminal que confirma la correcta construcción de esta imagen.

1.1.2. C2 (Ubuntu 18.10)

```
FROM ubuntu:18.10
ENV DEBIAN_FRONTEND=noninteractive
RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-
    releases.ubuntu.com/g' /etc/apt/sources.list
RUN apt-get update && \
    apt-get install -y openssh-client iproute2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
CMD ["tail", "-f", "/dev/null"]
```

Listing 2: Dockerfile para cliente C2 (Ubuntu 18.10).

```
[+] Building 60.9s (7/7) FINISHED
⇒ [internal] load build definition from Dockerfile
⇒ ⇒ transferring dockerfile: 378B
⇒ [internal] load metadata for docker.io/library/ubuntu:18.10
⇒ [internal] load .dockerignore
⇒ ⇒ transferring context: 2B
⇒ [1/3] FROM docker.io/library/ubuntu:18.10@sha256:7d657275047118bb77b052c4c0ae43e8a289ca2879ebfa78a703c93a 8.1s
⇒ ⇒ resolve docker.io/library/ubuntu:18.10@sha256:7d657275047118bb77b052c4c0ae43e8a289ca2879ebfa78a703c93a 0.0s
⇒ ⇒ sha256:230f0701585eb7153c6ba1a9b08f4cfbf6a25d026d7e3b78a47c0965e4c6d0a 868B / 868B 0.5s
⇒ ⇒ sha256:7d657275047118bb77b052c4c0ae43e8a289ca2879ebfa78a703c93a8fd686c 1.42kB / 1.42kB 0.0s
⇒ ⇒ sha256:c95b7b93cc48c3bfd97f8cac6d5ca8053ced584c9e8e6431861ca30b0d73114 1.15kB / 1.15kB 0.0s
⇒ ⇒ sha256:9dc19675e3276d9c028f64ba9a3fbba41e72c779faf8a35603f597310077fd08 3.41kB / 3.41kB 0.0s
⇒ ⇒ sha256:8a532469799e09ef8e1b56ebe39b87c899630c53e86380c13fb46a09e51170e 27.08MB / 27.08MB 7.3s
⇒ ⇒ sha256:32f4dcec3531395ca50469ccb6ca0d24fed1b0b2166c83b25b2f5171c7db62 35.14kB / 35.14kB 0.5s
⇒ ⇒ sha256:e01f70622967c0ca68d6a71ea7ff141c59ab979ac98b5184db665a4ace6415 164B / 164B 0.7s
⇒ ⇒ extracting sha256:8a532469799e09ef8e1b56ebe39b87c8b9630c53e86380c13fb46a09e51170e 0.7s
⇒ ⇒ extracting sha256:32f4dcec3531395ca50469ccb6ca0d24fed1b0b2166c83b25b2f5171c7db62 0.0s
⇒ ⇒ extracting sha256:230f0701585eb7153c6ba1a9b08f4cfbf6a25d026d7e3b78a47c0965e4c6d0a 0.0s
⇒ ⇒ extracting sha256:e01f70622967c0cca68d6a771ea7ff141c59ab979ac98b5184db665a4ace6415 0.0s
⇒ [2/3] RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list 0.3s
⇒ [3/3] RUN apt-get update && apt-get install -y openssh-client iproute2 && apt-get clean && r 45.8s
⇒ exporting to image set: 0.0s 0.1s
⇒ exporting layers 0.0s 0.1s
⇒ writing image sha256:5642f36779c47d5c74a1315f75bf7196ac0ab78cc31ce3fe9809ff818c0891e9 0.0s
⇒ naming to docker.io/library/ubuntu1810_ssh_client 0.0s
```

Figura 4: Resultado de la construcción de la imagen Docker para el cliente C2.

Explicación y Construcción C2: Para el cliente C2, basado en Ubuntu 18.10, el Dockerfile (Listado 2) sigue una estructura análoga a C1, realizando la corrección de repositorios e instalando el cliente OpenSSH. La Figura 4 verifica la finalización exitosa de la construcción de esta imagen.

1.1.3. C3 (Ubuntu 20.10)

```
FROM ubuntu:20.10
ENV DEBIAN_FRONTEND=noninteractive
RUN sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-
    releases.ubuntu.com/g' /etc/apt/sources.list
RUN apt-get update && \
    apt-get install -y openssh-client iproute2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
CMD ["tail", "-f", "/dev/null"]
```

Listing 3: Dockerfile para cliente C3 (Ubuntu 20.10).

```
[+] Building 91.7s (7/7) FINISHED
⇒ [internal] load build definition from Dockerfile
⇒ ⇒ transferring dockerfile: 378B
⇒ [internal] load metadata for docker.io/library/ubuntu:20.10
⇒ [internal] load .dockerignore
⇒ ⇒ transferring context: 2B
⇒ [1/3] FROM docker.io/library/ubuntu:20.10@sha256:a7b08558af07bcccca994b01e1c84f1d14a2156e0099fcf7fcf73f52
⇒ ⇒ resolve docker.io/library/ubuntu:20.10@sha256:a7b08558af07bcccca994b01e1c84f1d14a2156e0099fcf7fcf73f52
⇒ ⇒ sha256:a7b08558af07bcccca994b01e1c84f1d14a2156e0099fcf7fcf73f52d082791e 1.42kB / 1.42kB
⇒ ⇒ sha256:754eb641alba98a8b483c3595a14164fa4ed7f4b457e1aa05f13ce06f8151723 529B / 529B
⇒ ⇒ sha256:e508bd6d694ef622f1f5628104de30be6b58fc6c3c489f62e271b2d6f424a582 1.46kB / 1.46kB
⇒ ⇒ sha256:32f3531c8415258308e8d8dda886004bc90cdf4793f9b97c6f33e7903036294f 31.34MB / 31.34MB
⇒ ⇒ extracting sha256:32f3531c8415258308e8d8dda886004bc90cdf4793f9b97c6f33e7903036294f
⇒ ⇒ [2/3] RUN sed -i -re 's/([-a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
⇒ ⇒ [3/3] RUN apt-get update && apt-get install -y openssh-client iproute2 && apt-get clean && r 81.0s
⇒ ⇒ exporting to image
⇒ ⇒ exporting layers
⇒ ⇒ writing image sha256:d16bf47alef363fecb8716c68f7c02b34ed9cf87f25f3fc9bbd21fc124a683d4
⇒ ⇒ naming to docker.io/library/ubuntu2010_ssh_client
⇒ ⇒ 0.0s
```

Figura 5: Resultado de la construcción de la imagen Docker para el cliente C3.

Explicación y Construcción C3: El cliente C3 se configura sobre Ubuntu 20.10, como se detalla en el Listado 3. Nuevamente, se ajustan las fuentes de APT para la instalación del cliente OpenSSH. La Figura 5 corrobora la compilación satisfactoria de la imagen.

1.1.4. C4/S1 (Ubuntu 22.10)

```
FROM ubuntu:22.10
ENV DEBIAN_FRONTEND=noninteractive
RUN sed -i -re 's/([-a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
RUN apt-get update && \
    apt-get install -y openssh-client openssh-server sudo iproute2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
RUN useradd -m -s /bin/bash prueba && \
    echo "prueba:prueba" | chpasswd && \
    adduser prueba sudo
RUN mkdir -p /var/run/sshd
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config && \
    sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd_config && \
    sed -i 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
RUN ssh-keygen -A
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Listing 4: Dockerfile para C4 (cliente) y S1 (servidor) en Ubuntu 22.10.

