

Informe Laboratorio 5

Sección 1

Ignacio Santiago Medina Diaz
e-mail: ignacio.medina1@mail.udp.cl

Noviembre de 2024

Índice

Descripción de actividades	3
1. Desarrollo (Parte 1)	5
1.1. Códigos de cada Dockerfile	5
1.1.1. C1	6
1.1.2. C2	7
1.1.3. C3	9
1.1.4. C4/S1	10
1.2. Creación de las credenciales para S1	12
1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	14
1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	23
1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	30
1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	35
1.7. Compara la versión de HASSH obtenida con la base de datos para validar si el cliente corresponde al mismo	39
1.8. Tipo de información contenida en cada uno de los paquetes generados en texto plano	39
1.8.1. C1	40
1.8.2. C2	41
1.8.3. C3	42
1.8.4. C4/S1	43
1.9. Diferencia entre C1 y C2	44
1.10. Diferencia entre C2 y C3	44
1.11. Diferencia entre C3 y C4	44
2. Desarrollo (Parte 2)	45
2.1. Identificación del cliente SSH con versión “?”	45
2.2. Replicación de tráfico al servidor (paso por paso)	45
3. Desarrollo (Parte 3)	51
3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso)	51
4. Desarrollo (Parte 4)	55
4.1. Explicación OpenSSH en general	55
4.2. Capas de Seguridad en OpenSSH	55
4.3. Identificación de que protocolos no se cumplen	57

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. Cada HASSH deberá compararlo con la base de datos HASSH disponible en el módulo de TLS, e identificar si el hash obtenido corresponde a la misma versión de su cliente.

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

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

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

Figura 1: Tráfico generado del informante

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

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

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

Figura 2: Captura del Key Exchange

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

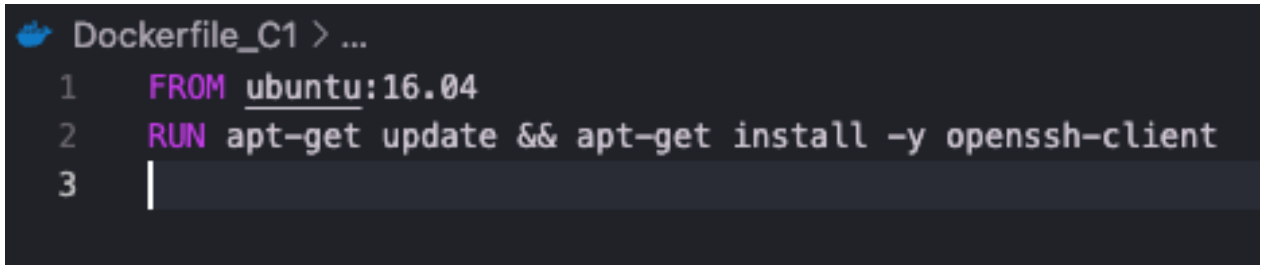
1. Desarrollo (Parte 1)

1.1. Códigos de cada Dockerfile

Para dar inicio al último laboratorio, se comenzará explicando cada uno de los dockerfiles de cada contenedor para la realización de esta actividad. Cabe destacar, que se utilizó Docker Desktop en un sistema operativo MacOS.

1.1.1. C1

Para el primer contenedor se utilizó el siguiente Dockerfile.



```
Dockerfile_C1 > ...  
1 FROM ubuntu:16.04  
2 RUN apt-get update && apt-get install -y openssh-client  
3 |
```

Figura 3: Dockerfile C1.

Tal como se aprecia en la figura 3, se utilizó Ubuntu 16.04 (Xenial Xerus) a diferencia de la versión 16:10, esto se debe que es una versión descontinuada, por lo tanto se utilizaron versiones LTS (que reciben soporte extendido).

Por otro lado, en la segunda línea de código, se ejecutan los siguientes comandos:

- `apt-get update`: Actualiza la lista de paquetes disponibles en los repositorios configurados, asegurando que las versiones más recientes estén disponibles.
- `apt-get install -y openssh-client`: Instala el paquete `openssh-client`, que proporciona las herramientas necesarias para utilizar el protocolo SSH desde el cliente. La opción `-y` se utiliza para evitar la necesidad de confirmar manualmente la instalación de los paquetes, lo que es útil en un entorno automatizado como Docker.

En resumen, este Dockerfile crea una imagen mínima basada en Ubuntu 16.04 que incluye el cliente de SSH.

Una vez creado el dockerfile, se comenzó con la creación y ejecución del contenedor en Docker.

```
$ docker build -f Dockerfile_C1 -t c1 .
[+] Building 16.3s (6/6) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile_C1           0.0s
=> => transferring dockerfile: 114B                               0.0s
=> [internal] load metadata for docker.io/library/ubuntu:16.04  1.3s
=> [internal] load .dockerignore                                 0.0s
=> => transferring context: 2B                                     0.0s
=> [1/2] FROM docker.io/library/ubuntu:16.04@sha256:1f1a2d56d604801a9671f301190704c25d604a416f59e03 2.1s
=> => resolve docker.io/library/ubuntu:16.04@sha256:1f1a2d56d604801a9671f301190704c25d604a416f59e03 0.0s
=> => sha256:66927c6d1d3d2e9321c4893f7f2105b7cd23dfb082853d97ec08f188e271e612 854B / 854B 0.3s
=> => sha256:000560be91651dcbf476ebac8b8bf1f1339694a3327f8e6da2519e0b29b33eb5d 479B / 479B 0.4s
=> => sha256:1f1a2d56d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d6 1.42kB / 1.42kB 0.0s
=> => sha256:f4c51ba054967fd4b06715f1b67078efbe9ca152e8be98d8f3c1f4d08c6042f8 1.15kB / 1.15kB 0.0s
=> => sha256:d125c6a1fe22504f84552b7eb11e353e88691875c18caf68847843892b190ccc 3.38kB / 3.38kB 0.0s
=> => sha256:828b35a09f0b2f3d1dead43aa2468ff5eba6c463423b3fff7ee6d150f6fd1b6b 41.24MB / 41.24MB 0.8s
=> => sha256:6225a0253717abdc2ee23ea211c1c439c93b84231ec0a4f1c74762a205ba7234 173B / 173B 0.5s
=> => extracting sha256:828b35a09f0b2f3d1dead43aa2468ff5eba6c463423b3fff7ee6d150f6fd1b6b 1.2s
=> => extracting sha256:66927c6d1d3d2e9321c4893f7f2105b7cd23dfb082853d97ec08f188e271e612 0.0s
=> => extracting sha256:000560be91651dcbf476ebac8b8bf1f1339694a3327f8e6da2519e0b29b33eb5d 0.0s
=> => extracting sha256:6225a0253717abdc2ee23ea211c1c439c93b84231ec0a4f1c74762a205ba7234 0.0s
=> [2/2] RUN apt-get update && apt-get install -y openssh-client 12.8s
=> => exporting to image                                           0.1s
=> => exporting layers                                             0.1s
=> => writing image sha256:a5c8661947da5ad7648f10e8d58efa36a9fecc160752cdc4ae8302375ab7e707 0.0s
=> => naming to docker.io/library/c1                               0.0s

What's next:
View a summary of image vulnerabilities and recommendations -> docker scout quickview
(base)
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:03:25]
$
```

Figura 4: Creación del contenedor C1.

```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:02:14]
$ docker run -dit --name C1 c1

5e6dd025f71cd7351df0f299907a7e096527c8debed6c022ab535ccfda99df0b
(base)
```

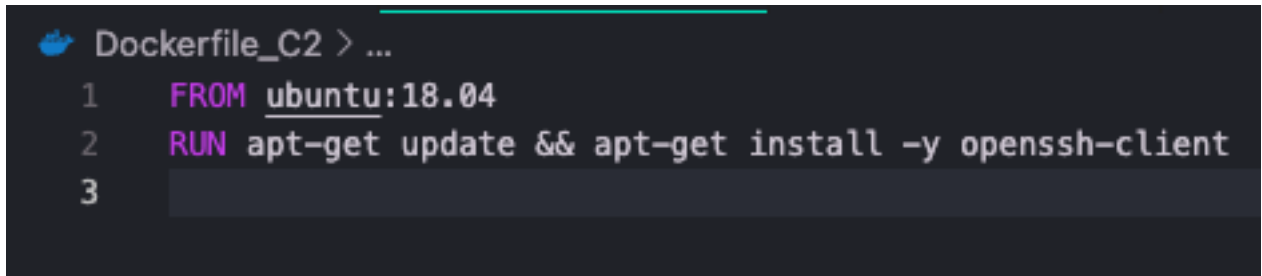
Figura 5: Ejecución del contenedor C1.

1.1.2. C2

Se utilizó la imagen base **Ubuntu 18.04** (Bionic Beaver), no se utilizó la versión 18.10, ya que, de la misma forma que la anterior, es una versión discontinuada.

A continuación se ejecuta dos pasos consecutivos dentro de la imagen:

- `apt-get update`: Actualiza la lista de índices de paquetes desde los repositorios configurados, asegurando que la información de los paquetes sea la más reciente.
- `apt-get install -y openssh-client`: Instala el paquete `openssh-client`, que proporciona herramientas para establecer conexiones SSH seguras desde el contenedor hacia otros servidores. La opción `-y` automatiza la aceptación de cualquier solicitud de

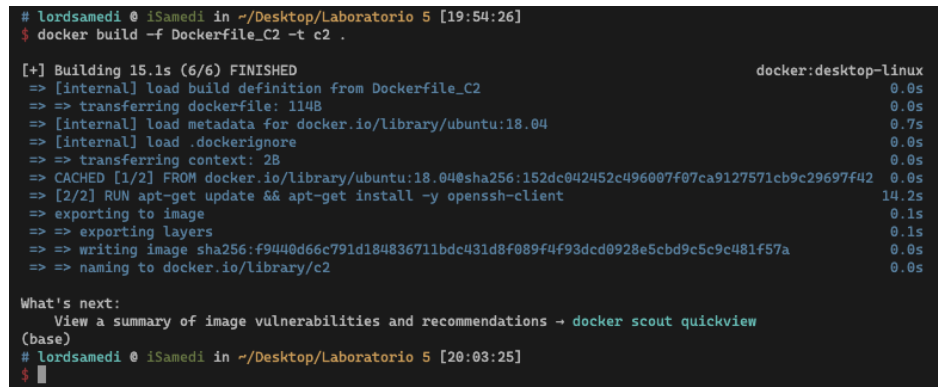


```
Dockerfile_C2 > ...  
1 FROM ubuntu:18.04  
2 RUN apt-get update && apt-get install -y openssh-client  
3
```

Figura 6: Dockerfile C2.

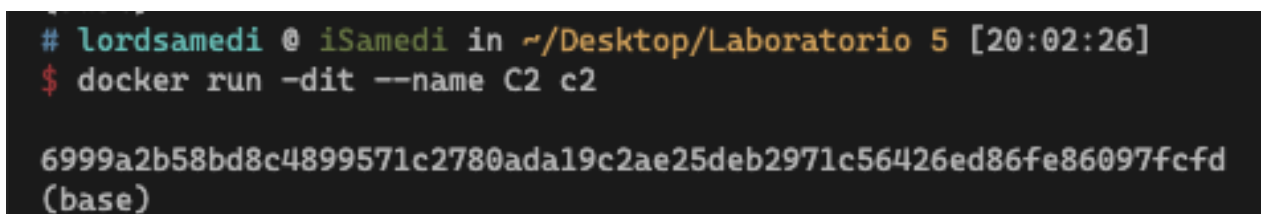
confirmación durante la instalación, lo cual es fundamental en scripts no interactivos como un Dockerfile.

Finalmente, se realizó la creación y ejecución del contenedor Docker para el dockerfile C2.



```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [19:54:26]  
$ docker build -f Dockerfile_C2 -t c2 .  
  
[+] Building 15.1s (6/6) FINISHED                                docker:desktop-linux  
=> [internal] load build definition from Dockerfile_C2           0.0s  
=> => transferring dockerfile: 114B                               0.0s  
=> [internal] load metadata for docker.io/library/ubuntu:18.04  0.7s  
=> [internal] load .dockerignore                                  0.0s  
=> => transferring context: 2B                                       0.0s  
=> CACHED [1/2] FROM docker.io/library/ubuntu:18.04@sha256:152dc042452c496007f07ca9127571cb9c29697f42 0.0s  
=> [2/2] RUN apt-get update && apt-get install -y openssh-client 14.2s  
=> exporting to image                                             0.1s  
=> => exporting layers                                              0.1s  
=> => writing image sha256:f9440d66c791d184836711bdc431d8f089f4f93dcd0928e5cbd9c5c9c481f57a 0.0s  
=> => naming to docker.io/library/c2                                0.0s  
  
What's next:  
View a summary of image vulnerabilities and recommendations → docker scout quickview  
(base)  
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:03:25]  
$
```

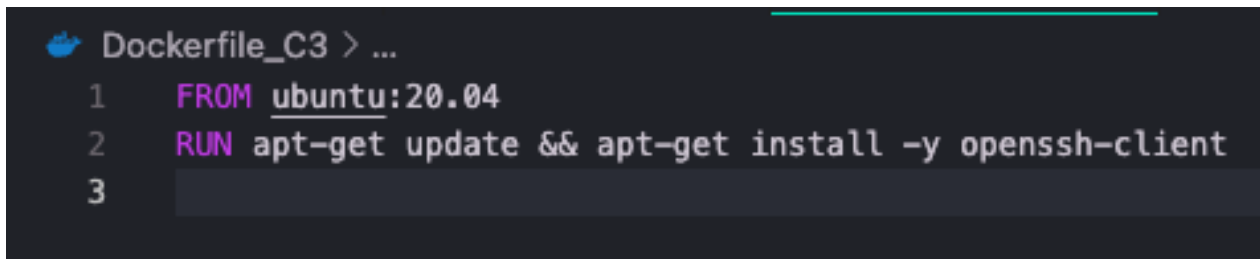
Figura 7: Creación del contenedor C2.



```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:02:26]  
$ docker run -dit --name C2 c2  
  
6999a2b58bd8c4899571c2780ada19c2ae25deb2971c56426ed86fe86097fcfd  
(base)
```

Figura 8: Ejecución del contenedor C2.

1.1.3. C3



```

Dockerfile_C3 > ...
1 FROM ubuntu:20.04
2 RUN apt-get update && apt-get install -y openssh-client
3

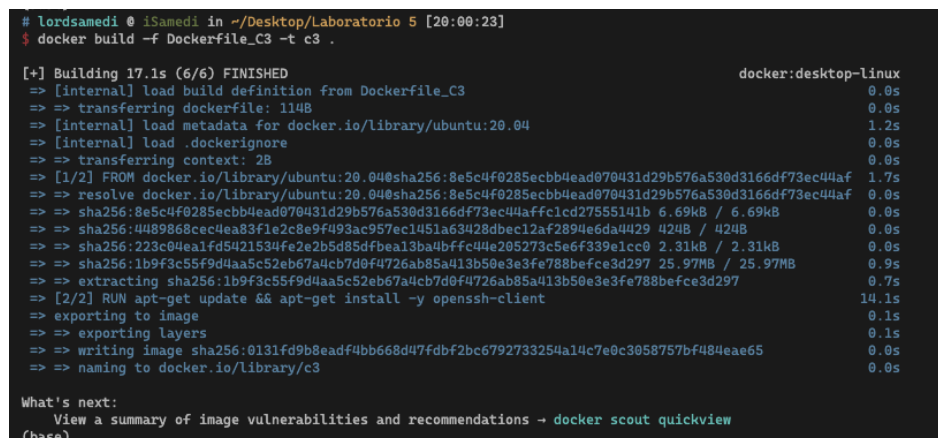
```

Figura 9: Dockerfile C3.

Este **Dockerfile** crea una imagen basada en **Ubuntu 20.04** y realiza las siguientes acciones:

1. **Base Image (FROM ubuntu:20.04):** Define que la imagen base será Ubuntu 20.04 (Focal Fossa), que incluye mejoras en estabilidad, seguridad y soporte de paquetes más modernos.
2. **Actualizar e instalar paquetes:**
 - **apt-get update:** Actualiza la lista de paquetes para asegurarse de trabajar con las versiones más recientes.
 - **apt-get install -y openssh-client:** Instala el cliente SSH, necesario para realizar conexiones seguras a otros servidores. El uso de **-y** permite que el proceso sea automático sin pedir confirmación.