```
[+] Building 107.0s (11/11) FINISHED                                            docker:default
⇒ [internal] load build definition from Dockerfile                         0.0s
⇒ ⇒ transferring dockerfile: 844B                                           0.0s
⇒ [internal] load metadata for docker.io/library/ubuntu:22.10                1.2s
⇒ [internal] load .dockerignore                                              0.0s
⇒ ⇒ transferring context: 28 files                                         0.0s
⇒ CACHED [1/7] FROM docker.io/library/ubuntu:22.10@sha256:e322f4808315c387868a9135beeb11435b5b83130a8599fd7 0.0s
⇒ [2/7] RUN sed -i -re '/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com|old-releases.ubuntu.com/g' /etc/ 0.2s
⇒ [3/7] RUN apt-get update && apt-get install -y openssh-client openssh-server sudo iproute2 && a 103.9s
⇒ [4/7] RUN useradd -m -s /bin/bash prueba && echo "prueba:prueba" | chpasswd && adduser prueba sud 0.3s
⇒ [5/7] RUN mkdir -p /var/run/sshd                                         0.3s
⇒ [6/7] RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config && 0.3s
⇒ [7/7] RUN ssh-keygen -A /etc/ssh/ 0.3s
⇒ ⇒ exporting to image: 0 layers                                          0.4s
⇒ ⇒ writing image sha256:26fcc6e9c5dd3ae89381bfbaa3389ec7846efbeb043a5b4007b894a6e177908 0.0s
⇒ ⇒ naming to docker.io/library/ubuntu2210_ssh_server                      0.0s
```

Figura 6: Resultado de la construcción de la imagen Docker para C4/S1.

Explicación y Construcción C4/S1: El Dockerfile para Ubuntu 22.10 (Listado 4) es multifuncional. Instala `openssh-client` (para C4) y `openssh-server` (para S1). Crea el usuario `prueba`, configura `sshd_config` para autenticación por contraseña, genera claves de host y ejecuta `sshd`. La Figura 6 confirma la correcta creación.

1.2. Creación de las credenciales para S1

Las credenciales del usuario `prueba` (usuario: `prueba`, contraseña: `prueba`) y la habilitación de la autenticación por contraseña para S1 se especificaron en el Dockerfile (Listado 4).

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

Figura 7: Salida del script `hassh_debug.py` para C1 → S1.

Análisis HASSH y Versión C1: La Figura 7 muestra que el HASSH del cliente C1 (Ubuntu 16.10, IP 172.19.0.3) es 0e4584cb9f2dd077dbf8ba0df8112d8e. La cadena completa de algoritmos que lo genera es: curve25519-sha256@libssh.org,ecdh-sha2-nistp256,...,ext-info-c;chacha20-poly1305@openssh.com,...,3des-cbc;umac-64-etm@openssh.com,...,hmac-sha1;none,zlib@openssh.com,zlib.

```
tshark -r ./c1_to_s1.pcap -Y "ip.src = 172.19.0.3 && ssh.protocol" -T fields -e ssh.protocol
```

La versión de este cliente, extraída del tráfico (Figura 8), es SSH-2.0-OpenSSH_7.3p1Ubuntu-1~16.04.1

Általános tanulási támogatás G1

```
└$ tshark -r ./c1_to_s1.pcap -Y "ip.src = 172.19.0.3 && ssh.protocol" -T fields -e frame.number -e frame.len -e ssh.protocol
4 172.19.0.3 172.19.0.1 1 SSH-2.0-OpenSSH_7.2p1 Ubuntu-1ubuntu0.1
```

Figura 9: C1: Paquete de Identificación de Protocolo del Cliente (Frame 4, 107 bytes)

Paquete Protocolo C1: La Figura 9 detalla el primer paquete enviado por C1, su string de protocolo, localizado en el frame 4 de la captura, con una longitud total de 107 bytes.

```
└$ tshark -r ./c1_to_s1.pcap -Y "ip.src = 172.19.0.3 && ssh.message_code = 20" -T fields -e frame.number -e frame.len -e ssh.packet_length -e ssh.kex_algorithms -e ssh.server_host_key_algorithms -e ssh.encryption_algorithms_client_to_server -e ssh.mac_algorithms_client_to_server -e ssh.compression_algorithms_client_to_server -e ssh.compression_algorithms_server_to_client
8      1498    curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group14-sha1,ext-info-c ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,3des-cbc chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,3des-cbc 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,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1 umac-64-etm@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1 none,zlib@openssh.com,zlib none,zlib@openSSH.com,zlib
```

Figura 10: C1: Paquete KEXINIT (Frame 8, 1498B total, 1428B payload SSH).

Paquete KEXINIT C1: Seguidamente, C1 envía su paquete Key Exchange Init (KEXINIT) (Figura 10), crucial para la negociación de algoritmos.

```
└$ tshark -r ./c1_to_s1.pcap -Y "frame.number == 10" -o ssh
Frame 10: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
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: 45302, Dst Port: 22, Seq: 1474, Ack: 1122, Len: 48
SSH Protocol
  SSH Version 2
    Packet Length: 44
    Padding Length: 6
    Key Exchange (method:curve25519-sha256@libssh.org) application failed to start because no Qt platform plugin could be initialized
      Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
      ECDH client's ephemeral public key length: 32
      ECDH client's ephemeral public key (Q_C): 4b329c50056b9a7c54744a7bc1453a3896b916099f5e04eb6d0e840a6f61dc0c deg1
    Padding String: 000000000000
    [Sequence number: 1] along on fatal log level exception
    [Direction: client-to-server]
```

Figura 11: C1: Paquete ECDH Key Exchange Init (Frame 10, 114B total, 44B payload SSH).

Paquete ECDH Init C1: El intercambio de claves prosigue con el envío del material de clave efímera de C1 (Figura 11).

```
└$ tshark -r ./c1_to_s1.pcap -Y "frame.number == 12" -o ssh
Frame 12: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
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: 45302, Dst Port: 22, Seq: 1522, Ack: 1718, Len: 16
SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none) no Qt could be installed the application may fix this problem.
  Packet Length: 12
  Padding Length: 10
  Key Exchange (method:curve25519-sha256@libssh.org) isplay, offscreen, xcb, eglfs, vnc, linux-land-egl, Message Code: New Keys (21)
  Padding String: 00000000000000000000000000000000
  [Sequence number: 2] along on fatal log level exception
  [Direction: client-to-server]
```

Figura 12: C1: Paquete New Keys (Frame 12, 82B total, 12B payload SSH).

Paquete New Keys C1: C1 envía New Keys (Figura 12), marcando la transición a comunicación cifrada.

```

└$ tshark -r ./c1_to_s1.pcap -Y "ip.src = 172.19.0.3 && frame.number > 12"
  14  0.056862  172.19.0.3 → 172.19.0.2  SSHv2 110 Client: Encrypted packet (len=44)
  17  0.057320  172.19.0.3 → 172.19.0.2  SSHv2 134 Client: Encrypted packet (len=68)
  19  0.108811  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=1650 Ack=1814 Win=68352 Len=0 TSval=1882005
151 TSecr=2346950914
  20  3.700332  172.19.0.3 → 172.19.0.2  SSHv2 214 Client: Encrypted packet (len=148)
  23  3.761049  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=1798 Ack=1842 Win=68352 Len=0 TSval=1882008
803 TSecr=2346954610
  24  3.761339  172.19.0.3 → 172.19.0.2  SSHv2 178 Client: Encrypted packet (len=112)
  27  3.824843  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=1910 Ack=2470 Win=68352 Len=0 TSval=1882008
867 TSecr=2346954630
  29  3.825007  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=1910 Ack=2514 Win=68352 Len=0 TSval=1882008
867 TSecr=2346954674
  30  3.825331  172.19.0.3 → 172.19.0.2  SSHv2 442 Client: Encrypted packet (len=376)
  33  3.829350  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2286 Ack=3114 Win=68480 Len=0 TSval=1882008
871 TSecr=2346954677
  36  3.841015  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2286 Ack=3338 Win=68608 Len=0 TSval=1882008
883 TSecr=2346954685
  37  16.410313  172.19.0.3 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  39  16.456788  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2322 Ack=3374 Win=68608 Len=0 TSval=1882021
499 TSecr=2346967260
  40  16.713008  172.19.0.3 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  42  16.713529  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2358 Ack=3410 Win=68608 Len=0 TSval=1882021
755 TSecr=2346967562
  43  16.905101  172.19.0.3 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  45  16.905899  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2394 Ack=3446 Win=68608 Len=0 TSval=1882021
948 TSecr=2346967755
  46  17.259427  172.19.0.3 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  48  17.259958  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2430 Ack=3482 Win=68608 Len=0 TSval=1882022
302 TSecr=2346968109
  49  18.450817  172.19.0.3 → 172.19.0.2 5 SSHv2 102 Client: Encrypted packet (len=36)
  51  18.451581  172.19.0.3 → 172.19.0.2 5 TCP 66 45302 → 22 [ACK] Seq=2466 Ack=3542 Win=68608 Len=0 TSval=1882023
493 TSecr=2346969300
  53  18.454019  172.19.0.3 → 172.19.0.2 5 TCP 66 45302 → 22 [ACK] Seq=2466 Ack=3718 Win=68608 Len=0 TSval=1882023
496 TSecr=2346969303
  54  18.454159  172.19.0.3 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  55  18.454201  172.19.0.3 → 172.19.0.2  SSHv2 126 Client: Encrypted packet (len=60)
  56  18.454236  172.19.0.3 → 172.19.0.2 5 TCP 66 45302 → 22 [FIN, ACK] Seq=2562 Ack=3718 Win=68608 Len=0 TSval=1882023
82023496 TSecr=2346969303
  59  18.457232  172.19.0.3 → 172.19.0.2  TCP 66 45302 → 22 [ACK] Seq=2563 Ack=3719 Win=68608 Len=0 TSval=1882023
499 TSecr=2346969306

```

Figura 13: C1: Primer paquete cifrado, probable Service Request (Frame 14).

```

  17  0.057320  172.19.0.3 → 172.19.0.2  SSHv2 134 Client: Encrypted packet (len=68)

```

Figura 14: C1: Segundo paquete cifrado, probable Userauth Request (Frame 17).

Paquetes Cifrados de Autenticación C1: Las Figuras 13 y 14 muestran los primeros paquetes cifrados de C1, correspondientes a la fase de autenticación.

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

Figura 15: Salida HASSH para C2 (Ubuntu 18.10).

Análisis HASSH y Versión C2: El cliente C2 (IP 172.19.0.4) presenta el HASSH 06 046964c022c6407d15a27b12a6a4fb (Figura 15). Su versión (Figura 16) es SSH-2.0-OpenSSH_7.7p1Ubuntu-4ubuntu0.3.

```
└$ tshark -r ./c2_to_s1.pcap -Y "ip.src = 172.19.0.4 && ssh.protocol" -T fields -e ssh.protocol  
SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3
```

Figura 16: Versión del cliente OpenSSH C2

Análisis de paquetes C2: Los paquetes de C2 siguen una estructura similar a C1. El Protocol String se observa en la Figura 17. El KEXINIT, con un payload SSH de 1356 bytes, se muestra en la Figura 18. El ECDH Init y New Keys se detallan en las Figuras 19 y 20 respectivamente. Los paquetes de autenticación cifrados se ven en las Figuras 21 y 22.

```
$ tshark -r ./c2_to_s1.pcap -Y "ip.src = 172.19.0.4 && ssh.protocol" -T fields -e frame.number -e frame.len -e ss.h.protocol  
4      107    SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3
```

Figura 17: C2: Paquete Protocolo.

```
└$ tshark -r ./c2_to_s1.pcap -Y "ip.src = 172.19.0.4 && ssh.message_code = 20" -T fields -e frame.number -e frame.len -e ssh.packet_length -e ssh.kex.algorithms -e ssh.server_host_key_algorithms -e ssh.encryption_algorithms_client_to_server -e ssh.compression_algorithms_server_to_client -e ssh.mac_algorithms_client_to_server -e ssh.mac_algorithms_server_to_client -e ssh.compression_algorithms_client_to_server -e ssh.compression_algorithms_server_to_client
8      1426    1356  curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c          ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa  chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com          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,umac-128@openssh.com,hmac-sha2-256,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-512,umac-sha1  none,zlib@openssh.com,zlib      none,zlib@openssh.com,zlib
```

Figura 18: C2: Paquete KEXINIT.

```
└$ tshark -r ./c2_to_s1.pcap -Y "frame.number == 10" -0 ssh
Frame 10: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
Ethernet II, Src: 02:42:ac:13:00:04 (02:42:ac:13:00:04), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
Internet Protocol Version 4, Src: 172.19.0.4, Dst: 172.19.0.2
Transmission Control Protocol, Src Port: 46860, Dst Port: 22, Seq: 1402, Ack: 1122, Len: 48
SSH Protocol          https://ubuntu.com/advantage
  SSH Version 2
  This by Packet Length: 44 sized by removing packages and content that are
  not required for users to log into.
  Padding Length: 6 that users do not log into.
  Key Exchange (method:curve25519-sha256)
  To restore Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
  Last login: ECDH client's ephemeral public key length: 32
  To run a command ECDH client's ephemeral public key (Q_C): d92d933121672c76a7db2f5ddbd6148cb29dbe1e510a9b12df7104d45fa4a4
  4e  man sudo root for details.
  Padding String: 000000000000
prueba@ [Sequence number: 1]
logo [Direction: client-to-server]
Connection to s1 closed.
```

Figura 19: C2: Paquete ECDH Init.

```
└$ tshark -r ./c2_to_s1.pcap -Y "frame.number == 12" -0 ssh
Frame 12: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
Ethernet II, Src: 02:42:ac:13:00:04 (02:42:ac:13:00:04), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
Internet Protocol Version 4, Src: 172.19.0.4, Dst: 172.19.0.2
Transmission Control Protocol, Src Port: 46860, Dst Port: 22, Seq: 1450, Ack: 1718, Len: 16
SSH Protocol          on a system that users do not log into.
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
  To restore Packet Length: 12 you can run the 'umminimize' command.
  Last login: 10:45:33 2025 from 172.19.0.3
  To run a Key Exchange (method:curve25519-sha256) use "sudo <command>".
  See "man sudo_root" for details.
  Message Code: New Keys (21)
  Padding String: 00000000000000000000
prueba@ [Sequence number: 2]
logo [Direction: client-to-server]
```

Figura 20: C2: Paquete New Keys.

```

└$ tshark -r ./c2_to_s1.pcap -Y "ip.src = 172.19.0.4 && frame.number > 12"
  14  0.058459  172.19.0.4 → 172.19.0.2  SSHv2 110 Client: Encrypted packet (len=44)
  17  0.059154  172.19.0.4 → 172.19.0.2  SSHv2 134 Client: Encrypted packet (len=68)
  19  0.110239  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=1578 Ack=1814 Win=68352 Len=0 TSval=3737958
945 TSecr=791210450
  20  4.750267  172.19.0.4 → 172.19.0.2  SSHv2 214 Client: Encrypted packet (len=148)
  23  4.811099  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=1726 Ack=1842 Win=68352 Len=0 TSval=3737963
645 TSecr=791215194
  24  4.811408  172.19.0.4 → 172.19.0.2  SSHv2 178 Client: Encrypted packet (len=112)
  27  4.870308  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=1838 Ack=2470 Win=68352 Len=0 TSval=3737963
705 TSecr=791215213
  29  4.870500  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=1838 Ack=2514 Win=68352 Len=0 TSval=3737963
705 TSecr=791215254
  30  4.870806  172.19.0.4 → 172.19.0.2  SSHv2 442 Client: Encrypted packet (len=376)
  33  4.874981  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2214 Ack=3114 Win=68480 Len=0 TSval=3737963
709 TSecr=791215257
  36  4.886498  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2214 Ack=3338 Win=68608 Len=0 TSval=3737963
721 TSecr=791215265
  37  8.307333  172.19.0.4 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  39  8.350321  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2250 Ack=3374 Win=68608 Len=0 TSval=3737967
185 TSecr=791218691
  40  9.005977  172.19.0.4 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  42  9.006497  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2286 Ack=3410 Win=68608 Len=0 TSval=3737967
841 TSecr=791219390
  43  9.466242  172.19.0.4 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  45  9.466832  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2322 Ack=3446 Win=68608 Len=0 TSval=3737968
301 TSecr=791219850
  46  9.889027  172.19.0.4 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  48  9.889619  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2358 Ack=3482 Win=68608 Len=0 TSval=3737968
724 TSecr=791220273
  49  14.714027  172.19.0.4 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  51  14.714645  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2394 Ack=3534 Win=68608 Len=0 TSval=3737973
549 TSecr=791225098
  53  14.714698  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2394 Ack=3578 Win=68608 Len=0 TSval=3737973
549 TSecr=791225098
  55  14.717729  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2394 Ack=3754 Win=68608 Len=0 TSval=3737973
552 TSecr=791225101
  56  14.717836  172.19.0.4 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
  57  14.717885  172.19.0.4 → 172.19.0.2  SSHv2 126 Client: Encrypted packet (len=60)
  58  14.717926  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [FIN, ACK] Seq=2490 Ack=3754 Win=68608 Len=0 TSval=3737973
37973552 TSecr=791225101
  61  14.721252  172.19.0.4 → 172.19.0.2  TCP 66 46860 → 22 [ACK] Seq=2491 Ack=3755 Win=68608 Len=0 TSval=3737973
556 TSecr=791225105

```

Figura 21: C2: Primer paquete cifrado (ServiceReq).