Finalmente, se realizó la creación y ejecución del contenedor Docker para el dockerfile C3.



```

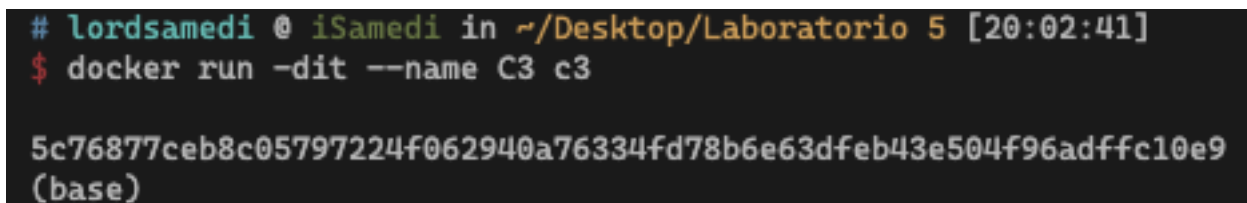
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:00:23]
$ docker build -f Dockerfile_C3 -t c3 .

[+] Building 17.1s (6/6) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile_C3           0.0s
=> => transferring dockerfile: 114B                               0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04  1.2s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [1/2] FROM docker.io/library/ubuntu:20.04@sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44af  1.7s
=> => resolve docker.io/library/ubuntu:20.04@sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44af  0.0s
=> => sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44affc1cd27555141b  6.69kB / 6.69kB  0.0s
=> => sha256:4489868cec4ea83f1e2c8e9f493ac957ec1451a63428dbec12af2894e6da4429  424B / 424B  0.0s
=> => sha256:223c04ealfd5421534fe2e2b5d85dfbea13ba4bffc44e205273c5e6f339e1cc0  2.31kB / 2.31kB  0.0s
=> => sha256:1b9f3c55f9d4aa5c52eb67a4cb7d0f4726ab85a413b50e3e3fe788befce3d297  25.97MB / 25.97MB  0.9s
=> => extracting sha256:1b9f3c55f9d4aa5c52eb67a4cb7d0f4726ab85a413b50e3e3fe788befce3d297  0.7s
=> [2/2] RUN apt-get update && apt-get install -y openssh-client 14.1s
=> exporting to image                                           0.1s
=> => exporting layers                                           0.1s
=> => writing image sha256:0131fd9b8eadf4bb668d47f4b2bc6792733254a14c7e0c3058757bf484eae65  0.0s
=> => naming to docker.io/library/c3                             0.0s

What's next:
View a summary of image vulnerabilities and recommendations -> docker scout quickview
(base)

```

Figura 10: Creación del contenedor C3.



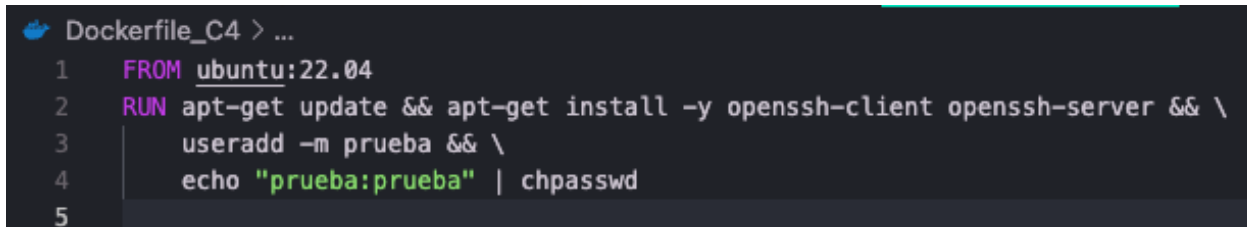
```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:02:41]
$ docker run -dit --name C3 c3

5c76877ceb8c05797224f062940a76334fd78b6e63dfefb43e504f96adffc10e9
(base)
```

Figura 11: Ejecución del contenedor C3.

1.1.4. C4/S1

Finalmente, el cuarto **Dockerfile** crea una imagen basada en **Ubuntu 22.04** con configuraciones adicionales relacionadas con SSH y un usuario personalizado. A continuación, se explica paso a paso:



```
Dockerfile_C4 > ...
1 FROM ubuntu:22.04
2 RUN apt-get update && apt-get install -y openssh-client openssh-server && \
3     useradd -m prueba && \
4     echo "prueba:prueba" | chpasswd
5
```

Figura 12: Dockerfile C4/S1.

1. Se utiliza **Ubuntu 22.04** (Jammy Jellyfish) como base, ofreciendo un sistema operativo moderno con soporte extendido y características actualizadas.
2. `apt-get update`: Actualiza la lista de paquetes disponibles.
3. `apt-get install -y openssh-client openssh-server`: Instala tanto el cliente como el servidor SSH, lo que permite al contenedor tanto iniciar conexiones SSH como aceptar conexiones entrantes.

A diferencia de los contenedores anteriores, en este contenedor se crea un usuario llamado 'prueba', de la siguiente forma:

- `useradd -m prueba`: De esta forma se crea un nuevo usuario llamado **prueba** y genera automáticamente su directorio home.
- `echo "prueba:prueba" | chpasswd`: Define la contraseña del usuario **prueba** como **prueba**. Esto permite al usuario autenticarse al conectarse al servidor SSH del contenedor.

En resumen, el propósito de este Dockerfile, es configurar un contenedor basado en Ubuntu 22.04 con un entorno SSH funcional. Además, agrega un usuario personalizado con acceso

mediante contraseña. Con el fin de realizar pruebas de conexiones SSH y para simular un servidor SSH dentro de un contenedor.

Finalmente, de igual forma que en los casos anteriores, se realizó la creación y ejecución del contenedor Docker para el dockerfile C4/S1.

```
# lordsamedí @ iSamedí in ~/Desktop/Laboratorio 5 [20:04:40]
$ docker build -f Dockerfile_C4 -t c4 .

[+] Building 35.1s (7/7) FINISHED
=> [internal] load build definition from Dockerfile_C4
=> => transferring dockerfile: 200B
=> [internal] load metadata for docker.io/library/ubuntu:22.04
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/2] FROM docker.io/library/ubuntu:22.04@sha256:0e5e8a57c2499209aafc3b40fcd541e9a456aab7296681a3994d631587203f97
=> [2/2] RUN apt-get update && apt-get install -y openssh-client openssh-server && useradd -m prueba && echo "prueba:prueba" | chpasswd
=> => exporting to image
=> => exporting layers
=> => writing image sha256:fc46dcdd484ae311eb72ec4b6103d8f8e88cabeab8eda93f0576716555b3d87
=> => naming to docker.io/library/c4

What's next:
View a summary of image vulnerabilities and recommendations - docker scout quickview
(base)
# lordsamedí @ iSamedí in ~/Desktop/Laboratorio 5 [20:05:19]
$
```

Figura 13: Creación del contenedor C4/S1.

```
(base)
# lordsamedí @ iSamedí in ~/Desktop/Laboratorio 5 [20:03:25] C:125
$ docker run -dit --name C4 c4

c9228708f4be7e31fda6db695f665bbb46ec90e1bdf65924c1f521e2f85fe1b12
(base)
# lordsamedí @ iSamedí in ~/Desktop/Laboratorio 5 [20:05:59]
$
```

Figura 14: Ejecución del contenedor C4.

A continuación se mostrará los contenedores en ejecución a través de la aplicación Docker Desktop, comprobando la correcta ejecución de los contenedores anteriores.

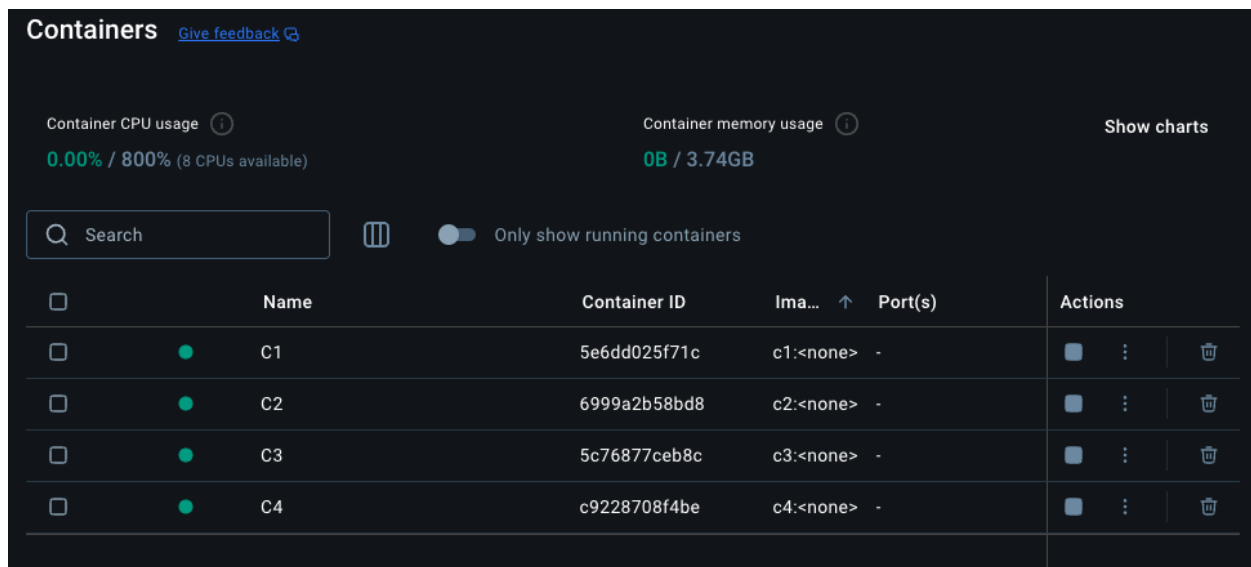


Figura 15: Correcta ejecución de los contenedores.

1.2. Creación de las credenciales para S1

Las credenciales para el servidor OpenSSH o equipo S1, se realizó a través del Dockerfile mencionado anteriormente (C4/S1), creando un usuario llamado **prueba** con su directorio personal asignándole la contraseña **prueba**. Esto permite que el contenedor actúe como un servidor SSH, listo para aceptar conexiones y con un usuario configurado para pruebas o tareas específicas.

Una vez creada las credenciales, se realizó la ejecución del servicio SSH con el siguiente comando:

- **service ssh start:** Este comando pone en funcionamiento el servicio SSH (**ssh**), lo que permite al contenedor aceptar conexiones SSH entrantes, tal como se mostrará en la siguiente figura.

```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:05:19]
$ docker exec -it C4 bash

root@c9228708f4be:/# service ssh start
* Starting OpenBSD Secure Shell server sshd
root@c9228708f4be:/#
```

Figura 16: Ejecución servicio SSH.

Posteriormente, para proceder correctamente con los pasos siguientes, se extraerá la dirección IP que le ha sido asignada al contenedor C4/S1 dentro de la red Docker, utilizando el siguiente comando:

- `docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' C4`

```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:11:04]
$ docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' C4

172.17.0.5
(base)
```

Figura 17: Extraer dirección IP del contenedor C4.

Tal como se aprecia en la figura 17, el comando muestra que el contenedor C4 tiene la dirección IP **172.17.0.5** en su red Docker. Esta dirección pertenece al rango predeterminado utilizado por el controlador de red "bridge" de Docker. La dirección IP servirá a continuación para generar tráfico entre contenedores a través de OpenSSH Client to Server.

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

Antes de proceder a realizar el trafico generado, se instalo **tcpdump** en el contenedor 4 para poder realizar una captura directa en el contenedor del servidor OpenSSH. Una vez

```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:59:51] C:1
$ docker exec -it C4 bash

root@c9228708f4be:/# apt-get update && apt-get install -y tcpdump
Hit:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease
Get:2 http://ports.ubuntu.com/ubuntu-ports jammy-updates InRelease [128 kB]
Hit:3 http://ports.ubuntu.com/ubuntu-ports jammy-backports InRelease
Get:4 http://ports.ubuntu.com/ubuntu-ports jammy-security InRelease [129 kB]
Fetched 257 kB in 2s (108 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libpcap0.8
Suggested packages:
```

Figura 18: Instalación de tcpdump.

instalada, se procede a realizar la captura de trafico dentro del contenedor C4 para cada uno de los contenedores. En esta sección se creará un archivo llamado `captura.pcap` para capturar el trafico de los paquetes del flujo generados por el cliente y el contenido asociado a cada uno de ellos

```
root@c9228708f4be:/# tcpdump -i eth0 port 22 -w captura.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C39 packets captured
39 packets received by filter
0 packets dropped by kernel
root@c9228708f4be:/# █
```

Figura 19: Captura de trafico para el contenedor C1.