```
17  0.059154  172.19.0.4 → 172.19.0.2  SSHv2 134 Client: Encrypted packet (len=68)
```

Figura 22: C2: Segundo paquete cifrado (UserAuthReq).

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

Figura 23: Salida HASSH para C3 (Ubuntu 20.10).

Análisis HASSH y Versión C3: El cliente C3 (IP 172.19.0.5) tiene un HASSH ae8bd7dd09970555aa4c6ed22adbbf56 (Figura 23). Su versión es SSH-2.0-OpenSSH_8.3p1Ubuntu-1ubuntu0.1 (Figura 24).

```
tshark -r ./c3_to_s1.pcap -Y "ip.src = 172.19.0.5 && ssh.protocol" -T fields -e ssh.protocol
```

Figura 24: Versión del cliente OpenSSH C3.

Análisis de paquetes C3: Los paquetes de C3 (Protocolo en Figura 25, KEXINIT en Figura 26 con payload SSH de 1508 bytes, ECDH Init en Figura 27, New Keys en Figura 28, y paquetes cifrados en Figuras 29 y 30) siguen el flujo esperado.

```
$ tshark -r ./c3_to_s1.pcap -Y "ip.src == 172.19.0.5 && ssh.protocol" -T fields -e frame.number -e frame.len -e ss.h.protocol  
4      107      SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1
```

Figura 25: C3: Paquete Protocolo.

```
└$ tshark -r ./c3_to_s1.pcap -Y "ip.src = 172.19.0.5 && ssh.message_code = 20" -T fields -e frame.number -e frame.len -e ssh.packet_length -e ssh.kex_algorithms -e ssh.server_host_key_algorithms -e ssh.encryption_algorithms_client_to_server -e ssh.compression_algorithms_server_to_client -e ssh.mac_algorithms_client_to_server -e ssh.mac_algorithms_server_to_client
8      1578  curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,ext-info-c      ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,ssh-ed25519-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,ssh-ed25519,sk-ssh-ed25519@openssh.com,rsa-sha2-512,rsa-sha2-256,ssh-rsa      chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com      chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com      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,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1      umac-64-etm@openssh.com,umac-128-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1      none,zlib@openssh.com,zlib      none,zlib@openssh.com,zlib
```

Figura 26: C3: Paquete KEXINIT.

```
└$ tshark -r ./c3_to_s1.pcap -Y "frame.number == 11" -0 ssh
Frame 11: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
Ethernet II, Src: 02:42:ac:13:00:05 (02:42:ac:13:00:05), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
Internet Protocol Version 4, Src: 172.19.0.5, Dst: 172.19.0.2
Transmission Control Protocol, Src Port: 35580, Dst Port: 22, Seq: 1554, Ack: 1122, Len: 48
SSH Protocol      https://ubuntu.com/advantage
    SSH Version 2
        This is a Packet Length: 44 maximized by removing packages and content that are
        not required for users do not log into.
        Padding Length: 6
        Key Exchange (method:curve25519-sha256)
        To restore the Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
        Last login:      ECDH client's ephemeral public key length: 32
        To run a command: ECDH client's ephemeral public key (Q_C): 2c17ef1f2285bf1b2893bb3c903dc6e769fd0bd685e7d94a4e97409086c76a
33      Max sudo rights for details.
        Padding String: 000000000000
        [Sequence number: 1]
        [Direction: client-to-server]
```

Figura 27: C3: Paquete ECDH Init.

```
└$ tshark -r ./c3_to_s1.pcap -Y "frame.number == 13" -0 ssh
Frame 13: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
Ethernet II, Src: 02:42:ac:13:00:05 (02:42:ac:13:00:05), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
Internet Protocol Version 4, Src: 172.19.0.5, Dst: 172.19.0.2
Transmission Control Protocol, Src Port: 35580, Dst Port: 22, Seq: 1602, Ack: 1718, Len: 16
SSH Protocol      https://ubuntu.com/advantage
    SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    This is a Packet Length: 12 maximized by removing packages and content that are
    not required for users do not log into.
    Padding Length: 10
    Key Exchange (method:curve25519-sha256)
    To restore the Message Code: New Keys (21) unminimize command.
    Last login:      Padding String: 00000000000000000000000000000000
    To run a command: [Sequence number: 2] -at- (user "root"), use "sudo <command>".
    See [Direction: client-to-server]
```

Figura 28: C3: Paquete New Keys.

```
└$ tshark -r ./c3_to_s1.pcap -Y "ip.src = 172.19.0.5 && frame.number > 13"
 15  0.059197  172.19.0.5 → 172.19.0.2  SSHv2 110 Client: Encrypted packet (len=44)
 18  0.059769  172.19.0.5 → 172.19.0.2  SSHv2 134 Client: Encrypted packet (len=68)
 20  0.114999  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=1730 Ack=1814 Win=68352 Len=0 TSval=2084637
130 TSecr=3018645796
 21  4.363772  172.19.0.5 → 172.19.0.2  SSHv2 214 Client: Encrypted packet (len=148)
 24  4.424209  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=1878 Ack=1842 Win=68352 Len=0 TSval=2084641
439 TSecr=3018650153
 25  4.424458  172.19.0.5 → 172.19.0.2  SSHv2 178 Client: Encrypted packet (len=112)
 28  4.483107  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=1990 Ack=2470 Win=68352 Len=0 TSval=2084641
498 TSecr=3018650171
 30  4.483290  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=1990 Ack=2514 Win=68352 Len=0 TSval=2084641
498 TSecr=3018650212
 31  4.483551  172.19.0.5 → 172.19.0.2  SSHv2 442 Client: Encrypted packet (len=376)
 34  4.486822  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2366 Ack=3114 Win=68480 Len=0 TSval=2084641
501 TSecr=3018650215
 37  4.498565  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2366 Ack=3338 Win=68608 Len=0 TSval=2084641
513 TSecr=3018650222
 38  12.395272  172.19.0.5 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
 40  12.438973  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2402 Ack=3374 Win=68608 Len=0 TSval=2084649
454 TSecr=3018658125
 41  12.663048  172.19.0.5 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
 43  12.663503  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2438 Ack=3410 Win=68608 Len=0 TSval=2084649
678 TSecr=3018658392
 44  13.013244  172.19.0.5 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
 46  13.013727  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2474 Ack=3446 Win=68608 Len=0 TSval=2084650
028 TSecr=3018658742
 47  13.471706  172.19.0.5 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
 49  13.472588  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2510 Ack=3482 Win=68608 Len=0 TSval=2084650
487 TSecr=3018659201
 50  14.532264  172.19.0.5 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
 52  14.532692  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2546 Ack=3534 Win=68608 Len=0 TSval=2084651
547 TSecr=3018660261
 54  14.532747  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2546 Ack=3578 Win=68608 Len=0 TSval=2084651
547 TSecr=3018660261
 56  14.535353  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2546 Ack=3754 Win=68608 Len=0 TSval=2084651
550 TSecr=3018660264
 57  14.535486  172.19.0.5 → 172.19.0.2  SSHv2 102 Client: Encrypted packet (len=36)
 58  14.535533  172.19.0.5 → 172.19.0.2  SSHv2 126 Client: Encrypted packet (len=60)
 59  14.535582  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [FIN, ACK] Seq=2642 Ack=3754 Win=68608 Len=0 TSval=2084650
84651550 TSecr=3018660264
 62  14.538840  172.19.0.5 → 172.19.0.2  TCP 66 35580 → 22 [ACK] Seq=2643 Ack=3755 Win=68608 Len=0 TSval=2084651
553 TSecr=3018660267
```

Figura 29: C3: Primer paquete cifrado.

```
18  0.059769  172.19.0.5 → 172.19.0.2  SSHv2 134 Client: Encrypted packet (len=68)
```

Figura 30: C3: Segundo paquete cifrado.

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

```
→ $ sudo docker exec -it s1 ssh prueba@localhost
The authenticity of host 'localhost (::1)' can't be established.
ED25519 key fingerprint is SHA256:U/z51a0kJrFm6lds57FuQrR33rAv2zid0Hbq3JZSKc.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
prueba@localhost's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.12.25-amd64 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: Tue Jun 17 07:19:29 2025 from 172.19.0.5
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
Source IP: Source Port: Dest IP: Dest Port: HASSH
prueba@s1:~$ exit
logout
Connection to localhost closed.
```

Figura 31: Proceso de la conexión SSH de C4 a localhost.

Conexión C4: El cliente C4 (Ubuntu 22.10), actuando desde el mismo contenedor que S1, se conectó a localhost (::1 en IPv6), puerto efímero 55424, como se muestra en la Figura 31.

Figura 32: Salida HASSH para el cliente C4.

Análisis HASSH y Versión C4: Su HASSH (cliente) es 78c05d999799066a2b4554ce7b1585a6 (Figura 32). La versión es SSH-2.0-OpenSSH_9.0p1Ubuntu-1ubuntu7.3 (Figura 33).

```
$ tshark -r ./c4_to_s1.pcap -Y "tcp.srcport = 55424 && ssh.protocol" -T fields -e ssh.protocol
SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
```

Figura 33: Versión del cliente OpenSSH C4.

Análisis de paquetes C4: El Protocol String (Figura 34, sobre IPv6) es el frame 4 (127B). El KEXINIT (Figura 35, frame 8) tiene 1590B totales y 1500B de payload SSH. El paquete de KEX Payload para el algoritmo post-cuántico (Figura 36, frame 11) es notablemente grande, con 1294B totales y 1204B de payload SSH. El New Keys (Figura 37, frame 14) tiene 102B totales (IPv6) y 12B de payload SSH. Los paquetes cifrados de autenticación (Figuras 38 y 39) siguen el patrón.

```
$ tshark -r ./c4_to_s1.pcap -Y "tcp.srcport = 55424 && ssh.protocol" -T fields -e frame.number -e frame.len -e ssh.protocol
4      127 0.111000000 SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
```

Figura 34: C4: Paquete Protocolo (sobre IPv6).

```
$ tshark -r ./c4_to_s1.pcap -Y "tcp.srcport = 55424 && ssh.message_code = 20" -T fields -e frame.number -e frame.len -e ssh.packet_length -e ssh.kex_algorithms -e ssh.server_host_key_algorithms -e ssh.encryption_algorithms_client_to_server -e ssh.encryption_algorithms_server_to_client -e ssh.mac_algorithms_client_to_server -e ssh.mac_algorithms_server_to_client
8      1590 1500 sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,ext-info-c ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,sk-sh-ed25519-cert-v01@openssh.com,sk-ecdsa-sha2-nistp256-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ssh-ed25519@openssh.com,sk-ecdsa-sha2-nistp256@openssh.com,rsa-sha2-512,rsa-sha2-256 chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1 umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-56-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1 none,zlib@openssh.com,zlib none,zlib@openssh.com,zlib
```

Figura 35: C4: Paquete KEXINIT.

```

└$ tshark -r ./c4_to_s1.pcap -Y "frame.number == 11" -O ssh
Frame 11: 1294 bytes on wire (10352 bits), 1294 bytes captured (10352 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: 55424, Dst Port: 22, Seq: 1546, Ack: 1122, Len: 1208
SSH Protocol
    SSH Version 2, password?
        Packet Length: 1204 [/Linux 6.12.25-aml64 x86_64]
        Padding Length: 8
    * Debug Key Exchange (method:sntrup761x25519-sha512@openssh.com)
    * Management Message Code: Diffie-Hellman Key Exchange Init (30)
    * Support: Invalid key length: 1190
        [Expert Info (Error/Protocol): Invalid key length: 1190]
    This system has been configured to minimize the amount of log output
    not required for security or troubleshooting purposes. If you need to
    [Severity level: Error] log into this system, you can run the 'logger'
    [Group: Protocol]
    To restore the original configuration, run the 'logger' command.
Last login: DH client e [...] 2d30b09685c56fib83502cb24cef8ab3701b11713e47afec39607179f3e33be5d3e5a8d1b5f18697e850850
ea84ee64bcce865d8bc533011c1f56606dc1b682e072772fa0ac4ae0d593d66831595734fb12a234a6be76c09278f72c942f7e1728298dfa7b8
0a6318b68a51863c8
        Padding String: 0000000000000000
    prueban [Sequence number: 2]
    login [Direction: client-to-server]
Connection to localhost closed.

```

Figura 36: C4: Paquete de Intercambio de Claves Post-Cuántico.

```

└$ tshark -r ./c4_to_s1.pcap -Y "frame.number == 14" -O ssh
Frame 14: 102 bytes on wire (816 bits), 102 bytes captured (816 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: 55424, Dst Port: 22, Seq: 2754, Ack: 2686, Len: 16
SSH Protocol
    SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
    Packet Length: 12
    Last login: 2025-09-29 10:25 from 172.19.0.5
    To run a Key Exchange (method:sntrup761x25519-sha512@openssh.com) and >".
    See "man sshd" for more information.
        Message Code: New Keys (21)
        Padding String: 00000000000000000000000000000000
    prueban [Sequence number: 2]
    login [Direction: client-to-server]
Connection to localhost closed.

```

Figura 37: C4: Paquete New Keys (sobre IPv6).