Una vez iniciado la captura de trafico dentro del servidor, se realizo la conexión desde el cliente C1 para conectarse a S1(C4) usando SSH, utilizando el siguiente comando marcado con un *recuadro color rojo*.

```
root@5e6dd025f71c:/# ssh prueba@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.
ECDSA key fingerprint is SHA256:9NqkxSuEA77IuNIb2F6pT+305pvXa4ytXu00i08Re4s.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.17.0.5' (ECDSA) to the list of known hosts.
prueba@172.17.0.5's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.10.4-linuxkit aarch64)

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

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

To restore this content, you can run the 'unminimize' command.

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 20: Conexión desde Cliente (C1) a S1.

Tal como se aprecia en el *recuadro rojo*, indica que se quiere establecer una conexión SSH para conectarse al servidor del usuario prueba de C4 con la dirección IP anteriormente obtenida del contenedor C4/S1.

Una vez ejecutado el comando anterior, el servidor solicita la contraseña del usuario prueba (*recuadro amarillo*). Esto sirve para autenticar el ingreso al servidor. Una vez ingresado la contraseña correcta, siendo esta **prueba**, se observa mensajes de bienvenida logrando un correcto inicio de sesión al servidor S1 desde el cliente.

Una vez establecida la conexión, se detiene la captura de tráfico en el contenedor C4 presionando la tecla **CONTROL + C**, guardando la captura dentro de los archivos del docker.

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

1 DESARROLLO (PARTE 1)

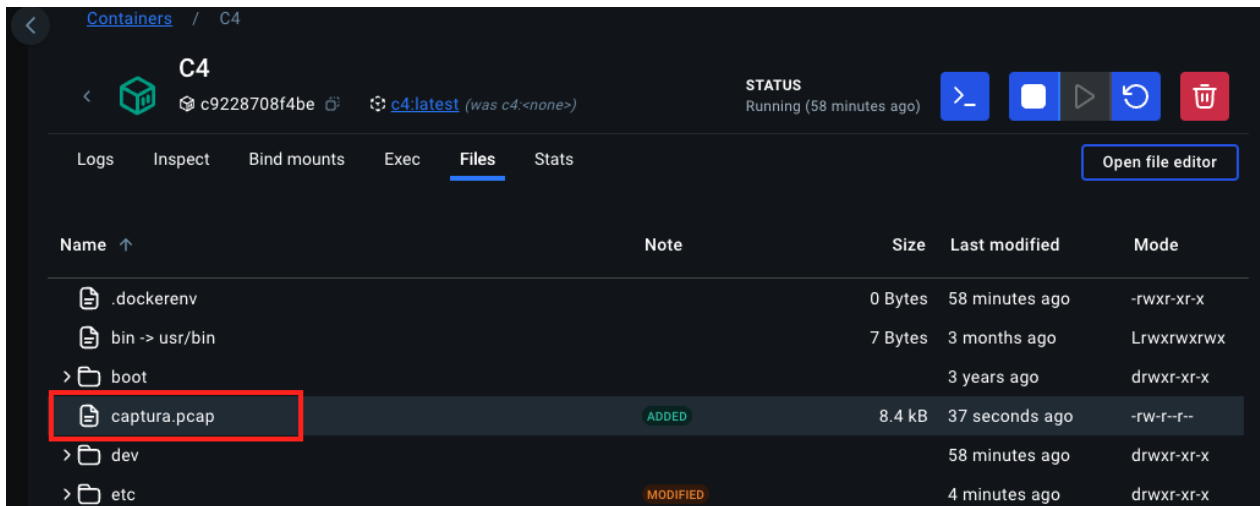


Figura 21: Captura C1 guardada en Docker.

La captura se descarga y se guarda en la carpeta personal en donde se esta llevando a cabo el laboratorio. Una vez descargada se ejecuto con Wireshark, obteniendo el siguiente trafico de capturas.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.17.0.3	172.17.0.5	TCP	74	52562 -> ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495
2	0.000099	172.17.0.5	172.17.0.3	TCP	66	52562 -> 52562 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0
3	0.000159	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=1 Win=65536 Len=0 TSv
4	0.000897	172.17.0.3	172.17.0.5	SSHv2	108	Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu
5	0.000907	172.17.0.5	172.17.0.3	TCP	66	52562 -> 52562 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSv
6	0.020354	172.17.0.3	172.17.0.5	SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu
7	0.020413	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSv
8	0.022273	172.17.0.5	172.17.0.3	SSHv2	1178	Server: Key Exchange Init
9	0.022297	172.17.0.3	172.17.0.5	SSHv2	1402	Client: Key Exchange Init
10	0.067769	172.17.0.5	172.17.0.3	TCP	66	ssh(22) -> 52562 [ACK] Seq=1155 Ack=1379 Win=65536 Len=0
11	0.067805	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=1379 Ack=1155 Win=65536 Len=0
12	0.067833	172.17.0.3	172.17.0.5	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Ini
13	0.067862	172.17.0.5	172.17.0.3	TCP	66	ssh(22) -> 52562 [ACK] Seq=1155 Ack=1427 Win=65536 Len=0
14	0.073793	172.17.0.5	172.17.0.3	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Req
15	0.073834	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=1427 Ack=1751 Win=65536 Len=0
16	0.080721	172.17.0.3	172.17.0.5	SSHv2	82	Client: New Keys
17	0.121950	172.17.0.5	172.17.0.3	TCP	66	ssh(22) -> 52562 [ACK] Seq=1751 Ack=1443 Win=65536 Len=0
18	0.122839	172.17.0.3	172.17.0.5	SSHv2	110	Client: Encrypted packet (len=44)
19	0.122861	172.17.0.5	172.17.0.3	TCP	66	ssh(22) -> 52562 [ACK] Seq=1751 Ack=1487 Win=65536 Len=0
20	0.122183	172.17.0.5	172.17.0.3	SSHv2	110	Server: Encrypted packet (len=44)
21	0.122285	172.17.0.3	172.17.0.5	SSHv2	134	Client: Encrypted packet (len=68)
22	0.128694	172.17.0.5	172.17.0.3	SSHv2	118	Server: Encrypted packet (len=52)
23	0.170216	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=1555 Ack=1847 Win=65536 Len=0
24	3.095711	172.17.0.3	172.17.0.5	SSHv2	214	Client: Encrypted packet (len=148)
25	3.136673	172.17.0.5	172.17.0.3	TCP	66	ssh(22) -> 52562 [ACK] Seq=1847 Ack=1703 Win=65536 Len=0
26	3.133726	172.17.0.5	172.17.0.3	SSHv2	94	Server: Encrypted packet (len=28)
27	3.183842	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=1703 Ack=1875 Win=65536 Len=0
28	3.184138	172.17.0.3	172.17.0.5	SSHv2	178	Client: Encrypted packet (len=112)
29	3.184149	172.17.0.5	172.17.0.3	TCP	66	ssh(22) -> 52562 [ACK] Seq=1875 Ack=1815 Win=65536 Len=0
30	3.199251	172.17.0.5	172.17.0.3	SSHv2	694	Server: Encrypted packet (len=628)
31	3.248733	172.17.0.3	172.17.0.5	TCP	66	52562 -> ssh(22) [ACK] Seq=1815 Ack=2503 Win=65536 Len=0
32	3.248747	172.17.0.5	172.17.0.3	SSHv2	110	Server: Encrypted packet (len=41)

Figura 22: Capturas del trafico de C1.

Tal como se parecia en la figura anterior, el sistema arranca con paquetes SYN, SYN-ACK y ACK, siendo este el saludo a tres vías, tal como se aprecia en la siguiente imagen.

Length	Info
74	52562 → ssh(22) [SYN] Seq=0
74	ssh(22) → 52562 [SYN, ACK] S
66	52562 → ssh(22) [ACK] Seq=1

Figura 23: Inicio de los paquetes.

En la información del Source es quien inicia la conexión es el equipo **cliente con IP 172.17.0.3** dirigido al destination quien es el **servidor IP 172.17.0.5**.

SSHv2	108	Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10)
TCB	66	ssh(22) → 52562 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=323

Figura 24: Identificación del cliente.

Lo primero que hace el cliente es identificarse con un tamaño de 108 bytes. Si examinamos el SSH Protocol, nos encontraremos con lo siguiente.

SSH Protocol
Protocol: SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10
[Direction: client-to-server]

Figura 25: SSH Protocol Client.

En el contenido del protocolo de la figura 25, nos indica que es un **protocolo SSH 2.0** y aparte de eso nos da información de la versión del sistema operativo, que en este caso es **Ubuntu-4ubuntu2.10** en el cual se esta corriendo, con una dirección de cliente a servidor. Luego el paquete siguiente es un ACK, que significa que el servidor esta reconociendo el paquete del servidor.

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

1 DESARROLLO (PARTE 1)

TCP	66	52562 → ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSval=43
SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)
TCP	66	52562 → ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSval=43

Figura 26: Identificación del servidor.

Lo siguiente es la identificación del servidor (tal como se aprecia en la figura 26) con un tamaño de 108 bytes. La información que el server dice es la siguiente.

```
> Internet Protocol Version 4, Src: 172.17.0.5 (172.17.0.5), Dst:
> Transmission Control Protocol, Src Port: ssh (22), Dst Port: 52
v SSH Protocol
  Protocol: SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10
  [Direction: server-to-client]
```

Figura 27: SSH Protocol Server.

El servidor se identifica con una versión de SSH 2.0 y con una versión distinta de Ubuntu a diferencia del cliente, esto se debe a que cada apartado se está corriendo con versiones diferentes de esta.

Posteriormente, el cliente reconoce el paquete ACK. Luego el servidor comienza el intercambio de claves hacia el cliente, tal como se muestra en la siguiente figura.

TCP	66	52562 → ssh(22) [ACK] Seq=43 Ack=4
SSHv2	1178	Server: Key Exchange Init
SSHv2	1402	Client: Key Exchange Init
TCP	66	ssh(22) → 52562 [ACK] Seq=1155 Ack
TCP	66	52562 → ssh(22) [ACK] Seq=1370 Ack

Figura 28: Intercambio de claves entre Servidor y Cliente.

En el primer paquete de **tamaño 1178 bytes** llamado **Server: Key Exchange Init (KEI)**, la información en su SSH Protocol es la siguiente.

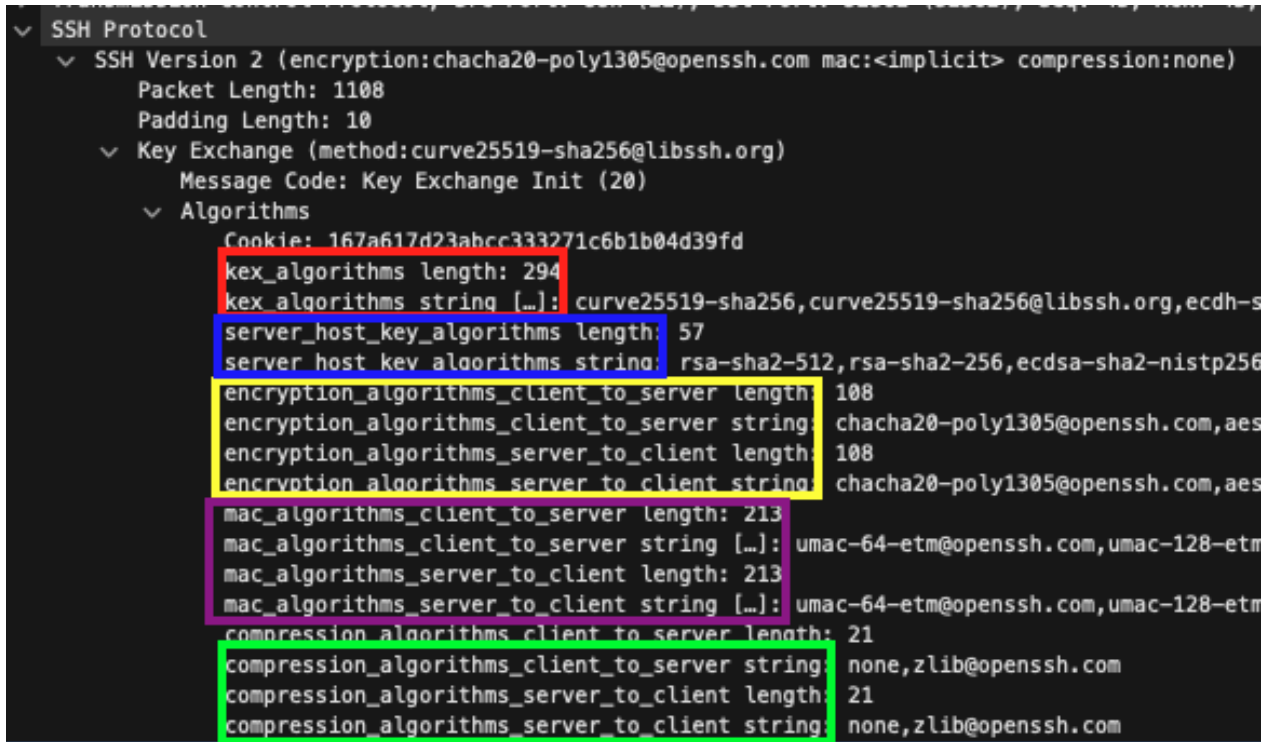


Figura 29: SSH Protocol Server C1 KEI.

En el paquete de la figura 29 es bastante grande, lo que se encuentra es lo siguiente:

- En el apartado de Algorithms, en el recuadro rojo que se aprecia en la imagen, corresponde a los KEX o Key Exchange poseen un tamaño de 294 bytes, donde posee una serie de algoritmos que sirven el intercambio de claves. Estos algoritmos son los siguientes:
 - curve25519-sha256
 - curve25519-sha256@libssh.org
 - ecdh-sha2-nistp256
 - ecdh-sha2-nistp384
 - ecdh-sha2-nistp521
 - sntrup761x25519-sha512@openssh.com
 - diffie-hellman-group-exchange-sha256
 - diffie-hellman-group16-sha512
 - ... Entre otros.
- Luego en el recuadro amarillo, se encuentran los algoritmos de encriptación de cliente a servidor y de servidor a cliente, identificandose el algoritmo y la modalidad, por ejemplo algoritmo AES-128 con modalidad CTR. Estos algoritmos son los siguientes:

- Cliente a Servidor con tamaño de 108 bytes:
 - chacha20-poly1305@openssh.com
 - aes128-ctr
 - aes192-ctr
 - aes256-ctr
 - aes128-gcm@openssh.com
 - aes256-gcm@openssh.com
- Servidor a Cliente con tamaño de 108 bytes:
 - chacha20-poly1305@openssh.com
 - aes128-ctr
 - aes192-ctr
 - aes256-ctr
 - aes128-gcm@openssh.com
 - aes256-gcm@openssh.com
- Cabe destacar que se permite negociar algoritmos distintos del servidor al cliente.
- Tambien se tiene algoritmos Mac, como se aprecia en el recuadro morado. MAC quiere decir Message Authentication Code, por ende, aquí se contiene todos los algoritmos que nos van a servir para comprobar la integridad de cada paquete, ya sea de cliente a servidor y servidor a cliente, ambos con un tamaño de 213 bytes.
 - Cliente a Servidor:
 - 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
 - hma... entre otros.
 - Servidor a Cliente:
 - 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
 - hma... entre otros.
- Finalmente, en el recuadro verde, se encuentran los algoritmos de compresión SSH para lograr una mayor eficiencia y para generar complejidad y confundir todavía más el contenido de los paquetes, teniendo la opción de comprimir el contenido utilizando el algoritmo de **zlib** par ambos casos.

Luego el Cliente contesta con los algoritmos que el soporta el cual son los siguientes:

- **kex algorithms string:**
 - curve25519-sha256@libssh.org,
 - ecdh-sha2-nistp256,
 - ecdh-sha2-nistp384,
 - ecdh-sha2-nistp521,
 - diffie-hellman-group-exchange-sha256,
 - diffie-hellman-group-exchange-sha1,
 - diffie-hellman-group14-sha1,
 - Entre otros.
- **encryption algorithms client to server string:**
 - 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
- **encryption algorithms server to client string:**
 - 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
- **mac algorithms client to server string** [Iguales para server to client]:
- 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,
 - hma... Entre otros.
- **compression algorithms string**: none,zlib@openssh.com,zlib

Luego, comienza el intercambio Diffie Hellman entre Cliente y Servidor