```

└$ tshark -r ./c4_to_s1.pcap -Y "tcp.srcport == 55424 && frame.number > 14"
 16  4.105363  https://::1 → ::1  SSHv2 130 Client: Encrypted packet (len=44)
 19  4.105675  https://::1 → ::1  TCP 86 55424 → 22 [ACK] Seq=2814 Ack=2730 Win=79360 Len=0 TSval=6669341
29 TSecr=666934129 https://ubuntu.com/advantage
 20  4.105932  ::1 → ::1      SSHv2 154 Client: Encrypted packet (len=68)
22  4.157507 been minim::1 → ::1 removing p TCP 86 55424 → 22 [ACK] Seq=2882 Ack=2782 Win=79360 Len=0 TSval=6669341
81 TSecr=666934137 system that users do not log into
 23  9.433138  ::1 → ::1      SSHv2 234 Client: Encrypted packet (len=148)
 26  9.493822 content, you can't log into the system that users do not log into
 27  9.494090 as administrator → ::1 → ::1  SSHv2 198 Client: Encrypted packet (len=112)
 30  9.55748207 for de minimis TCP 86 55424 → 22 [ACK] Seq=3030 Ack=2810 Win=79360 Len=0 TSval=6669395
17 TSecr=666939517 17:07:04:05.2025 From 172.19.0.5
 27  9.494090 as administrator → ::1 → ::1  SSHv2 198 Client: Encrypted packet (len=112)
 30  9.55748207 for de minimis TCP 86 55424 → 22 [ACK] Seq=3142 Ack=3438 Win=80000 Len=0 TSval=6669395
81 TSecr=666939537
 32  9.5575301   ::1 → ::1      TCP 86 55424 → 22 [ACK] Seq=3142 Ack=3482 Win=80000 Len=0 TSval=6669395
81 TSecr=666939581
 33  9.557838 closed.   ::1 → ::1      SSHv2 462 Client: Encrypted packet (len=376)
 36  9.560743    ::1 → ::1      TCP 86 55424 → 22 [ACK] Seq=3518 Ack=4082 Win=80000 Len=0 TSval=6669395
84 TSecr=666939583 user@kali:~/~/.Cripto
 39  9.571443 exec -it ssh-1@localhost:10000 TCP 86 55424 → 22 [ACK] Seq=3518 Ack=4306 Win=80256 Len=0 TSval=6669395
95 TSecr=666939590 host 'localhost' (127.0.0.1) can't be established.
 40  20.878410 reprinted is ::1 → ::1/5140kIn  SSHv2 122 Client: Encrypted packet (len=36)
 42  20.925463 known by ::1 → ::1 using otherTCP 86 55424 → 22 [ACK] Seq=3554 Ack=4342 Win=80256 Len=0 TSval=6669509
49 TSecr=666950902 stsc:17 (hashed name)
 43  21.217355 want to connect ::1 → ::1 connecting  SSHv2 122 Client: Encrypted packet (len=36)
 45  21.218269 only added ::1 → ::1 to the ED255 TCP 86 55424 → 22 [ACK] Seq=3590 Ack=4378 Win=80256 Len=0 TSval=6669512
41 TSecr=666951241 password
 46  21.433933 user@kali:22:10 (0) ::1 → ::1 12.25 a  SSHv2 122 Client: Encrypted packet (len=36)
 48  21.434341    ::1 → ::1      TCP 86 55424 → 22 [ACK] Seq=3626 Ack=4414 Win=80256 Len=0 TSval=6669514
58 TSecr=666951458 https://help.ubuntu.com
 49  21.801672  https://::1  SSHv2 122 Client: Encrypted packet (len=36)
 51  21.801939  https://::1/om/advant TCP 86 55424 → 22 [ACK] Seq=3662 Ack=4450 Win=80256 Len=0 TSval=6669518
25 TSecr=666951825
 52  23.574273 been minim::1 → ::1 removing p  SSHv2 122 Client: Encrypted packet (len=36)
 54  23.574768 a system that users do not log into TCP 86 55424 → 22 [ACK] Seq=3698 Ack=4510 Win=80256 Len=0 TSval=6669535
98 TSecr=666953598
 56  23.576878 content, you can't log into the system that users do not log into TCP 86 55424 → 22 [ACK] Seq=3698 Ack=4686 Win=80256 Len=0 TSval=6669536
00 TSecr=666953600 17:07:10:29.2025 From 172.19.0.5
 57  23.576999 as administrator ::1 → ::1  SSHv2 122 Client: Encrypted packet (len=36)
 58  23.57702507 for de minimis ::1 → ::1  SSHv2 146 Client: Encrypted packet (len=60)
 59  23.577056    ::1 → ::1      TCP 86 55424 → 22 [FIN, ACK] Seq=3794 Ack=4686 Win=80256 Len=0 TSval=6669536
6953600 TSecr=666953600
 62  23.579932    ::1 → ::1      TCP 86 55424 → 22 [ACK] Seq=3795 Ack=4687 Win=80256 Len=0 TSval=6669536
03 TSecr=666953603 host closed.

```

Figura 38: C4: Primer paquete cifrado (ServiceReq, sobre IPv6).

20	4.105932	::1 → ::1	SSHv2 154 Client: Encrypted packet (len=68)
----	----------	-----------	---

Figura 39: C4: Segundo paquete cifrado (UserAuthReq, sobre IPv6).

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

myceliumbroker Add files via upload · 2 years ago

214 lines (214 loc) · 212 KB

Preview Code Blame Raw ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

Q 0e4584cb9f2dd077dbf8ba0df8112d8e

1	hassh	observations	versions

Figura 40: Búsqueda del HASSH de C1 en Mycelium Broker (no se encontró coincidencia).

Cliente C1: El HASSH 0e4584cb9f2dd077dbf8ba0df8112d8e no fue hallado en Mycelium Broker (Figura 40).

myceliumbroker Add files via upload · 2 years ago

214 lines (214 loc) · 212 KB

Preview Code Blame Raw ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

Q 06046964c022c6407d15a27b12a6a4fb

1	hassh	observations	versions
41	06046964c022c6407d15a27b12a6a4fb	1620	SSH-2.0-OpenSSH_7.6p1 Debian-2, SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.5, SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3

Figura 41: Coincidencia del HASSH de C2 en la base de datos de Mycelium Broker.

Cliente C2: El HASSH 06046964c022c6407d15a27b12a6a4fb sí encontró correspondencia con OpenSSH_7.6p1Ubuntu-4ubuntu0.3 (Figura 41).

	hash	observations	versions
1	ae8bd7dd09970555aa4c6ed22adbbf56	3919	SSH-2.0-OpenSSH_8.4p1 Raspbian-5+deb11u1, SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u1, S

Figura 42: Coincidencias del HASSH de C3 en la base de datos de Mycelium Broker.

Cliente C3: El HASSH `ae8bd7dd09970555aa4c6ed22adbbf56` se asoció con versiones `OpenSSH_8.2p1/8.4p1` (Figura 42).

	hash	observations	versions
190	78c05d999799066a2b4554ce7b1585a6	2	SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7

Figura 43: Coincidencia directa del HASSH de C4 en la base de datos de Mycelium Broker.

Cliente C4: El HASSH `78c05d999799066a2b4554ce7b1585a6` tuvo una coincidencia directa con `OpenSSH_9.0p1Ubuntu-1ubuntu7` (Figura 43).

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

La información detallada se encuentra en las subsecciones 1.3 a 1.6. En resumen: Protocol String (versión), KEXINIT (algoritmos), KEX Payloads (claves efímeras), New Keys (inicio de cifrado).

1.9. Diferencia entre C1 y C2

Al comparar el cliente C1 (OpenSSH 7.3p1, Ubuntu 16.10) con C2 (OpenSSH 7.7p1, Ubuntu 18.10), se observan las siguientes diferencias clave:

- **Versión de OpenSSH:** C2 utiliza una versión más reciente (7.7p1 vs 7.3p1).

- **HASSH:** Distinto (0e45... para C1, 0604... para C2), reflejando cambios en los algoritmos.
- **Algoritmos KEX:** C2 añade `curve25519-sha256` (sin sufijo @libssh.org) como su primera preferencia.
- **Algoritmos de Cifrado:** C2 elimina de su oferta por defecto los cifrados CBC (`aes128-cbc`, `aes192-cbc`, `aes256-cbc`) y el obsoleto `3des-cbc`, que C1 sí ofrecía. C2 se centra en modos de cifrado más modernos y seguros como CTR y GCM. Esta reducción en la lista de cifrados es la causa principal de que el payload del KEXINIT de C2 (1356 bytes) sea más corto que el de C1 (1428 bytes).

1.10. Diferencia entre C2 y C3

La transición de C2 (OpenSSH 7.7p1, Ubuntu 18.10) a C3 (OpenSSH 8.3p1, Ubuntu 20.10) introduce más refinamientos:

- **Versión de OpenSSH:** Salto a 8.3p1 desde 7.7p1.
- **HASSH:** Cambia (0604... para C2, ae8b... para C3).
- **Algoritmos KEX:** C3 elimina de su lista los algoritmos de intercambio de grupos Diffie-Hellman que dependen de SHA1, como `diffie-hellman-group-exchange-sha1` y `diffie-hellman-group14-sha1`, manteniendo solo las variantes más seguras basadas en SHA256/SHA512.
- **Algoritmos de Clave de Host del Servidor (Soportados por el Cliente):** C3 anuncia una lista de `server_host_key_algorithms` considerablemente más extensa que C2. Esta lista en C3 incluye soporte para algoritmos de firma relacionados con claves de seguridad de hardware (FIDO/U2F), identificables por el prefijo `sk-` (ej., `sk-ecdsa-sha2-nistp256-cert-v01@openssh.com`), y más variantes de certificados. Este incremento es la razón principal del mayor tamaño del payload del KEXINIT de C3 (1508 bytes) en comparación con el de C2 (1356 bytes).

1.11. Diferencia entre C3 y C4

El paso de C3 (OpenSSH 8.3p1, Ubuntu 20.10) al cliente C4 (OpenSSH 9.0p1, Ubuntu 22.10) revela cambios significativos, alineados con las tendencias más recientes en seguridad SSH:

- **Versión de OpenSSH:** Avance importante a 9.0p1.
- **HASSH:** Cambia (ae8b... para C3, 78c0... para C4).

- **Algoritmos KEX:** El cambio más notable es la introducción y priorización por parte de C4 del algoritmo de intercambio de claves híbrido post-cuántico `sntrup761x2551-9-sha512@openssh.com`. Este movimiento busca proteger las comunicaciones contra futuros ataques de computadoras cuánticas.
- **Algoritmos de Clave de Host del Servidor (Soportados por el Cliente):** C4 elimina el soporte por defecto para `ssh-rsa` (firmas RSA que utilizan el hash SHA-1, considerado débil) de su lista de `server_host_key_algorithms`. Mantiene el soporte para `rsa-sha2-256` y `rsa-sha2-512`.
- **Tamaño del Paquete de Carga Útil KEX:** Como consecuencia del uso del KEX post-cuántico, el paquete donde el cliente C4 envía su material de clave pública efímera tiene un payload SSH de 1204 bytes (ver Figura ??), mucho mayor que los 44 bytes del paquete ECDH Init de C3.

2. Desarrollo (Parte 2)

Identificación y replicación del tráfico de un cliente SSH informante”.

2.1. Identificación del cliente SSH con versión “?”

La Figura 1 (referencia del enunciado) muestra el tráfico del informante. Su paquete de protocolo indica **SSH-2.0-OpenSSH_?** (longitud 85). Paquetes clave: KEXINIT (longitud 1578), ECDH Init (longitud 114), New Keys (longitud 82). Comparando con C1-C4, C3 (OpenSSH 8.3p1) es el más similar en patrón de tamaños. Se infiere una versión similar a OpenSSH 8.3p1, con string de versión ofuscada.

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

Se utilizó C3 para replicar el comportamiento, usando la captura **c3_to_s1.pcap**.