```

00  32302 + ssh(22) [ACK] Seq=1155 Ack=1427 Win=65536 Len=0 TSval:
114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
66  ssh(22) + 52562 [ACK] Seq=1155 Ack=1427 Win=65536 Len=0 TSval:
662 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New
66  52562 + ssh(22) [ACK] Seq=1427 Ack=1751 Win=65536 Len=0 TSval:

```

Figura 30: Intercambio Diffie Hellman.

Finalmente el cliente y servidor poseen lo necesario para generar la nueva clave de cifrado, por ende, el cliente envía el siguiente mensaje.

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

1 DESARROLLO (PARTE 1)

TCP	66	52562 → ssh(22) [ACK]
SSHv2	82	Client: New Keys
TCP	66	ssh(22) → 52562 [ACK]

Figura 31: New Keys C1.

Y a partir de este momento, todo el resto del contenido de la conversación es cifrado, apareciendo en el tráfico como **"Encrypted request/response packet"**.

Para terminar, para la obtención del HASSH, se empleó la herramienta en línea llamada <https://www.md5hashgenerator.com/>, especializada en el cálculo de hash MD5, cuyo resultado se encuentra en el paquete KEI del cliente C1, obteniendo lo siguiente.

Your String	kex_algorithms string: curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1,ext-info-c
MD5 Hash	ca977d2e2932ced588c39781485aaadf <input type="button" value="Copy"/>

Figura 32: HASSH Respectivo C1.

A partir de la imagen podemos notar que el HASSH asociado al cliente C1 es:

- **ca977d2e2932ced588c39781485aaadf**

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

Se realizaron los mismos pasos de anteriormente a la hora de capturar el tráfico con **tcpdump** ejecutado en el contenedor C4 e iniciando **SSH prueba@172.17.0.5** en este caso con el contenedor C2, colocando la contraseña **prueba** para poder acceder al servidor y finalmente finalizar la captura de tráfico con las teclas **CTL+C**.

Una vez descargado el archivo llamado **captura2.pcap** se ejecuto en Wireshark, obteniendo el siguiente tráfico de datos.

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

1 DESARROLLO (PARTE 1)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.17.0.4	172.17.0.5	TCP	74	45048 → ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1170480901 TSecr=
2	0.000007	172.17.0.5	172.17.0.4	TCP	74	ssh(22) → 45048 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1170480901 TSecr=
3	0.000114	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1170480901 TSecr=4068125491
4	0.000526	172.17.0.4	172.17.0.5	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7)
5	0.000535	172.17.0.5	172.17.0.4	TCP	66	ssh(22) → 45048 [ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=4068125492 TSecr=1170480902
6	0.015478	172.17.0.5	172.17.0.4	SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)
7	0.015556	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=2 Ack=43 Win=65536 Len=0 TSval=1170480917 TSecr=4068125507
8	0.017019	172.17.0.5	172.17.0.4	SSHv2	1178	Server: Key Exchange Init
9	0.017053	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=42 Ack=1155 Win=65536 Len=0 TSval=1170480918 TSecr=4068125508
10	0.017217	172.17.0.5	172.17.0.4	SSHv2	1426	Client: Key Exchange Init
11	0.063249	172.17.0.5	172.17.0.4	TCP	66	ssh(22) → 45048 [ACK] Seq=1155 Ack=1482 Win=65536 Len=0 TSval=4068125555 TSecr=1170480918
12	0.063300	172.17.0.4	172.17.0.5	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
13	0.063330	172.17.0.5	172.17.0.4	TCP	66	ssh(22) → 45048 [ACK] Seq=1155 Ack=1450 Win=65536 Len=0 TSval=4068125555 TSecr=1170480918
14	0.067690	172.17.0.5	172.17.0.4	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=44)
15	0.072164	172.17.0.4	172.17.0.5	SSHv2	82	Client: New Keys
16	0.113302	172.17.0.5	172.17.0.4	TCP	66	ssh(22) → 45048 [ACK] Seq=1751 Ack=1466 Win=65536 Len=0 TSval=4068125605 TSecr=1170480918
17	0.113384	172.17.0.4	172.17.0.5	SSHv2	110	Client: Encrypted packet (len=44)
18	0.113404	172.17.0.5	172.17.0.4	TCP	66	ssh(22) → 45048 [ACK] Seq=1751 Ack=1518 Win=65536 Len=0 TSval=4068125605 TSecr=1170480918
19	0.113487	172.17.0.5	172.17.0.4	SSHv2	118	Server: Encrypted packet (len=44)
20	0.113605	172.17.0.4	172.17.0.5	SSHv2	134	Client: Encrypted packet (len=68)
21	0.119583	172.17.0.5	172.17.0.4	SSHv2	118	Server: Encrypted packet (len=52)
22	0.160955	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=1578 Ack=1847 Win=65536 Len=0 TSval=1170480962 TSecr=4068125605
23	0.339006	172.17.0.5	172.17.0.4	SSHv2	214	Client: Encrypted packet (len=148)
24	0.379205	172.17.0.4	172.17.0.5	TCP	66	ssh(22) → 45048 [ACK] Seq=1847 Ack=1726 Win=65536 Len=0 TSval=4068128071 TSecr=1170492328
25	0.427125	172.17.0.5	172.17.0.4	SSHv2	94	Server: Encrypted packet (len=28)
26	0.427160	172.17.0.4	172.17.0.5	SSHv2	66	45048 → ssh(22) [ACK] Seq=1726 Ack=1875 Win=65536 Len=0 TSval=1170492328 TSecr=4068128071
27	0.427207	172.17.0.5	172.17.0.4	SSHv2	1178	Client: Encrypted packet (len=112)
28	0.427293	172.17.0.4	172.17.0.5	TCP	66	ssh(22) → 45048 [ACK] Seq=1875 Ack=1838 Win=65536 Len=0 TSval=4068128019 TSecr=1170492328
29	0.430832	172.17.0.5	172.17.0.4	SSHv2	694	Server: Encrypted packet (len=628)
30	0.480518	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=1838 Ack=2503 Win=65536 Len=0 TSval=1170492382 TSecr=4068128019
31	0.480536	172.17.0.4	172.17.0.5	SSHv2	110	Server: Encrypted packet (len=44)
32	0.480554	172.17.0.5	172.17.0.4	TCP	66	45048 → ssh(22) [ACK] Seq=1838 Ack=2547 Win=65536 Len=0 TSval=1170492382 TSecr=4068128019
33	0.480763	172.17.0.4	172.17.0.5	SSHv2	442	Client: Encrypted packet (len=376)
34	0.482083	172.17.0.5	172.17.0.4	SSHv2	174	Server: Encrypted packet (len=108)
35	0.482300	172.17.0.4	172.17.0.5	SSHv2	566	Server: Encrypted packet (len=500)
36	0.482345	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=2214 Ack=3155 Win=65536 Len=0 TSval=1170492384 TSecr=4068128019
37	0.484314	172.17.0.5	172.17.0.4	SSHv2	182	Server: Encrypted packet (len=36)
38	0.525470	172.17.0.4	172.17.0.5	TCP	66	45048 → ssh(22) [ACK] Seq=2214 Ack=3191 Win=65536 Len=0 TSval=1170492427 TSecr=4068128019

Figura 33: Trafico de datos entre C2 y C4/S1.

De igual forma que en el paso anterior, se parecia en la figura anterior, el sistema arranca con paquetes SYN, SYN-ACK y ACK, siendo este el saludo a tres vías, tal como se mostrará a continuación.

Protocol	Length	Info
TCP	74	45048 → ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=1170480901 TSecr=
TCP	74	ssh(22) → 45048 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=1170480901 TSecr=
TCP	66	45048 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1170480901 TSecr=4068125491
SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7)

Figura 34: Inicio de paquetes C2.

En la información del Source es quien inicia la conexión es el equipo **cliente** con IP **172.17.0.4** dirigido al destination quien es el **servidor** IP **172.17.0.5**.

Luego, el cliente realiza la acción de identificarse con un tamaño de 107 bytes. Si examinamos el SSH Protocol, nos encontraremos con lo siguiente.

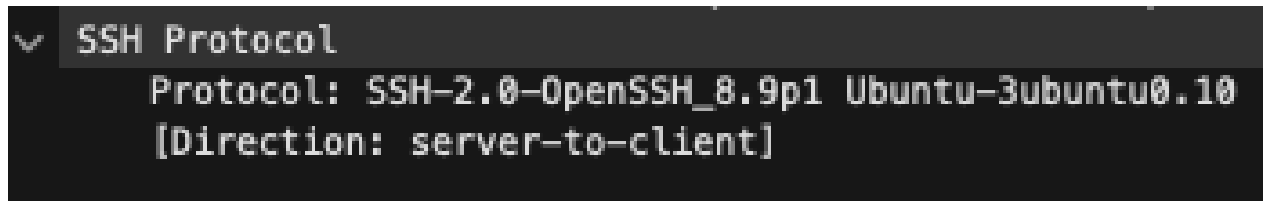
SSH Protocol
Protocol: SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7
[Direction: client-to-server]

Figura 35: SSH Protocol Client.

En el contenido del protocolo de la figura 34, nos indica que es un **protocolo SSH 2.0** y aparte de eso nos da información de la versión del sistema operativo, que en este caso es **Ubuntu-4ubuntu0.7** en el cual se esta corriendo, con una dirección de cliente a servidor.

Luego el paquete siguiente es un ACK, que significa que el servidor esta reconociendo el paquete del servidor.

Por otro lado la información de SSH Protocol Server se obtiene lo siguiente.



```

SSH Protocol
Protocol: SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10
[Direction: server-to-client]

```

Figura 36: SSH Protocol Server.

El servidor se identifica con una versión de SSH 2.0 y con una versión distinta de Ubuntu a diferencia del cliente, esto se debe a que cada apartado se esta corriendo con versiones diferentes de esta.

Posteriormente, el cliente reconoce el paquete ACK. Luego el servidor comienza el intercambio de claves hacia el cliente, tal como se muestra en la siguiente figura.

SSHv2	1178	Server: Protocol (SSH-2.0-openssh_8.9p1 Ubuntu-3ubuntu0.10)
TCP	66	45048 → ssh(22) [ACK] Seq=42 Ack=42
SSHv2	1178	Server: Key Exchange Init
TCP	66	45048 → ssh(22) [ACK] Seq=42 Ack=42
SSHv2	1426	Client: Key Exchange Init

Figura 37: Intercambio de claves Server KEI.

El paquete llamado Server: Key Exchange Init posee un tamaño de 1178 bytes, la información en su SSH Protocol es la siguiente.

```

v Algorithms
Cookie: f36139f24b91a3179e77943a4b062310
kex_algorithms length: 294
kex_algorithms string [...]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,rsa-sha2-256,rsa-sha2-512,ssh-ed25519,ssh-ed448
server_host_key_algorithms length: 57
server host key algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed25519,ssh-ed448
encryption_algorithms_client_to_server length: 108
encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-gcm,aes256-ctr
encryption_algorithms_server_to_client length: 108
encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-gcm,aes256-ctr
mac_algorithms_client_to_server length: 21
mac_algorithms_client_to_server string [...]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1
mac_algorithms_server_to_client length: 21
mac algorithms server to client string [...]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1
compression_algorithms_client_to_server length: 21
compression_algorithms_client_to_server string: none,zlib@openssh.com
compression_algorithms_server_to_client length: 21
compression_algorithms_server_to_client string: none,zlib@openssh.com
languages_client_to_server length: 0
languages_client_to_server string:
languages server to client length: 0

```

Figura 38: SSH Protocol Server KEI C2.

- En el apartado de Algorithms, en el recuadro rojo que se aprecia en la imagen, corresponde a los KEX o Key Exchange poseen un tamaño de 294 bytes, donde posee una serie de algoritmos que sirven el intercambio de claves. Estos algoritmos son los siguientes:
 - curve25519-sha256,
 - curve25519-sha256@libssh.org,
 - ecdh-sha2-nistp256,
 - ecdh-sha2-nistp384,
 - ecdh-sha2-nistp521,
 - sntrup761x25519-sha512@openssh.com,
 - diffie-hellman-group-exchange-sha256,
 - diffie-hellman-group16-sha512,
 - Entre otros...
- Luego en el recuadro amarillo, se encuentran los algoritmos de encriptación de cliente a servidor y de servidor a cliente, identificandose el algoritmo y la modalidad, por ejemplo algoritmo AES-128 con modalidad CTR. Estos algoritmos son los siguientes (son identicos para Cliente a Server y de Server a Cliente):
 - chacha20-poly1305@openssh.com,

- aes128-ctr,
 - aes192-ctr,
 - aes256-ctr,
 - aes128-gcm@openssh.com,
 - aes256-gcm@openssh.com
- Tambien se tiene algoritmos Mac, como se aprecia en el recuadro morado. MAC quiere decir Message Authentication Code, por ende, aquí se contiene todos los algoritmos que nos van a servir para comprobar la integridad de cada paquete, ya sea de cliente a servidor y servidor a cliente, ambos con un tamaño de 213 bytes (son identicos para Cliente a Server y de Server a Cliente):
- 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,
 - hma... Entre otros...
- Finalmente, en el recuadro azul, se encuentran los algoritmos de compresión SSH para lograr una mayor eficiencia y para generar complejidad y confundir todavía más el contenido de los paquetes, teniendo la opción de comprimir el contenido utilizando el algoritmo de **zlib** par ambos casos.

Luego el Cliente contesta con los algoritmos que el soporta el cual son los siguientes:

```

v Algorithms
  Cookie: 0c7c7ba76aa318cf38f7d3837c2d7239
  kex_algorithms length: 304
  kex_algorithms string [...]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh
  server_host_key_algorithms length: 290
  server_host_key_algorithms string [...]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp3
  encryption_algorithms_client_to_server length: 108
  encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr
  encryption_algorithms_server_to_client length: 108
  encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr
  mac_algorithms_client_to_server length: 213
  mac_algorithms_client_to_server string [...]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac
  mac_algorithms_server_to_client length: 213
  mac_algorithms_server_to_client string [...]: umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac
  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:

```

Figura 39: SSH Protocol Client KEI C2.

- **kex algorithms string** [304 bytes]:
 - 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,
 - entre otros.
- **encryption algorithms client to server string** (Igual que en server to client) con tamaño 108 bytes:
 - chacha20-poly1305@openssh.com,
 - aes128-ctr,
 - aes192-ctr,
 - aes256-ctr,
 - aes128-gcm@openssh.com,
 - aes256-gcm@openssh.com

- **mac algorithms client to server string** (Igual que en server to client) con tamaño 213 bytes :
 - 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,
 - hma...entre otros..
- **compression algorithms string**: none,zlib@openssh.com,zlib

Luego, comienza el intercambio Diffie Hellman entre Cliente y Servidor, y finalmente el cliente y servidor poseen lo necesario para generar la nueva clave de cifrado, por ende, el cliente envía el siguiente mensaje.

SSHv2	662	Server: Elliptic Curv
SSHv2	82	Client: New Keys
TCP	66	ssh(22) → 45048 [ACK]

Figura 40: New Keys C2.

Y a partir de este momento, todo el resto del contenido es cifrado apareciendo en el tráfico como **"Encrypted request/response packet"**.

Para terminar, igual que anteriormente, obtendremos el HASSH respectivo, volviendo a utilizar la herramienta en línea anterior, especializada en el cálculo de hash MD5, cuyo resultado se encuentra en el paquete KEI del cliente C2, obteniendo lo siguiente.

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

1 DESARROLLO (PARTE 1)

Your String	curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,sntrup761x25519-sha512@openssh.com,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512
MD5 Hash	cde2984d83538b112eff4b330f5cde00 <button>Copy</button>

Figura 41: HASSH Respectivo C2.

A partir de la imagen podemos notar que el HASSH asociado al cliente C2 es:

■ cde2984d83538b112eff4b330f5cde00

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

Se capturo el trafico con tcpdump con el nombre del archivo llamado captura3.pcap. Una vez ejecutado en Wireshark, se obtuvo el siguiente trafico de paquetes.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.17.0.2	172.17.0.5	TCP	74	51186 → ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4050667911 TSecr=
2	0.000138	172.17.0.5	172.17.0.2	TCP	74	ssh(22) → 51186 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=13476
3	0.000281	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 MSS=65495 SACK_PERM TSval=1347665581
4	0.001555	172.17.0.2	172.17.0.5	SSHv2	108	Client: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.11)
5	0.001563	172.17.0.5	172.17.0.2	TCP	66	ssh(22) → 51186 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=1347665582 TSecr=4050667913
6	0.016576	172.17.0.5	172.17.0.2	SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.10)
7	0.016638	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSval=4050667928 TSecr=1347665597
8	0.017263	172.17.0.2	172.17.0.5	SSHv2	1602	Client: Key Exchange Init
9	0.018370	172.17.0.5	172.17.0.2	SSHv2	1178	Server: Key Exchange Init
10	0.020984	172.17.0.2	172.17.0.5	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11	0.020827	172.17.0.5	172.17.0.2	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (
12	0.020951	172.17.0.2	172.17.0.5	SSHv2	82	Client: New Keys
13	0.071175	172.17.0.5	172.17.0.2	TCP	66	ssh(22) → 51186 [ACK] Seq=1751 Ack=1643 Win=65536 Len=0 TSval=1347665652 TSecr=4050667
14	0.071263	172.17.0.2	172.17.0.5	SSHv2	118	Client: Encrypted packet (Len=44)
15	0.071297	172.17.0.5	172.17.0.2	SSHv2	66	ssh(22) → 51186 [ACK] Seq=1751 Ack=1687 Win=65536 Len=0 TSval=1347665652 TSecr=4050667
16	0.071427	172.17.0.5	172.17.0.2	SSHv2	110	Server: Encrypted packet (Len=44)
17	0.071577	172.17.0.2	172.17.0.5	SSHv2	134	Client: Encrypted packet (Len=68)
18	0.071821	172.17.0.5	172.17.0.2	SSHv2	118	Server: Encrypted packet (Len=62)
19	0.119751	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=1755 Ack=1847 Win=65536 Len=0 TSval=4050668831 TSecr=1347665
20	2.000876	172.17.0.5	172.17.0.2	SSHv2	214	Client: Encrypted packet (Len=148)
21	2.041081	172.17.0.2	172.17.0.5	TCP	66	ssh(22) → 51186 [ACK] Seq=1847 Ack=1983 Win=65536 Len=0 TSval=1347668422 TSecr=4050670
22	2.090854	172.17.0.5	172.17.0.2	SSHv2	94	Server: Encrypted packet (Len=28)
23	2.090724	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=1983 Ack=1875 Win=65536 Len=0 TSval=4050670802 TSecr=1347668
24	2.090951	172.17.0.2	172.17.0.5	SSHv2	178	Client: Encrypted packet (Len=112)
25	2.090959	172.17.0.5	172.17.0.2	TCP	66	ssh(22) → 51186 [ACK] Seq=1875 Ack=2015 Win=65536 Len=0 TSval=1347668471 TSecr=4050670
26	2.092485	172.17.0.2	172.17.0.5	SSHv2	694	Server: Encrypted packet (Len=628)
27	2.947839	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=2015 Ack=2583 Win=65536 Len=0 TSval=4050670858 TSecr=1347668
28	2.947861	172.17.0.5	172.17.0.2	SSHv2	118	Server: Encrypted packet (Len=44)
29	2.947181	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=2015 Ack=2547 Win=65536 Len=0 TSval=4050670859 TSecr=1347668
30	2.947319	172.17.0.2	172.17.0.5	SSHv2	442	Client: Encrypted packet (Len=376)
31	2.948839	172.17.0.5	172.17.0.2	SSHv2	174	Server: Encrypted packet (Len=188)
32	2.949185	172.17.0.2	172.17.0.5	SSHv2	566	Server: Encrypted packet (Len=500)
33	2.949277	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=2391 Ack=3155 Win=65536 Len=0 TSval=4050670861 TSecr=1347668
34	2.951977	172.17.0.5	172.17.0.2	SSHv2	182	Server: Encrypted packet (Len=36)
35	2.992392	172.17.0.2	172.17.0.5	TCP	66	51186 → ssh(22) [ACK] Seq=2391 Ack=3191 Win=65536 Len=0 TSval=4050670984 TSecr=1347668

Figura 42: Trafico de datos entre C3 y C4/S1.

En la información del Source es quien inicia la conexión es el equipo **cliente** con IP **172.17.0.2** dirigido al destination quien es el **servidor** IP **172.17.0.5**.

Luego, el cliente realiza la acción de identificarse con un tamaño de 108 bytes. Si examinamos el SSH Protocol, nos encontraremos con lo siguiente.

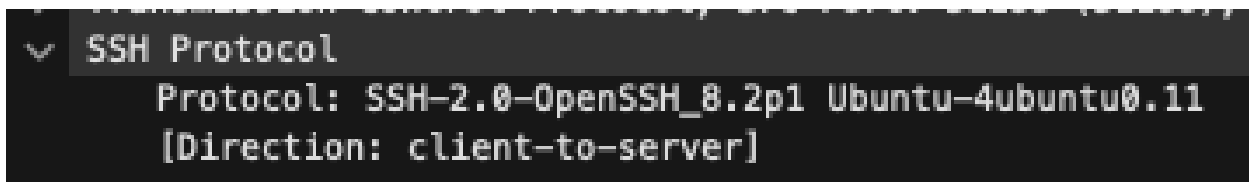


Figura 43: SSH Protocol Client.

En el contenido del protocolo de la figura 34, nos indica que es un **protocolo SSH 2.0** y aparte de eso nos da información de la versión del sistema operativo, que en este caso es **Ubuntu-4ubuntu0.11** en el cual se esta corriendo, con una dirección de cliente a servidor. Luego el paquete siguiente es un ACK, que significa que el servidor esta reconociendo el paquete del servidor.

Por otro lado la información de SSH Protocol Server se obtiene lo siguiente.

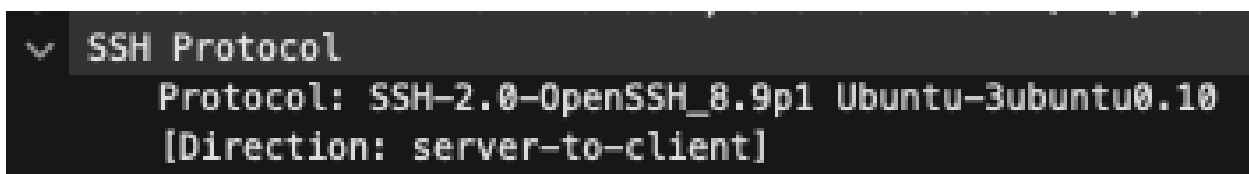


Figura 44: SSH Protocol Server.

El servidor se identifica con una versión de SSH 2.0 y con una versión distinta de Ubuntu a diferencia del cliente, esto se debe a que cada apartado se esta corriendo con versiones diferentes de esta.

Posteriormente, el cliente reconoce el paquete ACK. Luego el servidor comienza el intercambio de claves hacia el cliente, tal como se muestra en la siguiente figura.

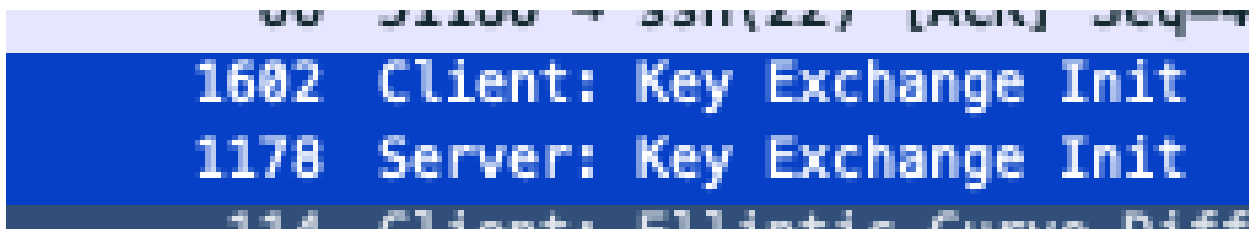


Figura 45: Intercambio claves entre Cliente y Servidor KEI.

El paquete Client Key Exchange Init posee un tamaño de 1602 bytes, y dentro de su SSH Protocol posee los siguiente algoritmos.