```
tshark -r ./c3_to_s1.pcap -Y "ip.src = 172.19.0.5 && ssh.protocol" -T fields -e frame.number -e frame.len -e ssh.h.protocol
```

Figura 44: Paquete Protocolo C3 para replicación.

Protocolo C3 vs Informante: El cliente C3 anuncia **SSH-2.0-OpenSSH_8.3p1Ubuntu-1ubuntu0.1** (Figura 44). El informante muestra **SSH-2.0-OpenSSH_?** (longitud 85).

```
tshark -r ./c3_to_s1.pcap -Y "ip.src = 172.19.0.5 && ssh.message_code == 20" -T fields -e frame.len -e ssh.packet_length -e ssh.kex_algorithms -e ssh.server_host_key_algorithms -e ssh.encryption_algorithms_client_to_server -e ssh.mac_algorithms_server_to_client -e ssh.mac_algorithms_client_to_server -e ssh.mac_algorithms_server_to_client -e ssh.compression_algorithms_client_to_server -e ssh.compression_algorithms_server_to_client
8      1578      curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,ext-info-c      ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-eddsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-eddsa-sha2-nistp519,sk-ssh-ed25519@openssh.com,ssh-ed25519-cert-v01@openssh.com,sk-ssh-ed25519-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-ed25519-cert-v01@openssh.com,ssh-ed25519-cert-v01@openssh.com,sk-ssh-ed25519@openssh.com,rsa-sha2-512,rsa-sha2-256,ssh-rsa      chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com      chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com      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,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1      umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1      none,zlib@openssh.com,zlib      none,zlib@openssh.com,zlib
```

Figura 45: Paquete KEXINIT C3 para replicación.

KEXINIT C3 vs Informante: El payload del KEXINIT de C3 es de 1508 bytes (Figura 45). El del informante es de 1578 bytes.

```

└$ tshark -r ./c3_to_s1.pcap -Y "frame.number == 11" -O ssh
Frame 11: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
Ethernet II, Src: 02:42:ac:13:00:05 (02:42:ac:13:00:05), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
Internet Protocol Version 4, Src: 172.19.0.5, Dst: 172.19.0.2
Transmission Control Protocol, Src Port: 35580, Dst Port: 22, Seq: 1554, Ack: 1122, Len: 48
SSH Protocol
    SSH Version 2
        Packet Length: 44
        Padding Length: 6
        Key Exchange (method:curve25519-sha256)
            Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
            Last login: ECDH client's ephemeral public key length: 32
            To run a command: ECDH client's ephemeral public key (Q_C): 2c17ef1f2285bf1b2893bb3c903dc6e769fd0bd685e7d94a4e97409086c76a
33
        Padding String: 0000000000000000
        [Sequence number: 1]
        See [Direction: client-to-server]

```

Figura 46: Paquete ECDH Init C3 para replicación.

ECDH Init C3 vs Informante: El frame del ECDH Init de C3 es de 114 bytes (Figura 46), coincidiendo con el informante.

```

└$ tshark -r ./c3_to_s1.pcap -Y "frame.number == 13" -O ssh
Frame 13: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
Ethernet II, Src: 02:42:ac:13:00:05 (02:42:ac:13:00:05), Dst: 02:42:ac:13:00:02 (02:42:ac:13:00:02)
Internet Protocol Version 4, Src: 172.19.0.5, Dst: 172.19.0.2
Transmission Control Protocol, Src Port: 35580, Dst Port: 22, Seq: 1602, Ack: 1718, Len: 16
SSH Protocol
    SSH Version 2 (encryption:chacha20-poly1305@openssh.com mac:<implicit> compression:none)
        Packet Length: 12
        Padding Length: 10
        Key Exchange (method:curve25519-sha256)
            Message Code: New Keys (21)
            Last login: New Keys (21) from 172.19.0.4
            Padding String: 00000000000000000000000000000000
            To run a command: New Keys (21) (use "sudo <command>").
            See [Sequence number: 2]
            See [Direction: client-to-server]

```

Figura 47: Paquete New Keys C3 para replicación.

New Keys C3 vs Informante: El frame del New Keys de C3 es de 82 bytes (Figura 47), coincidiendo con el informante.

Conclusión Replicación: C3 replica bien los tamaños de ECDH Init y New Keys.

3. Desarrollo (Parte 3)

Modificación del servidor S1 para que su KEI sea menor a 300 bytes (ref. Figura 2).

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

```
└$ tshark -r ./c4_to_s1.pcap -Y "tcp.srcport == 22 && ssh.message_code == 20" -T fields -e frame.number -e frame.length -e ssh.packet_length
  9      1166    1076
```

Figura 48: Tamaño KEI inicial del Servidor S1 (Payload SSH: 1076 bytes).

Análisis Inicial: El KEI original del servidor S1 tenía un payload SSH de 1076 bytes (Figura 48).

```
# ----- INICIO MODIFICACIONES LABORATORIO 5 PARTE 3 -----
KexAlgorithms curve25519-sha256@libssh.org
Ciphers chacha20-poly1305@openssh.com
MACs hmac-sha2-256-etm@openssh.com removing packages and contents
Compression no on a system that users do not log into.

HostKeyAlgorithms ssh-ed25519 can run the 'unminimize' command.
# ----- FIN MODIFICACIONES ----- 2025 from ::1
```

Figura 49: Directivas añadidas a `sshd_config` en S1 para reducir KEI.

Modificación de `sshd_config`: En S1, se editó `/etc/ssh/sshd_config` para restringir los algoritmos, como se muestra en la Figura 49. Las directivas específicas fueron: `KexAlgorithms curve25519-sha256@libssh.org`, `Ciphers chacha20-poly1305@openssh.com`, `MACs hmac-sha2-256-etm@openssh.com`, `Compression no`, `HostKeyAlgorithms ssh-ed25519`.

Reinicio y Verificación: Tras verificar sintaxis (`sshd -t`) y reiniciar SSHD, se realizó una nueva captura.

```
└$ tshark -r ./s1_kex_attempt1.pcap -Y "tcp.srcport == 22 && ssh.message_code == 20" -T fields -e frame.number -e frame.length -e ssh.packet_length
  9      326    236
```

Figura 50: Tamaño del KEI del Servidor S1 modificado (Payload SSH: 236 bytes).

El nuevo KEI del servidor fue de **236 bytes** (Figura 50), cumpliendo el requisito.