```

    Algorithms
      Cookie: c5a9c3ebc099ce839636c410fdda05b5
      kex_algorithms length: 270
      kex_algorithms string [...]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-
      server_host_key_algorithms length: 500
      server_host_key_algorithms string [...]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-
      encryption_algorithms_client_to_server length: 108
      encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-
      encryption_algorithms_server_to_client length: 108
      encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-
      mac_algorithms_client_to_server length: 213
      mac_algorithms_client_to_server string [...]: umac-64-etm@openssh.com,umac-128-etm@ope
      mac_algorithms_server_to_client length: 213
      mac_algorithms_server_to_client string [...]: umac-64-etm@openssh.com,umac-128-etm@ope
      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
  
```

Figura 46: SSH Protocol Client KEI C3.

- En el apartado de Algorithms, en el recuadro rojo que se aprecia en la imagen, corresponde a los KEX o Key Exchange poseen un tamaño de 270 bytes, donde posee una serie de algoritmos que sirven el intercambio de claves. Estos algoritmos son los siguientes:
 - 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-he... entre otros.
- Luego en el recuadro amarillo, se encuentran los algoritmos de encriptación de cliente a servidor y de servidor a cliente, identificandose el algoritmo y la modalidad, por ejemplo algoritmo AES-128 con modalidad CTR. Estos algoritmos son los siguientes (son identicos para Cliente a Server y de Server a Cliente):
 - chacha20-poly1305@openssh.com,
 - aes128-ctr,

- aes192-ctr,
 - aes256-ctr,
 - aes128-gcm@openssh.com,
 - aes256-gcm@openssh.com
- Tambien se tiene algoritmos Mac, como se aprecia en el recuadro morado. MAC quiere decir Message Authentication Code, por ende, aquí se contiene todos los algoritmos que nos van a servir para comprobar la integridad de cada paquete, ya sea de cliente a servidor y servidor a cliente, ambos con un tamaño de 213 bytes (son identicos para Cliente a Server y de Server a Cliente):
- 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,
 - hma... entre otros.
- Finalmente, en el recuadro azul, se encuentran los algoritmos de compresión SSH para lograr una mayor eficiencia y para generar complejidad y confundir todavía más el contenido de los paquetes, teniendo la opción de comprimir el contenido utilizando el algoritmo de **zlib** par ambos casos.

Luego el Server contesta con los algoritmos que el soporta el cual son los siguientes:

- **kex algorithms string** [304 bytes]:
- curve25519-sha256,
 - curve25519-sha256@libssh.org,
 - ecdh-sha2-nistp256,
 - ecdh-sha2-nistp384,
 - ecdh-sha2-nistp521,
 - sntrup761x25519-sha512@openssh.com,
 - diffie-hellman-group-exchange-sha256,
 - diffie-hellman-group16-sha512,
 - diff... entre otros...

- **encryption algorithms client to server string (Igual que en server to client) con tamaño 108 bytes:**
 - chacha20-poly1305@openssh.com,
 - aes128-ctr,
 - aes192-ctr,
 - aes256-ctr,
 - aes128-gcm@openssh.com,
 - aes256-gcm@openssh.com
- **mac algorithms client to server string (Igual que en server to client) con tamaño 213 bytes :**
 - 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,
 - hma...entre otros..
- **compression algorithms string:** none,zlib@openssh.com,zlib

Luego, comienza el intercambio Diffie Hellman entre Cliente y Servidor, y finalmente el cliente y servidor poseen lo necesario para generar la nueva clave de cifrado, por ende, el cliente envía el siguiente mensaje. Y a partir de este momento, todo el resto del contenido es cifrado apareciendo en el tráfico como **"Encrypted request/response packet"**.

Para terminar, igual que anteriormente, obtendremos el HASSH respectivo, volviendo a utilizar la herramienta en línea anterior, especializada en el cálculo de hash MD5, cuyo resultado se encuentra en el paquete KEI del cliente C3, obteniendo lo siguiente.

Your String	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-he	
MD5 Hash	7b70652ea6d3646cd97953e9b9fb03ff	<button>Copy</button>

Figura 47: HASSH Respectivo C3.

A partir de la imagen podemos asumir que el HASSH asociado a C3 es:

- 7b70652ea6d3646cd97953e9b9fb03ff

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

Se capturo el trafico con tcpdump con el nombre del archivo llamado captura4.pcap dentro del mismo contenedor en donde se ejecuta el server (C4 hacia C4/S1).

```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [0:56:23] C:1
$ docker exec -it C4 bash

root@c9228708f4be:/# tcpdump -i any port 22 -w /tmp/captura4.pcap
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
^C35 packets captured
70 packets received by filter
0 packets dropped by kernel
root@c9228708f4be:/#
```

Figura 48: Captura de trafico directo.

A diferencia de las capturas anteriores, esta captura se realiza de forma directa dentro del mismo contenedor y guardada en la carpeta tmp dentro del contenedor C4 para poder capturar correctamente los paquetes en la comunicación.

Una vez ejecutado en Wireshark, se obtuvo el siguiente trafico de paquetes.

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

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.17.0.5	172.17.0.5	TCP	80	58952 → ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=2567986529 TSecr=2567986529
2	0.000035	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2567986529 TSecr=2567986529
3	0.000051	172.17.0.5	172.17.0.5	TCP	114	Client: Protocol (SSH-2.0-openssh.8.9p1 Ubuntu-jubuntu0.10)
4	0.001356	172.17.0.5	172.17.0.5	SSHv2	72	ssh(22) → 58952 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=2567986530 TSecr=2567986530
5	0.001371	172.17.0.5	172.17.0.5	TCP	114	Server: Protocol (SSH-2.0-openssh.8.9p1 Ubuntu-jubuntu0.10)
6	0.013570	172.17.0.5	172.17.0.5	SSHv2	72	58952 → ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSval=2567986542 TSecr=2567986542
7	0.013589	172.17.0.5	172.17.0.5	TCP	1608	Client: Key Exchange Init
8	0.014054	172.17.0.5	172.17.0.5	SSHv2	1184	Server: Key Exchange Init
9	0.015368	172.17.0.5	172.17.0.5	SSHv2	120	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
10	0.016904	172.17.0.5	172.17.0.5	SSHv2	596	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=44)
11	0.022590	172.17.0.5	172.17.0.5	SSHv2	80	Client: New Keys
12	0.029182	172.17.0.5	172.17.0.5	TCP	72	ssh(22) → 58952 [ACK] Seq=1679 Ack=1643 Win=65536 Len=0 TSval=2567986600 TSecr=2567986600
13	0.071468	172.17.0.5	172.17.0.5	SSHv2	116	Client: Encrypted packet (len=44)
14	0.071482	172.17.0.5	172.17.0.5	TCP	72	ssh(22) → 58952 [ACK] Seq=1679 Ack=1687 Win=65536 Len=0 TSval=2567986600 TSecr=2567986600
15	0.071492	172.17.0.5	172.17.0.5	SSHv2	116	Server: Encrypted packet (len=44)
16	0.071555	172.17.0.5	172.17.0.5	SSHv2	140	Client: Encrypted packet (len=68)
17	0.071636	172.17.0.5	172.17.0.5	SSHv2	124	Server: Encrypted packet (len=52)
18	0.078371	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=1775 Ack=1775 Win=65536 Len=0 TSval=2567986648 TSecr=2567986648
19	0.119497	172.17.0.5	172.17.0.5	SSHv2	228	Client: Encrypted packet (len=148)
20	2.153425	172.17.0.5	172.17.0.5	TCP	72	ssh(22) → 58952 [ACK] Seq=1775 Ack=1983 Win=65536 Len=0 TSval=2567988723 TSecr=2567988723
21	2.194067	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=1983 Ack=1803 Win=65536 Len=0 TSval=2567988726 TSecr=2567988726
22	2.197727	172.17.0.5	172.17.0.5	SSHv2	184	Client: Encrypted packet (len=112)
23	2.197753	172.17.0.5	172.17.0.5	TCP	72	ssh(22) → 58952 [ACK] Seq=1803 Ack=2015 Win=65536 Len=0 TSval=2567988727 TSecr=2567988727
24	2.197964	172.17.0.5	172.17.0.5	SSHv2	700	Server: Encrypted packet (len=628)
25	2.197969	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=2015 Ack=2431 Win=65536 Len=0 TSval=2567988785 TSecr=2567988785
26	2.214777	172.17.0.5	172.17.0.5	SSHv2	116	Server: Encrypted packet (len=44)
27	2.256285	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=2015 Ack=2475 Win=65536 Len=0 TSval=2567988785 TSecr=2567988785
28	2.256228	172.17.0.5	172.17.0.5	SSHv2	180	Server: Encrypted packet (len=100)
29	2.256248	172.17.0.5	172.17.0.5	TCP	572	Server: Encrypted packet (len=508)
30	2.256428	172.17.0.5	172.17.0.5	SSHv2	448	Client: Encrypted packet (len=376)
31	2.257725	172.17.0.5	172.17.0.5	SSHv2	72	58952 → ssh(22) [ACK] Seq=2391 Ack=3083 Win=65536 Len=0 TSval=2567988787 TSecr=2567988787
32	2.257938	172.17.0.5	172.17.0.5	TCP	180	Server: Encrypted packet (len=36)
33	2.257984	172.17.0.5	172.17.0.5	SSHv2	72	58952 → ssh(22) [ACK] Seq=2391 Ack=3119 Win=65536 Len=0 TSval=2567988830 TSecr=2567988830
34	2.260836	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=2391 Ack=3119 Win=65536 Len=0 TSval=2567988830 TSecr=2567988830
35	2.300982	172.17.0.5	172.17.0.5	TCP	72	58952 → ssh(22) [ACK] Seq=2391 Ack=3119 Win=65536 Len=0 TSval=2567988830 TSecr=2567988830

Figura 49: Trafico de datos entre C4 y C4/S1.

En la información del Source es quien inicia la conexión es el equipo **cliente con IP 172.17.0.5** dirigido al destination quien es el **servidor IP 172.17.0.5**. Son identicas ya que se ejecutaron dentro del mismo contenedor.

Luego, el cliente realiza la acción de identificarse con un tamaño de 108 bytes. Si examinamos el SSH Protocol, nos encontraremos que ambos, ya sea, cliente y servidor, poseen el mismo SSH Protocol, indicando versión de SSH 2.0 y una versión de Ubuntu-3ubuntu0.10, ya que se ejecuto dentro del mismo contenedor.

Posteriormente el cliente reconoce el paquete ACK. Luego el cliente comienza el intercambio de claves hacia el server. Envia un paquete llamado Client Key Exchange Init de tamaño 1608 bytes, que contiene lo siguiente.

```

    Algorithms
    Cookie: ee0a18ad09bb4ce7685b29dbfe71af19
    kex_algorithms length: 305
    kex_algorithms string [...]: curve25519-sha256,curve25519-sha256@libssh.org,ecdh-
    server_host_key_algorithms length: 463
    server host key algorithms string [...]: ssh-ed25519-cert-v01@openssh.com,ecdsa-
    encryption_algorithms_client_to_server length: 108
    encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,a
    encryption_algorithms_server_to_client length: 108
    encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,a
    mac_algorithms_client_to_server length: 213
    mac_algorithms_client_to_server string [...]: umac-64-etm@openssh.com,umac-128-e
    mac_algorithms_server_to_client length: 213
    mac_algorithms_server_to_client string [...]: umac-64-etm@openssh.com,umac-128-e
    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:
    
```

Figura 50: SSH Protocol Client C4 KEI.

En el paquete de la figura 29 es bastante grande, lo que se encuentra es lo siguiente:

- En el apartado de Algorithms, en el recuadro rojo que se aprecia en la imagen, corresponde a los KEX o Key Exchange poseen un tamaño de 305 bytes, donde posee una serie de algoritmos que sirven el intercambio de claves. Estos algoritmos son los siguientes:
 - curve25519-sha256,
 - curve25519-sha256@libssh.org,
 - ecdh-sha2-nistp256,
 - ecdh-sha2-nistp384,
 - ecdh-sha2-nistp521,
 - sntrup761x25519-sha512@openssh.com,
 - diffie-hellman-group-exchange-sha256,
 - diffie-hellman-group16-sha512,
 - diff.. entre otros
- Luego en el recuadro amarillo, se encuentran los algoritmos de encriptación de cliente a servidor y de servidor a cliente, identificandose el algoritmo y la modalidad, por ejemplo algoritmo AES-128 con modalidad CTR. Estos algoritmos son los siguientes:

- Cliente a Servidor (igual que Server a Cliente) con tamaño de 108 bytes:
 - chacha20-poly1305@openssh.com
 - aes128-ctr
 - aes192-ctr
 - aes256-ctr
 - aes128-gcm@openssh.com
 - aes256-gcm@openssh.com

- Cabe destacar que se permite negociar algoritmos distintos del servidor al cliente.

También se tiene algoritmos Mac, como se aprecia en el recuadro morado. MAC quiere decir Message Authentication Code, por ende, aquí se contiene todos los algoritmos que nos van a servir para comprobar la integridad de cada paquete, ya sea de cliente a servidor y servidor a cliente, ambos con un tamaño de 213 bytes.

- Cliente a Servidor (igual que Server a Cliente):
 - 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
 - hma... entre otros.

Finalmente, en el recuadro verde, se encuentran los algoritmos de compresión SSH para lograr una mayor eficiencia y para generar complejidad y confundir todavía más el contenido de los paquetes, teniendo la opción de comprimir el contenido utilizando el algoritmo de **zlib** par ambos casos.

Por otro lado, el Server contesta con los algoritms que el soporta, en este caso son los mismos que soporta el cliente, ya que, estan corriendo dentro del mismo contenedor.

Luego, comienza el intercambio Diffie Hellman entre Cliente y Servidor, y finalmente el cliente y servidor poseen lo necesario para generar la nueva clave de cifrado, por ende, el cliente envia el siguiente mensaje. Y a partir de este momento, todo el resto del contenido es cifrado apareciendo en el trafico como **"Encrypted request/response packet"**.

Para terminar, igual que anteriormente, obtendremos el HASSH respectivo, volviendo a utilizar la herramienta en línea anterior, especializada en el cálculo de hash MD5, cuyo resultado se encuentra en el paquete KEI del cliente C4, obteniendo lo siguiente.

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

1 DESARROLLO (PARTE 1)

Your String	curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,sntrup761x25519-sha512@openssh.com,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diff
MD5 Hash	54806ddcf330abf3d4e13e85942b8671 <input type="button" value="Copy"/>

Figura 51: HASSH Respectivo C4.

A partir de la imagen podemos notar que el HASSH asociado al cliente C4 es:

- 54806ddcf330abf3d4e13e85942b8671

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

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

Para comprender a profundidad el análisis de los datos transmitidos en comunicaciones de red, podemos distinguir tres etapas primordiales en el flujo de información de los paquetes de texto:

- **Establecimiento de Conexión Inicial:** Esta primera fase se centra en los procedimientos preliminares de conexión, donde los sistemas involucrados negocian los parámetros básicos de comunicación y establecen el marco inicial del protocolo de interacción.
- **Fase de Intercambio Criptográfico:** En esta etapa crítica, los dispositivos proceden al intercambio seguro de claves criptográficas, implementando mecanismos que garantizan la confidencialidad y la integridad de la información transmitida.
- **Proceso de Verificación de Identidad (Autenticación):** La última fase comprende los mecanismos de autenticación, donde se validan las credenciales de los participantes, se confirman sus identidades y se establecen los niveles de acceso y privilegios correspondientes.

Cada una de estas etapas juega un papel fundamental en la seguridad y la eficacia de las comunicaciones de red, asegurando que la transferencia de información se realice de manera protegida y controlada.

1.8.1. C1

1. Establecimiento de Conexión Inicial: Durante la fase inicial de conexión, se intercambian paquetes que revelan detalles fundamentales sobre la implementación del protocolo SSH. Estos paquetes incluyen información crítica como la identificación de la versión del protocolo y el software específico que se utilizará para la comunicación. En un caso particular, se identificó una configuración con **SSH versión 2.0, utilizando OpenSSH 7.2p2** en un entorno **Ubuntu-4ubuntu2.10**, lo que permite una negociación precisa entre el cliente y el servidor, tal como se aprecia en la imagen de acontinuación.

TCP	66	52562 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=454120
SSHv2	108	Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10)
TCP	66	ssh(22) → 52562 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=32379
SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)

Figura 52: Versión SSH y OpenSSH Client C1.

El proceso de conexión implica un intercambio detallado que define los parámetros de comunicación, especialmente los algoritmos compatibles. Se negocian aspectos como los métodos de cifrado, compresión e intercambio de claves. A modo de ejemplo, entre las opciones de cifrado se puede encontrar el algoritmo aes256-ctr. Esta información se transmite en formato de texto plano, revelando únicamente las capacidades de comunicación sin comprometer ninguna información sensible.

2. Fase de Intercambio Criptográfico: El proceso de establecimiento de una conexión segura involucra un mecanismo de intercambio de claves que define los parámetros fundamentales para garantizar una comunicación protegida. En este escenario, se observa la aplicación de un método criptográfico de intercambio de claves conocido por su robustez y eficacia: el algoritmo de Diffie-Hellman.

66	52562 → ssh(22) [ACK] Seq=1379 Ack=1155 Win=65536 Len=0 TSval=454120192 TSecr=3237948751
114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
66	ssh(22) → 52562 [ACK] Seq=1155 Ack=1427 Win=65536 Len=0 TSval=3237948797 TSecr=454120192
662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
66	52562 → ssh(22) [ACK] Seq=1427 Ack=1751 Win=65536 Len=0 TSval=454120198 TSecr=3237948803

Figura 53: Intercambio Algoritmo Diffie Hellman.

Los paquetes transmitidos durante esta etapa revelan detalles técnicos sobre los métodos propuestos para generar una clave de sesión compartida. La información se presenta en formato de texto plano, lo que permite una negociación transparente entre los sistemas involucrados. Este enfoque permite a ambas partes acordar un método seguro de generación de claves sin comprometer la confidencialidad del proceso.

3. Proceso de Verificación de Identidad (Autenticación): Durante el proceso de conexión segura, las sesiones que emplean autenticación mediante contraseña revelan ciertos

aspectos interesantes sobre la comunicación. Específicamente, el nombre de usuario se transmite de manera transparente, visible en el flujo de comunicación inicial.

Sin embargo, el protocolo implementa medidas de seguridad significativas para proteger la información sensible. La contraseña, elemento crítico en el proceso de autenticación, se somete a un riguroso protocolo de encriptación. Esta estrategia previene la exposición directa de credenciales, garantizando que los datos confidenciales permanezcan protegidos durante la transmisión.

1.8.2. C2

1. Establecimiento de Conexión Inicial: Durante la fase inicial de conexión, se intercambian paquetes que revelan detalles fundamentales sobre la implementación del protocolo SSH. Estos paquetes incluyen información crítica como la identificación de la versión del protocolo y el software específico que se utilizará para la comunicación. En un caso particular, se identificó una configuración con **SSH versión 2.0, utilizando OpenSSH 7.6p1** en un entorno **Ubuntu-4ubuntu0.7**, lo que permite una negociación precisa entre el cliente y el servidor, tal como se aprecia en la imagen de continuación.

TCP	66	45048 → ssh(22)	[ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=117048
SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7)	
TCP	66	ssh(22) → 45048	[ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=40681
SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)	

Figura 54: Versión SSH y openSSH Client C2.

El proceso de conexión implica un intercambio detallado que define los parámetros de comunicación, especialmente los algoritmos compatibles. Se negocian aspectos como los métodos de cifrado, compresión e intercambio de claves. A modo de ejemplo, entre las opciones de cifrado se puede encontrar el algoritmo aes256-ctr. Esta información se transmite en formato de texto plano, revelando únicamente las capacidades de comunicación sin comprometer ninguna información sensible.

2. Fase de Intercambio Criptográfico: Los paquetes contienen los parámetros y los métodos de intercambio de claves propuestos para realizar la conexión. En este caso, también se muestra el algoritmo Diffie-Hellman. La información de estos paquetes va en texto plano debido a que son las instrucciones que se realizan para generar una clave de sesión compartida y segura, tal como se muestra en la siguiente figura.

66	ssh(22) → 45048	[ACK] Seq=1155 Ack=1402 Win=65536 Len=0 TSval=4068125555 TSecr=1170488919
114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init	
66	ssh(22) → 45048	[ACK] Seq=1155 Ack=1450 Win=65536 Len=0 TSval=4068125555 TSecr=1170488965
662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)	
82	Client: New Keys	

Figura 55: Intercambio Algoritmo Diffie Hellman.

3. Proceso de Verificación de Identidad (Autenticación): En las sesiones que exigen autenticación mediante contraseña, es posible interceptar mensajes que incluyen una solicitud de autenticación, donde se muestra el nombre del usuario que intenta establecer la conexión en texto sin cifrar. No obstante, por razones de seguridad, la contraseña no se transmite en texto claro, ya que SSH se encarga de encriptar los datos de autenticación.

1.8.3. C3

1. Establecimiento de Conexión Inicial: Durante la fase inicial de conexión, se intercambian paquetes que revelan detalles fundamentales sobre la implementación del protocolo SSH. Estos paquetes incluyen información crítica como la identificación de la versión del protocolo y el software específico que se utilizará para la comunicación. En un caso particular, se identificó una configuración con **SSH versión 2.0, utilizando OpenSSH 8.2p1** en un entorno **Ubuntu-4ubuntu0.11**, lo que permite una negociación precisa entre el cliente y el servidor, tal como se aprecia en la imagen de acontinuación.

TCP	66	51186 → ssh(22)	[ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4050667912
SSHv2	108	Client: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.11)	
TCP	66	ssh(22) → 51186	[ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=134766558
SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.11)	

Figura 56: SSH y OpenSSH Client C3.

El proceso de conexión implica un intercambio detallado que define los parámetros de comunicación, especialmente los algoritmos compatibles. Se negocian aspectos como los métodos de cifrado, compresión e intercambio de claves. A modo de ejemplo, entre las opciones de cifrado se puede encontrar el algoritmo aes256-ctr. Esta información se transmite en formato de texto plano, revelando únicamente las capacidades de comunicación sin comprometer ninguna información sensible.

2. Fase de Intercambio Criptográfico: Los paquetes contienen los parámetros y los métodos de intercambio de claves propuestos para realizar la conexión. En este caso, también se muestra el algoritmo Diffie-Hellman. La información de estos paquetes va en texto plano debido a que son las instrucciones que se realizan para generar una clave de sesión compartida y segura, tal como se muestra en la siguiente figura.

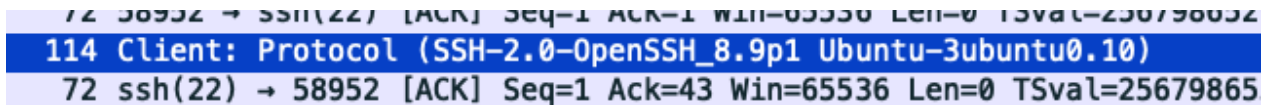
114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)

Figura 57: Intercambio Algoritmo Diffie Hellman.

3. Proceso de Verificación de Identidad (Autenticación): En las sesiones que exigen autenticación mediante contraseña, es posible interceptar mensajes que incluyen una solicitud de autenticación, donde se muestra el nombre del usuario que intenta establecer la conexión en texto sin cifrar. No obstante, por razones de seguridad, la contraseña no se transmite en texto claro, ya que SSH se encarga de encriptar los datos de autenticación.

1.8.4. C4/S1

1. Establecimiento de Conexión Inicial: Durante la fase inicial de conexión, se intercambian paquetes que revelan detalles fundamentales sobre la implementación del protocolo SSH. Estos paquetes incluyen información crítica como la identificación de la versión del protocolo y el software específico que se utilizará para la comunicación. En un caso particular, se identificó una configuración con **SSH versión 2.0, utilizando OpenSSH 8.9p1** en un entorno **Ubuntu-3ubuntu0.10**, lo que permite una negociación precisa entre el cliente y el servidor, tal como se aprecia en la imagen de continuación.

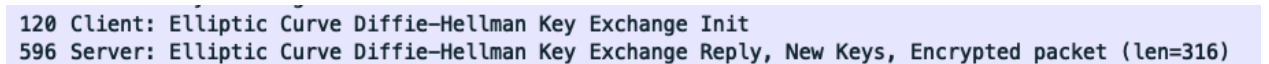


The image shows a network traffic capture with two lines of data. The first line is highlighted in blue and reads: "114 Client: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)". The second line is on a light blue background and reads: "72 ssh(22) → 58952 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=25679865".

Figura 58: SSH y OpenSSH Client C4.

El proceso de conexión implica un intercambio detallado que define los parámetros de comunicación, especialmente los algoritmos compatibles. Se negocian aspectos como los métodos de cifrado, compresión e intercambio de claves. A modo de ejemplo, entre las opciones de cifrado se puede encontrar el algoritmo aes256-ctr. Esta información se transmite en formato de texto plano, revelando únicamente las capacidades de comunicación sin comprometer ninguna información sensible.

2. Fase de Intercambio Criptográfico: Los paquetes contienen los parámetros y los métodos de intercambio de claves propuestos para realizar la conexión. En este caso, también se muestra el algoritmo Diffie-Hellman. La información de estos paquetes va en texto plano debido a que son las instrucciones que se realizan para generar una clave de sesión compartida y segura, tal como se muestra en la siguiente figura.



The image shows a network traffic capture with two lines of data. The first line is highlighted in blue and reads: "120 Client: Elliptic Curve Diffie-Hellman Key Exchange Init". The second line is on a light blue background and reads: "596 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)".

Figura 59: Intercambio Algoritmo Diffie Hellman.

3. Proceso de Verificación de Identidad (Autenticación): En las sesiones que exigen autenticación mediante contraseña, es posible interceptar mensajes que incluyen una solicitud de autenticación, donde se muestra el nombre del usuario que intenta establecer la conexión en texto sin cifrar. No obstante, por razones de seguridad, la contraseña no se transmite en texto claro, ya que SSH se encarga de encriptar los datos de autenticación.

1.9. Diferencia entre C1 y C2

Las diferencias observadas en las capturas realizadas para los clientes C1 y C2 radican en las versiones de OpenSSH utilizadas y en cómo estas manejan los tamaños de los paquetes generados. En el caso de **C1**, que utiliza la versión **OpenSSH 7.2p2**, los paquetes de intercambio de claves tienen **tamaños de 1402 y 1178 bytes**. Por otro lado, **C2**, que emplea la versión **OpenSSH 7.6p1**, genera paquetes de intercambio de claves con **tamaños de 1426 y 1178 bytes**, evidenciando una reducción en el tamaño del paquete KEI en el cliente.

1.10. Diferencia entre C2 y C3

Las diferencias observadas en las capturas realizadas para los clientes C1 y C2 radican en las versiones de OpenSSH utilizadas y en cómo estas manejan los tamaños de los paquetes generados. En el caso de **C3**, que utiliza la versión **OpenSSH 8.2p1**, los paquetes de intercambio de claves tienen **tamaños de 1602 y 1178 bytes**. Por otro lado, **C2**, que emplea la versión **OpenSSH 7.6p1**, genera paquetes de intercambio de claves con **tamaños de 1426 y 1178 bytes**, evidenciando una reducción en el tamaño del paquete KEI en el cliente.

1.11. Diferencia entre C3 y C4

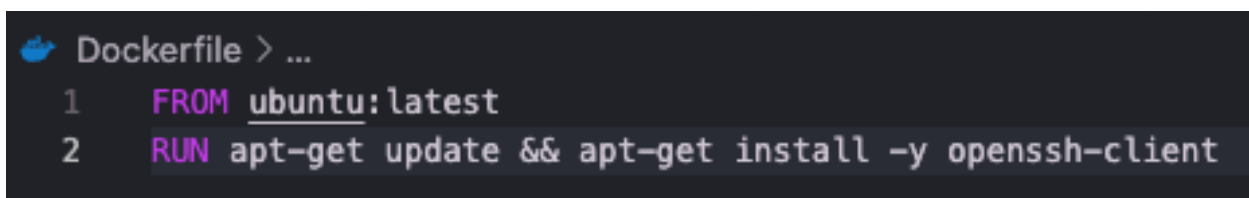
Las diferencias observadas en las capturas realizadas para los clientes C1 y C2 radican en las versiones de OpenSSH utilizadas y en cómo estas manejan los tamaños de los paquetes generados. En el caso de **C3**, que utiliza la versión **OpenSSH 8.2p1**, los paquetes de intercambio de claves tienen **tamaños de 1602 y 1178 bytes**. Por otro lado, **C4**, que emplea la versión **OpenSSH 8.9p1**, genera paquetes de intercambio de claves con **tamaños de 1608 y 1184 bytes**, evidenciando una reducción en el tamaño del paquete KEI en el cliente.

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión “?”

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

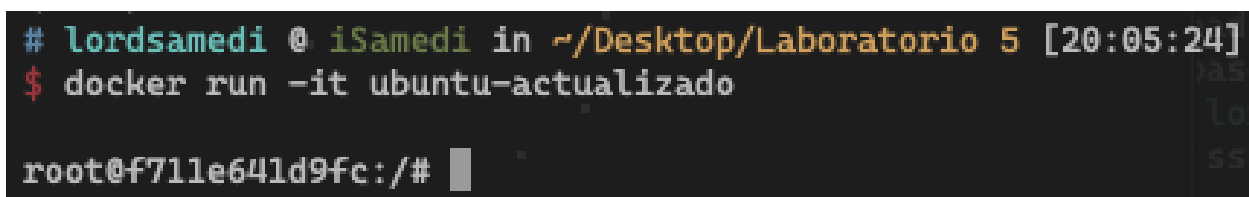
Para realizar la replica del tráfico al servidor, se creo un nuevo dockerfile que contendra la ultima versión de Ubuntu hasta el momento, siendo la versión 24.10 a fecha de hoy. A continuación se mostrara el dockerfile a utilizar.