```

└$ tshark -r ./s1_kex_attempt1.pcap -Y "tcp.srcport = 22 && ssh.message_code = 20" -o ssh
Frame 9: 326 bytes on wire (2608 bits), 326 bytes captured (2608 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 [x00_04]
Transmission Control Protocol, Src Port: 22, Dst Port: 39400, Seq: 42, Ack: 1546, Len: 240
SSH Protocol
  * SSH Version 2 https://landscape.canonical.com
  * Support Packet Length: 236 https://ubuntu.com/advantage
  * Padding Length: 10
  * This system minimized by removing packages and content that are
    not required.
  * Message Code: Key Exchange Init (20)
  * Algorithms
    To restore this: Cookie: 15b6254aba33752d55a378388eabd2a9
    Last login: Tue May 28 17:21:19 2019
    * kex_algorithms length: 28
    * server_host_key_algorithms length: 11
    * server_host_key_algorithms string: ssh-ed25519
    * encryption_algorithms_client_to_server length: 29
    * encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com
    * encryption_algorithms_server_to_client length: 29
    * encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com
    * mac_algorithms_client_to_server length: 29
    * mac_algorithms_client_to_server string: hmac-sha2-256-etm@openssh.com
    * mac_algorithms_server_to_client length: 29
    * mac_algorithms_server_to_client string: hmac-sha2-256-etm@openssh.com
    * mac_algorithms_server_to_client length: 29
    * mac_algorithms_server_to_client string: hmac-sha2-256-etm@openssh.com
    * compression_algorithms_client_to_server length: 4
    * compression_algorithms_client_to_server string: none
    * compression_algorithms_server_to_client length: 4
    * compression_algorithms_server_to_client string: none
    * Management: compression_algorithms_server_to_client length: 4
    * Management: compression_algorithms_server_to_client string: none
    * Languages: languages_client_to_server length: 0
    * Languages: languages_client_to_server string:
    * This system has languages_server_to_client length: 0 and content that are
      not required on languages_server_to_client string:
    * First KEX Packet Follows: 0
    * Reserved: 00000000
    To restore this: [hashServerAlgorithms: curve25519-sha256@libssh.org;chacha20-poly1305@openssh.com;hmac-sha2-256-etm
    @openssh.com;none] administrator (use sudo <command>).
    See "man sudo" [hashServer: f45ac7a93f99c1ccb786123d62948cb]
    * Padding String: 00000000000000000000000000000000
    * Sequence number: 0
    * Direction: server-to-client

```

Figura 51: Contenido del KEI del Servidor S1 modificado, mostrando listas de algoritmos reducidas.

La Figura 51 detalla el contenido del KEI modificado, evidenciando la reducción en los algoritmos anunciados.

4. Desarrollo (Parte 4)

Análisis del protocolo OpenSSH y su alineación con los principios de seguridad de la información.

4.1. Explicación OpenSSH en general

OpenSSH es la implementación de código abierto de referencia del protocolo Secure Shell (SSH) versión 2. Su objetivo es proveer un canal seguro sobre redes inseguras para servicios como acceso remoto (shell), transferencia de archivos (SFTP, SCP) y tunelización. Opera con una arquitectura cliente-servidor y se estructura en tres capas principales:

- **Capa de Transporte (SSH-TRANS):** Establece la conexión segura inicial. Incluye intercambio de versiones (ej. Figura 9), autenticación del servidor, intercambio de claves (KEX) para generar secretos compartidos (ej. Figura 11), y negociación de algoritmos de cifrado, MAC y compresión. Paquetes KEXINIT (ej. Figura 10) son clave aquí. NEWKEYS (ej. Figura 12) marca la transición a cifrado.
- **Capa de Autenticación de Usuario (SSH-USERAUTH):** Autentica al cliente ante el servidor sobre el canal seguro. Soporta métodos como contraseña o clave pública. Paquetes cifrados (ej. Figuras 13, 14) corresponden a este proceso.
- **Capa de Conexión (SSH-CONN):** Permite multiplexar múltiples canales lógicos (shells, reenvío de puertos) sobre la conexión segura y autenticada.

4.2. Capas de Seguridad en OpenSSH y Principios de la Información

- **Confidencialidad:** Garantizada por cifrado simétrico (ej., `chacha20-poly1305@openssh.com`) de los datos tras el KEX. Evidente en los paquetes KEXINIT y los “^Enrypt” posteriores.
- **Integridad:** Asegurada por Códigos de Autenticación de Mensajes (MAC) (ej., `hmac-sha2-256-etm@openssh.com`) negociados en KEXINIT. Cifrados AEAD como ChaCha20-Poly1305 integran confidencialidad e integridad.
- **Autenticidad:**
 - **Servidor:** Probada por el servidor usando su clave de host durante el KEX; el cliente verifica contra `known_hosts`. Algoritmos como `ssh-ed25519` se negocian.
 - **Usuario:** El servidor verifica al usuario mediante métodos como contraseña (evidenciado por paquetes cifrados post-NEWKEYS).
- **Disponibilidad:** SSH no la garantiza por sí mismo, pero la facilita mediante administración remota. El establecimiento de conexiones en el laboratorio prueba su disponibilidad momentánea.

- **No Repudio:** Limitado. Fuerte para autenticación de cliente con clave pública. Para acciones dentro de la sesión, no hay firmas por paquete, limitando el no repudio.

4.3. Identificación de qué principios no se cumplen (o se cumplen limitadamente)

- **Disponibilidad:** SSH depende de la infraestructura; no la provee intrínsecamente.
- **No Repudio:** Fuerte para autenticación de cliente con clave pública, pero limitado para acciones dentro de la sesión, ya que los datos no se firman individualmente.

El análisis del tráfico interceptado, observando la negociación de algoritmos (ej., la preferencia por Curve25519 y ChaCha20-Poly1305) y el cifrado del canal, permite comprender estas implementaciones.

Conclusiones y comentarios

Este laboratorio ha proporcionado una experiencia práctica invaluable sobre el funcionamiento interno del protocolo SSH y la importancia de la negociación de algoritmos criptográficos. La utilización de Docker facilitó la creación de un entorno controlado con múltiples versiones de clientes OpenSSH, permitiendo observar directamente cómo evoluciona el protocolo y sus preferencias de seguridad.

La Parte 1 del desarrollo, centrada en el análisis del tráfico, reveló cómo cada versión de OpenSSH (desde la 7.3p1 en C1 hasta la 9.0p1 en C4) presenta una huella digital (HASSH) única, resultado directo de los diferentes conjuntos de algoritmos que anuncia en su paquete KEXINIT. Fue particularmente interesante notar la progresiva eliminación de algoritmos considerados más débiles (como ciertos cifrados CBC o KEX basados en SHA1) y la adopción de opciones más robustas y modernas, como el intercambio de claves post-cuántico `sntrup76_1x25519-sha512@openssh.com` y la depreciación de `ssh-rsa` por defecto en OpenSSH 9.0p1. La comparación con bases de datos HASSH públicas, aunque no siempre resultó en coincidencias exactas para todas las versiones, sirvió para validar las identificaciones y comprender que estas bases son herramientas útiles pero no siempre exhaustivas.

En la Parte 2, la tarea de identificar un cliente informante en una versión obfuscada (`openSSH_?`) a partir de una imagen de tráfico resaltó la importancia del análisis forense de los metadatos de la comunicación, como las longitudes de los paquetes del handshake. Se determinó que C3 (OpenSSH 8.3p1) era el análogo más cercano, y se discutió la complejidad de replicar exactamente una string de protocolo modificada sin recurrir a la recompilación del software cliente.

La Parte 3 fue un ejercicio práctico en la configuración del lado del servidor. Se demostró exitosamente cómo, al restringir deliberadamente los algoritmos anunciados por el servidor S1 en su archivo `sshd_config`, es posible reducir significativamente el tamaño de su paquete `ServerKeyExchangeInit`, cumpliendo con el requisito de laboratorio de un tamaño inferior a 300 bytes (logrando 236 bytes). Esto no solo ilustra la flexibilidad de OpenSSH, sino

también las implicaciones que tales configuraciones pueden tener en la compatibilidad y la superficie de ataque.

Finalmente, la Parte 4 permitió articular la conexión entre las observaciones prácticas del tráfico y los principios teóricos de la seguridad de la información. Se pudo argumentar, con base en los paquetes interceptados, cómo SSH aborda eficazmente la confidencialidad, la integridad y la autenticidad, mientras que la disponibilidad y el no repudio se cumplen de manera más indirecta o con ciertas limitaciones inherentes al diseño del protocolo.

En conclusión, este laboratorio refuerza la idea de que la seguridad no es estática; los protocolos y sus implementaciones evolucionan para hacer frente a nuevas amenazas y mejorar las prácticas. El análisis del tráfico de red es una herramienta fundamental para comprender y auditar estos mecanismos de seguridad.