```
Dockerfile > ...  
1 FROM ubuntu:latest  
2 RUN apt-get update && apt-get install -y openssh-client
```

Figura 60: Versión actual de Ubuntu.

Posteriormente se realizo un docker build y docker run para crear y ejecutar el contenedor de Ubuntu.



```
# lordsamedi @ iSamedi in ~/Desktop/Laboratorio 5 [20:05:24]  
$ docker run -it ubuntu-actualizado  
  
root@f711e641d9fc:/#
```

Figura 61: Ejecución contenedor de ubuntu-actualizado.

Cabe mencionar, el fin de esta replica, es simular la ejecución dentro de un sistema operativo principal de Ubuntu con su ultima versión, ya que, en mi caso personal, utilizando el sistema operativo MacOS de Apple se dificulta realizar una conexión directa con los contenedores de Docker.

```
root@853ef134f77e:/# ssh prueba@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.
ED25519 key fingerprint is SHA256:N9S+5cSWj1UM0rpq8geDDu935aK7YnI7k02ysb65yT8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.17.0.5' (ED25519) to the list of known hosts.
prueba@172.17.0.5's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.10.4-linuxkit aarch64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Fri Dec  6 23:16:57 2024 from 172.17.0.6
$
```

Figura 62: Conexión desde Cliente Ubuntu-Latest.

Este paso se realizó paralelamente con el paso de captura del tráfico con **tcpdump** ejecutado en el contenedor C4 e iniciando **SSH prueba@172.17.0.5** en el contenedor Ubuntu-Latest, colocando la contraseña **prueba** para poder acceder al servidor y finalmente finalizar la captura de tráfico con las teclas **CTL+C**.

El tráfico generado anteriormente fue el siguiente.

Source	Destination	Protocol	Length	Info
172.17.0.7	172.17.0.5	TCP	80	39316 → ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=728785727 TSecr=0 WS=128
172.17.0.5	172.17.0.7	TCP	80	ssh(22) → 39316 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3240034506 TSecr=
172.17.0.7	172.17.0.5	TCP	72	39316 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=728785728 TSecr=3240034506
172.17.0.7	172.17.0.5	SSHv2	114	Client: Protocol (SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5)
172.17.0.5	172.17.0.7	TCP	72	ssh(22) → 39316 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=3240034507 TSecr=728785729
172.17.0.5	172.17.0.7	SSHv2	114	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)
172.17.0.7	172.17.0.5	TCP	72	39316 → ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSval=728785748 TSecr=3240034526
172.17.0.7	172.17.0.5	SSHv2	1608	Client: Key Exchange Init
172.17.0.5	172.17.0.7	SSHv2	1184	Server: Key Exchange Init
172.17.0.7	172.17.0.5	SSHv2	1280	Client: Diffie-Hellman Key Exchange Init
172.17.0.5	172.17.0.7	SSHv2	1636	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)
172.17.0.7	172.17.0.5	TCP	72	39316 → ssh(22) [ACK] Seq=2787 Ack=2719 Win=65536 Len=0 TSval=728785841 TSecr=3240034577
172.17.0.7	172.17.0.5	SSHv2	88	Client: New Keys
172.17.0.5	172.17.0.7	TCP	72	ssh(22) → 39316 [ACK] Seq=2719 Ack=2803 Win=65536 Len=0 TSval=3240036868 TSecr=728788044
172.17.0.7	172.17.0.5	SSHv2	116	Client: Encrypted packet (len=44)
172.17.0.5	172.17.0.7	TCP	72	ssh(22) → 39316 [ACK] Seq=2719 Ack=2847 Win=65536 Len=0 TSval=3240036868 TSecr=728788090
172.17.0.5	172.17.0.7	SSHv2	116	Server: Encrypted packet (len=44)
172.17.0.7	172.17.0.5	TCP	72	39316 → ssh(22) [ACK] Seq=2847 Ack=2763 Win=65536 Len=0 TSval=728788090 TSecr=3240036868
172.17.0.7	172.17.0.5	SSHv2	140	Client: Encrypted packet (len=68)
172.17.0.5	172.17.0.7	SSHv2	124	Server: Encrypted packet (len=52)
172.17.0.7	172.17.0.5	TCP	72	39316 → ssh(22) [ACK] Seq=2915 Ack=2815 Win=65536 Len=0 TSval=728788138 TSecr=3240036875
172.17.0.7	172.17.0.5	SSHv2	228	Client: Encrypted packet (len=148)
172.17.0.5	172.17.0.7	TCP	72	ssh(22) → 39316 [ACK] Seq=2815 Ack=3063 Win=65536 Len=0 TSval=3240039290 TSecr=728790470
172.17.0.5	172.17.0.7	SSHv2	108	Server: Encrypted packet (len=28)
172.17.0.7	172.17.0.5	TCP	72	39316 → ssh(22) [ACK] Seq=3063 Ack=2843 Win=65536 Len=0 TSval=728790557 TSecr=3240039335
172.17.0.7	172.17.0.5	SSHv2	184	Client: Encrypted packet (len=112)
172.17.0.5	172.17.0.7	TCP	72	ssh(22) → 39316 [ACK] Seq=2843 Ack=3175 Win=65536 Len=0 TSval=3240039336 TSecr=728790557
172.17.0.5	172.17.0.7	SSHv2	700	Server: Encrypted packet (len=628)
172.17.0.7	172.17.0.5	SSHv2	652	Client: Encrypted packet (len=580)
172.17.0.5	172.17.0.7	SSHv2	116	Server: Encrypted packet (len=44)

Figura 63: Trafico de datos entre UbuntuLatest y C4/S1.

Visualizando el trafico de paquetes capturado, observamos que se llevo a cabo correctamente la comunicación entre Cliente y Servidor utilizando Diffie Hellman para la creación de una NewKeys.

SSHv2	1608	Client: Key Exchange Init
SSHv2	1184	Server: Key Exchange Init
SSHv2	1280	Client: Diffie-Hellman Key Exchange Init
SSHv2	1636	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=316)

Figura 64: Enter Caption

En la información del Source es quien inicia la conexión es el equipo **cliente** con IP **172.17.0.7** dirigido al destination quien es el **servidor** IP **172.17.0.5**.

Luego, el cliente realiza la acción de identificarse con un tamaño de 1608 bytes. Si examinamos el SSH Protocol, nos encontraremos con lo siguiente.

SSH Protocol
Protocol: SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5
[Direction: client-to-server]

Figura 65: SSH Protocol Client.

En el contenido del protocolo de la figura 34, nos indica que es un **protocolo SSH 2.0** y aparte de eso nos da información de la versión del sistema operativo, que en este caso es **Ubuntu-3ubuntu13.5** en el cual se esta corriendo siendo esta la ultima versión, con una dirección de cliente a servidor. Luego el paquete siguiente es un ACK, que significa que el servidor esta reconociendo el paquete del servidor.

Por otro lado la información de SSH Protocol Server se obtiene lo siguiente. El servidor

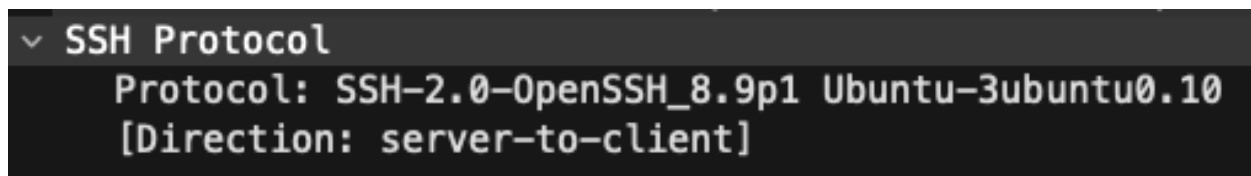


Figura 66: SSH Protocol Server.

se identifica con una versión de SSH 2.0 y con una versión distinta de Ubuntu a diferencia del cliente siendo la Ubuntu-3ubuntu0.10, una versión vieja pero no descontinuada, esto se debe a que cada apartado se está corriendo con versiones diferentes de esta.

Posteriormente, el cliente reconoce el paquete ACK. Luego el servidor comienza el intercambio de claves hacia el cliente, tal como se muestra en la siguiente figura.

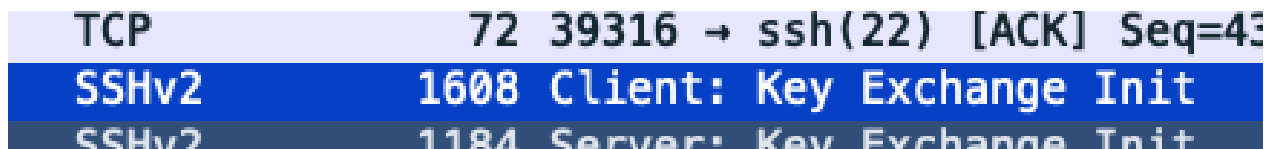


Figura 67: Intercambio de claves Cliente KEL.

El paquete llamado Client: Key Exchange Init posee un tamaño de 1608 bytes, la información en su SSH Protocol es la siguiente.


```

  Algorithms
    Cookie: 42528ba9c0d9f21553f5bea6762ef318
    kex_algorithms length: 305
    kex_algorithms string [...]: sntrup761x25519-sha512@openssh.com,curve25519-s
    server_host_key_algorithms length: 463
    server host key algorithms string [...]: ssh-ed25519-cert-v01@openssh.com,ec
    encryption_algorithms_client_to_server length: 108
    encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.c
    encryption_algorithms_server_to_client length: 108
    encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.c
    mac_algorithms_client_to_server length: 213
    mac_algorithms_client_to_server string [...]: umac-64-etm@openssh.com,umac-1
    mac_algorithms_server_to_client length: 213
    mac_algorithms_server_to_client string [...]: umac-64-etm@openssh.com,umac-1
    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
  
```

Figura 68: SSH Protocol Client KEI.

- En el apartado de Algorithms, en el recuadro rojo que se aprecia en la imagen, corresponde a los KEX o Key Exchange poseen un tamaño de 305 bytes, donde posee una serie de algoritmos que sirven el intercambio de claves. Estos algoritmos son los siguientes:

- 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,
- diff...Entre Otros.

- Luego en el recuadro amarillo, se encuentran los algoritmos de encriptación de cliente a servidor y de servidor a cliente, identificandose el algoritmo y la modalidad, por ejemplo algoritmo AES-128 con modalidad CTR. Estos algoritmos son los siguientes (son identicos para Cliente a Server y de Server a Cliente) con tamaño 108 bytes:
 - chacha20-poly1305@openssh.com,
 - aes128-ctr,
 - aes192-ctr,
 - aes256-ctr,
 - aes128-gcm@openssh.com,
 - aes256-gcm@openssh.com
- Tambien se tiene algoritmos Mac, como se aprecia en el recuadro morado. MAC quiere decir Message Authetication Code, por ende, aquí se contiene todos los algoritmos que nos van a servir para comprobar la integridad de cada paquete, ya sea de cliente a servidor y servidor a cliente, ambos con un tamaño de 213 bytes (son identicos para Cliente a Server y de Server a Cliente):
 - 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,
 - hma... Entre otros...
- Finalmente, en el recuadro azul, se encuentran los algoritmos de compresión SSH para lograr una mayor eficiencia y para generar complejidad y confundir todavía más el contenido de los paquetes, teniendo la opción de comprimir el contenido utilizando el algoritmo de **zlib** par ambos casos con tamaño de 26 bytes.

Por otro lado, los algoritmos de SSH Protocol del servidor son los mismos anteriormente explicados.

Luego, comienza el intercambio Diffie Hellman entre Cliente y Servidor, y finalmente el cliente y servidor poseen lo necesario para generar la nueva clave de cifrado, por ende, el cliente envía el siguiente mensaje. Y a partir de este momento, todo el resto del contenido es cifrado apareciendo en el tráfico como **"Encrypted request/response packet"**.

Para terminar, igual que anteriormente, obtendremos el HASSH respectivo, volviendo a utilizar la herramienta en línea anterior, especializada en el cálculo de hash MD5, cuyo resultado se encuentra en el paquete KEI del cliente ubuntu-latest, obteniendo lo siguiente.

Your String	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,diff	
MD5 Hash	96b87f0b8aa73047f364d8c714c00610	<input type="button" value="Copy"/>

Figura 69: HASSH Respectivo de la replica.

A partir de la imagen podemos asumir que el HASSH asociado es:

- 96b87f0b8aa73047f364d8c714c00610

Cabe destacar que estos algoritmos, la mayoría, son distintos a la de los contenedores anteriores a lo largo del desarrollo del laboratorio, esto se debe ya que, al ser una versión más actualizada que las anteriores, esto se debe a que en las versiones iniciales de Ubuntu, los algoritmos de intercambio de claves se basaban en implementaciones tradicionales de Diffie-Hellman. Estos métodos utilizaban grupos de claves más pequeños y presentaban vulnerabilidades inherentes a los avances computacionales de la época.

3. Desarrollo (Parte 3)

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

En este apartado se llevó a cabo la optimización de algoritmos para disminuir el tamaño del paquete Server Key Exchange Init a 300 bytes. Para ello, se tuvo que ir a `/etc/ssh/sshd_config` dentro del servidor C4/S1, donde se añadió lo siguiente.

```
18
19 KexAlgorithms curve25519-sha256@libssh.org
20 Ciphers aes128-ctr
21 MACs hmac-sha2-256
22 Compression no
23 RekeyLimit 256K 1h
24
25 HostKey /etc/ssh/ssh_host_ed25519_key
26
```

Figura 70: Modificación en el Servidor C4/S1.

- ***KexAlgorithms curve25519-sha256***: Define los algoritmos utilizados para el *intercambio de claves* entre el cliente y el servidor. Este algoritmo permite establecer una clave secreta compartida de manera segura incluso si la conexión es interceptada.
 - **curve25519-sha256**:
 - Es más eficiente y compacto que otros algoritmos como Diffie-Hellman.
 - Genera paquetes más pequeños, contribuyendo directamente a reducir el tamaño del **KEI**
- ***Ciphers aes128-ctr***: Especifica los algoritmos de cifrado que protegen los datos transmitidos después del intercambio de claves. Garantiza que los datos enviados entre el cliente y el servidor estén encriptados para evitar que terceros los lean.
 - **aes128-ctr**:
 - Es un algoritmo seguro y ligero, con un buen balance entre velocidad y seguridad.
 - Usar un cifrado más liviano como AES-128 en lugar de AES-256 reduce el tamaño de los datos intercambiados en el KEI.
- ***MACs hmac-sha2-256***: Define los algoritmos de *Message Authentication Code (MAC)* usados para garantizar la integridad de los datos. Y asegura que los datos no sean alterados durante la transmisión.
 - **hmac-sha2-256**:

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

- Es compacto y seguro, lo que ayuda a reducir el tamaño del KEI.
- **Compression no:** Indica si se debe comprimir la información transmitida entre cliente y servidor. En este caso, reduce el tamaño del paquete inicial eliminando datos innecesarios.
- **HostKey /etc/ssh/ssh_host_ed25519 key:** Es utilizada en la configuración del servidor SSH para definir la clave privada del host, específicamente usando el algoritmo de firma digital Ed25519.

Una vez explicado los cambios realizados en el servidor, se explicaran los cambios de ajuste por parte del cliente, utilizando el siguiente comando.

```
root@baf5c6c32b0a:/# ssh -o KexAlgorithms=curve25519-sha256 \
-o Ciphers=aes128-ctr \
-o MACs= hmac-sha2-256 \
prueba@172.17.0.5
The authenticity of host '172.17.0.5 (172.17.0.5)' can't be established.
ED25519 key fingerprint is SHA256:N9S+5cSWj1UM0rpq8geDDu935aK7YnI7k02ysb65yT8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.17.0.5' (ED25519) to the list of known hosts.
prueba@172.17.0.5's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.10.4-linuxkit aarch64)
```

Figura 71: Especificar Algoritmo en el cliente.

En la figura anterior se especifican los algoritmos en el cliente para obligar al cliente a utilizar los algoritmos descritos anteriormente y especificados en la figura 71, ya que, aunque el servidor puede estar configurado para usar ciertos algoritmos, si el cliente no los solicita específicamente, puede haber un conflicto en la negociación. Por otro lado, definir explícitamente los algoritmos asegura que el cliente se comunique de forma eficiente con el servidor.

Finalmente se realizo la captura del trafico con tcpdump dentro del contenedor C4 para realizar una captura directa dentro del servidor.

```
root@25c24a9a0248:/# tcpdump -i any port 22 -w /tmp/prueba.pcap
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
^C37 packets captured
37 packets received by filter
0 packets dropped by kernel
```

Figura 72: Captura de trafico en el contenedor C4.

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

Una vez ingresado al servidor por parte del cliente, ingresando la contraseña *prueba*, se finalizó la captura del tráfico con las teclas **CTL+C**, obteniendo un archivo llamado *prueba.pcap* para posteriormente ser ejecutado por Wireshark, obteniendo lo siguiente.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.17.0.12	172.17.0.5	TCP	80	51946 → ssh(22) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3304552170 TSecr=
2	0.000055	172.17.0.5	172.17.0.12	TCP	80	ssh(22) → 51946 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=849632963
3	0.000091	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3304552171 TSecr=849632963
4	0.002260	172.17.0.12	172.17.0.5	SSHv2	114	Client: Protocol (SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5)
5	0.002265	172.17.0.5	172.17.0.12	TCP	72	ssh(22) → 51946 [ACK] Seq=1 Ack=43 Win=65536 Len=0 TSval=849632965 TSecr=3304552173
6	0.023785	172.17.0.5	172.17.0.12	SSHv2	114	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)
7	0.023888	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=43 Ack=43 Win=65536 Len=0 TSval=3304552195 TSecr=849632987
8	0.024172	172.17.0.12	172.17.0.5	SSHv2	768	Client: Key Exchange Init
9	0.028547	172.17.0.5	172.17.0.12	SSHv2	336	Server: Key Exchange Init
10	0.030001	172.17.0.12	172.17.0.5	SSHv2	120	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11	0.034157	172.17.0.5	172.17.0.12	SSHv2	616	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (
12	0.075320	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=787 Ack=851 Win=65024 Len=0 TSval=3304552246 TSecr=849632997
13	1.981927	172.17.0.12	172.17.0.5	SSHv2	88	Client: New Keys
14	2.022964	172.17.0.5	172.17.0.12	TCP	72	ssh(22) → 51946 [ACK] Seq=851 Ack=883 Win=64896 Len=0 TSval=849634986 TSecr=3304554153
15	2.023129	172.17.0.12	172.17.0.5	SSHv2	136	Client: Encrypted packet (len=64)
16	2.023157	172.17.0.5	172.17.0.12	TCP	72	ssh(22) → 51946 [ACK] Seq=851 Ack=867 Win=64896 Len=0 TSval=849634986 TSecr=3304554194
17	2.023259	172.17.0.5	172.17.0.12	SSHv2	136	Server: Encrypted packet (len=64)
18	2.023300	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=867 Ack=915 Win=65024 Len=0 TSval=3304554194 TSecr=849634986
19	2.023400	172.17.0.12	172.17.0.5	SSHv2	152	Client: Encrypted packet (len=80)
20	2.033837	172.17.0.5	172.17.0.12	SSHv2	152	Server: Encrypted packet (len=80)
21	2.075344	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=947 Ack=995 Win=65024 Len=0 TSval=3304554246 TSecr=849634997
22	6.752185	172.17.0.12	172.17.0.5	SSHv2	232	Client: Encrypted packet (len=160)
23	6.792568	172.17.0.5	172.17.0.12	TCP	72	ssh(22) → 51946 [ACK] Seq=995 Ack=1107 Win=64768 Len=0 TSval=849639756 TSecr=330455892
24	6.824685	172.17.0.5	172.17.0.12	SSHv2	120	Server: Encrypted packet (len=48)
25	6.824755	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=1107 Ack=1043 Win=65024 Len=0 TSval=3304558996 TSecr=8496397
26	6.825036	172.17.0.12	172.17.0.5	SSHv2	232	Client: Encrypted packet (len=160)
27	6.825047	172.17.0.5	172.17.0.12	TCP	72	ssh(22) → 51946 [ACK] Seq=1043 Ack=1267 Win=64640 Len=0 TSval=849639788 TSecr=33045589
28	6.840996	172.17.0.5	172.17.0.12	SSHv2	728	Server: Encrypted packet (len=656)
29	6.841581	172.17.0.12	172.17.0.5	SSHv2	680	Client: Encrypted packet (len=608)
30	6.841514	172.17.0.5	172.17.0.12	SSHv2	136	Server: Encrypted packet (len=64)
31	6.841577	172.17.0.12	172.17.0.5	SSHv2	488	Client: Encrypted packet (len=416)
32	6.845758	172.17.0.5	172.17.0.12	SSHv2	632	Server: Encrypted packet (len=560)
33	6.847191	172.17.0.5	172.17.0.12	SSHv2	232	Server: Encrypted packet (len=160)
34	6.847296	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=2291 Ack=2483 Win=63744 Len=0 TSval=3304559018 TSecr=8496398
35	6.847315	172.17.0.5	172.17.0.12	SSHv2	584	Server: Encrypted packet (len=512)
36	6.849960	172.17.0.5	172.17.0.12	SSHv2	136	Server: Encrypted packet (len=64)
37	6.850000	172.17.0.12	172.17.0.5	TCP	72	51946 → ssh(22) [ACK] Seq=2483 Ack=2500 Win=63744 Len=0 TSval=3304559030 TSecr=8496398

Figura 73: Captura prueba.pcap.

Despues de implementar la configuraciones y capturar el tráfico, nos dirigimos a observar el tamaño del paquete **Server Key Exchange Init (KEI)**, visualizando lo siguiente.

TCP	72	ssh(22) → 51946 [ACK] Seq=1 Ack=43
SSHv2	114	Server: Protocol (SSH-2.0-OpenSSH_8
TCP	72	51946 → ssh(22) [ACK] Seq=43 Ack=43
SSHv2	768	Client: Key Exchange Init
SSHv2	336	Server: Key Exchange Init
SSHv2	120	Client: Elliptic Curve Diffie-Hellm
SSHv2	616	Server: Elliptic Curve Diffie-Hellm
TCP	72	51946 → ssh(22) [ACK] Seq=787 Ack=8
SSHv2	88	Client: New Keys

Figura 74: Paquete Server KEI reducido de tamaño.

El tamaño final del paquete Server KEI es de **336 bytes**, a diferencia de los iniciales con un promedio de tamaño de 1200 bytes. **Esto se logró al limitar los algoritmos de negociación, desactivar la compresión y elegir algoritmos más ligeros y eficientes.**

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es una herramienta tecnológica que representa una solución integral para conexiones de red seguras en entornos digitales vulnerables. Implementando el protocolo Secure Shell, OpenSSH ofrece un sistema de comunicación que protege la transmisión de información mediante técnicas criptográficas avanzadas. Su diseño se fundamenta en un modelo de interacción cliente-servidor que asegura la autenticación y el intercambio confidencial de datos.

La función fundamental de OpenSSH se centra en prevenir accesos no autorizados y mitigar riesgos de interceptación digital, utilizando complejos algoritmos de cifrado. Entre sus funcionalidades más relevantes destacan la conexión remota, la transferencia segura de archivos y la ejecución de comandos en sistemas distantes.

El protocolo utiliza métodos de encriptación sofisticados como AES, complementados con sistemas dinámicos de generación de claves que garantizan la unicidad de cada sesión de comunicación. La principal característica de OpenSSH es su capacidad para convertir redes potencialmente inseguras en canales de comunicación protegidos, resguardando la confidencialidad e integridad de la información transmitida entre diferentes plataformas informáticas.

4.2. Capas de Seguridad en OpenSSH

OpenSSH incorpora múltiples capas de seguridad para proteger las comunicaciones de red:

1. **Capa de Autenticación:** La capa de autenticación de OpenSSH implementa mecanismos de verificación de identidad múltiples y robustos. Esta capa garantiza que solo los usuarios autorizados puedan establecer conexiones, mediante métodos como:

- Autenticación por contraseña
- Autenticación mediante par de claves pública/privada
- Autenticación con certificados digitales
- Métodos de autenticación de doble factor

2. **Capa de Cifrado:** Tiene como objetivo principal proteger la confidencialidad de la información transmitida. Utiliza algoritmos de encriptación de alta complejidad que aseguran:

- Transformación de datos originales en contenido ilegible
- Generación de claves de sesión únicas
- Uso de estándares de cifrado como AES
- Implementación de algoritmos simétricos y asimétricos

- Protección contra interceptación y lectura de datos

3. **Capa de Integridad de Datos:** Diseñada para detectar cualquier manipulación o alteración de la información durante su transmisión. Sus características principales incluyen:

- Implementación de funciones hash criptográficas
- Uso de algoritmos HMAC (Hash-based Message Authentication Code)
- Verificación de la autenticidad de los paquetes transmitidos
- Detección inmediata de modificaciones no autorizadas

4. **Capa de Gestión de Conexiones:** Controla y supervisa las sesiones de conexión remota, implementando:

- Límites de tiempo de conexión
- Políticas de acceso configurables
- Registro detallado de actividades
- Mecanismos de auditoría de conexiones
- Control de intentos de acceso

5. **Capa de Reenvío Seguro:** Proporciona mecanismos avanzados para establecer conexiones seguras a través de redes no confiables:

- Creación de túneles SSH
- Reenvío de puertos de manera segura
- Protección de protocolos adicionales
- Encapsulamiento de tráfico de red

Cabe mencionar, que las cinco capas de seguridad de OpenSSH trabajan de manera integrada y sincronizada, proporcionando un sistema de comunicación robusto, flexible y altamente seguro para entornos de red complejos.

4.3. Identificación de que protocolos no se cumplen

A partir del análisis exhaustivo del tráfico SSH interceptado durante las sesiones de laboratorio, se puede evaluar detalladamente el grado de cumplimiento de los principios de seguridad informática. Este análisis permite identificar tanto las fortalezas del protocolo como las áreas donde no se satisfacen completamente los requisitos de seguridad. A continuación, se exponen las observaciones específicas:

- **Confidencialidad:** Este principio se cumple de manera completa. La comunicación posterior al intercambio de claves está íntegramente cifrada, utilizando algoritmos avanzados que impiden el acceso a la información por parte de terceros no autorizados. Este nivel de cifrado asegura que los datos sensibles permanezcan protegidos, incluso si el tráfico es interceptado.
- **No repudio:** Este es el único principio de seguridad que no se satisface de manera inherente en el protocolo SSH. Las conexiones estándar no utilizan mecanismos como firmas digitales para garantizar que una entidad no pueda negar haber realizado una acción específica. Esto significa que, en su implementación base, el protocolo no proporciona las herramientas necesarias para prevenir el repudio. Si este principio es un requisito esencial, será indispensable implementar soluciones complementarias, como registros de auditoría o el uso de certificados digitales firmados.
- **Autenticidad:** Este principio se cumple de manera robusta. El intercambio de claves Diffie-Hellman proporciona una verificación mutua entre cliente y servidor, asegurando que ambas partes son legítimas. Adicionalmente, la autenticación mediante huellas digitales del servidor refuerza la confianza en la identidad del servidor, previniendo ataques de suplantación de identidad.
- **Integridad:** El cumplimiento de este principio está plenamente demostrado. El tráfico analizado incorpora el uso de HMAC (Código de Autenticación de Mensajes Basado en Hash) como mecanismo para validar que los datos transmitidos no han sido alterados. Este método asegura la confiabilidad del mensaje recibido, garantizando que no haya ocurrido ninguna manipulación durante su tránsito.
- **Disponibilidad:** Aunque el protocolo SSH incluye herramientas diseñadas para garantizar la disponibilidad del servicio, este principio se considera parcialmente cumplido en el contexto del tráfico analizado. Las contramedidas contra ataques de denegación de servicio (DoS) implementadas por OpenSSH, tales como la gestión de límites de conexión o tiempos de espera, no pudieron verificarse directamente en los datos examinados. Esto sugiere que la disponibilidad dependerá en gran medida de las configuraciones específicas del servidor, y no de una funcionalidad inherente del protocolo en su configuración por defecto.

Conclusiones y comentarios

El desarrollo de este laboratorio permitió una exploración exhaustiva del funcionamiento y las capacidades de OpenSSH en un entorno virtualizado con Docker. A través de la implementación y análisis práctico de diferentes versiones de OpenSSH, se lograron valiosas observaciones sobre su desempeño y sobre los principios fundamentales de seguridad de la información. Las conclusiones más relevantes son las siguientes:

- **Impacto de las versiones de OpenSSH:** Se verificó que las distintas versiones de OpenSSH tienen un impacto significativo en el tráfico generado y en los algoritmos soportados, destacando la importancia de mantener versiones actualizadas para mitigar vulnerabilidades y garantizar mayor seguridad.
- **Optimización en la negociación de claves:** El análisis detallado del intercambio de claves reveló que es posible reducir el tamaño de los paquetes KEI mediante una adecuada selección de algoritmos, optimizando la eficiencia sin comprometer la seguridad.
- **Principios de seguridad:** Se observó cómo OpenSSH implementa los principios de confidencialidad, integridad y autenticidad a través de algoritmos robustos y eficientes. Sin embargo, se identificaron áreas de mejora, como la implementación del principio de no repudio de forma nativa.
- **Uso de entornos virtualizados:** La utilización de Docker facilitó la configuración y análisis de múltiples escenarios en un entorno controlado, lo que permitió identificar diferencias clave entre versiones y optimizar configuraciones para distintas necesidades.

En conclusión, el laboratorio no solo permitió comprender los fundamentos y configuraciones de OpenSSH, sino que también destacó su capacidad para garantizar comunicaciones seguras y su flexibilidad en distintos entornos. Este ejercicio refuerza la relevancia de los protocolos criptográficos en la protección de redes modernas y subraya la necesidad de seguir actualizando y afinando estas herramientas frente a los desafíos de ciberseguridad.