Informe Laboratorio 5

Sección 3

Vicente Silva Almarza
e-mail: vicente.silva5@mail.udp.cl

Noviembre de 2025

Índice

Descripción de actividades

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

Para lo anterior deberá:

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

Pasos:

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

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

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

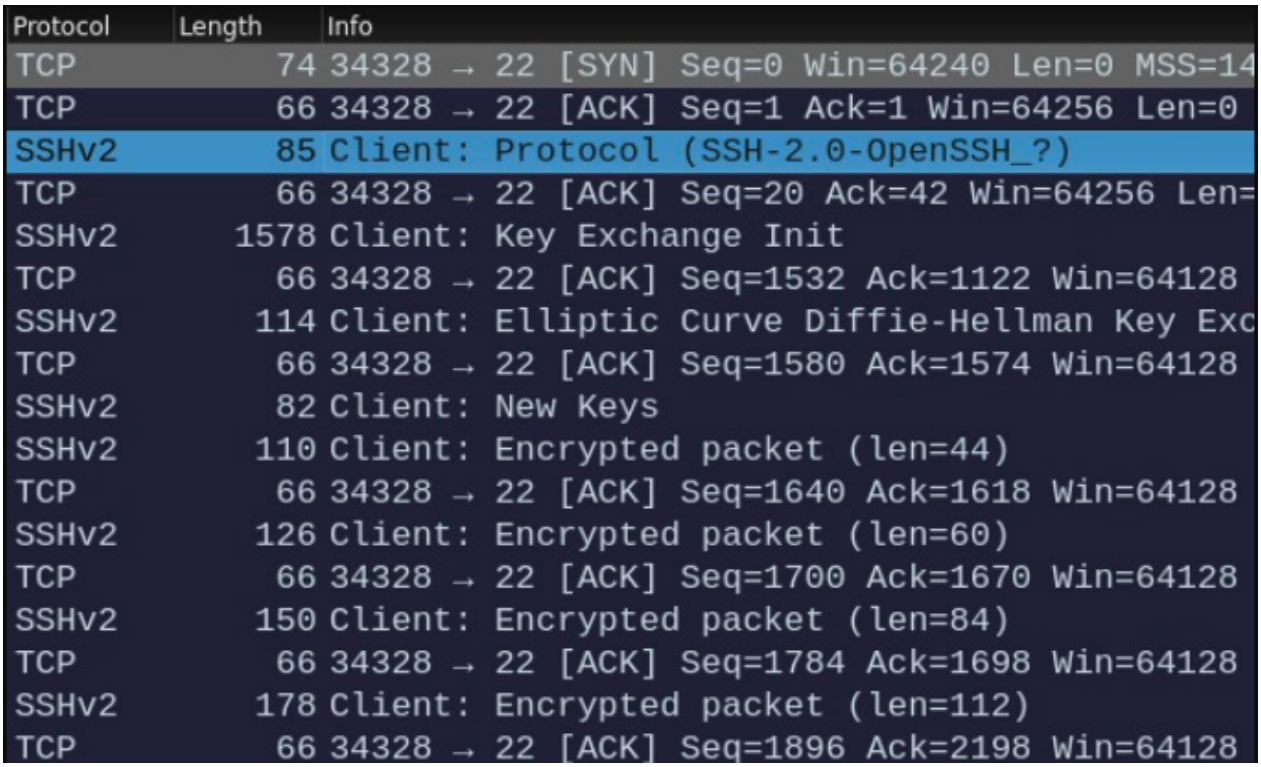


Figura 1: Tráfico generado del informante

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

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

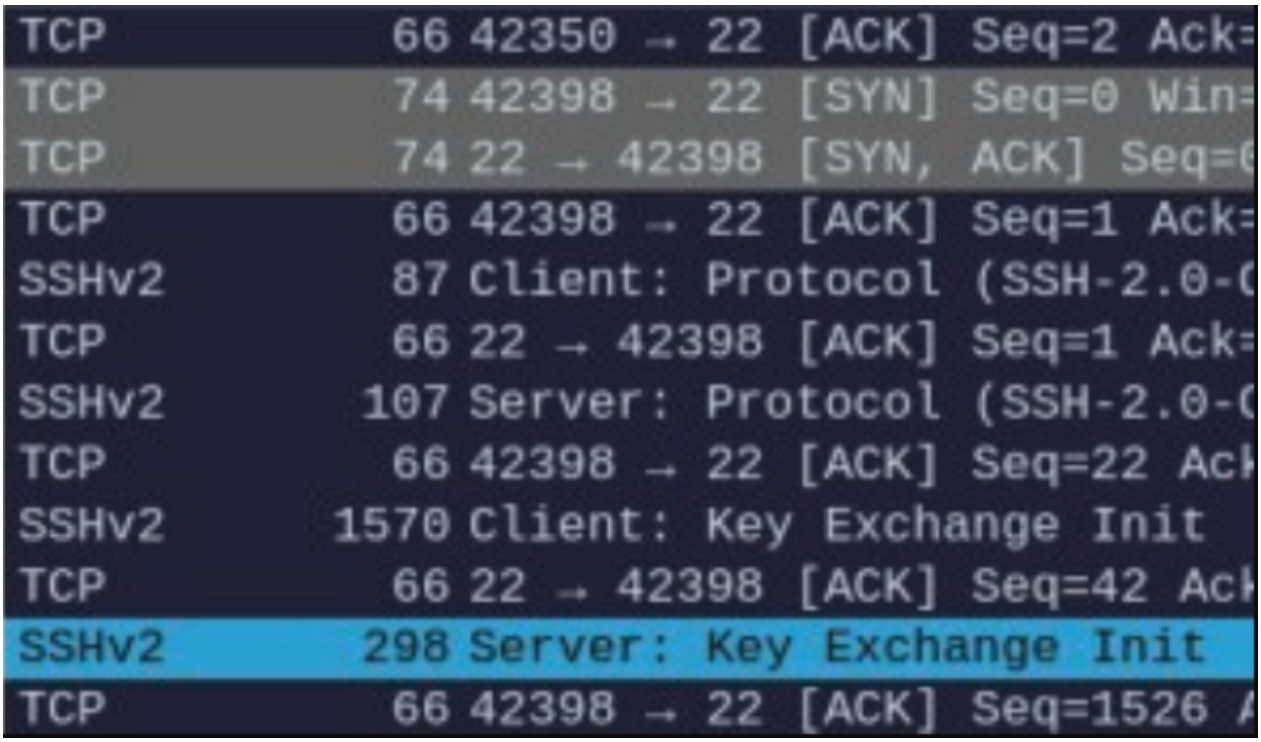


Figura 2: Captura del Key Exchange

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

1. Desarrollo (Parte 1)

1.1. Códigos de cada Dockerfile

Se crean los dockerfile de cada máquina, con los Os solicitados. Cada uno de estos redirigirá a sus capturas .pcap y se levantarán mediante un docker-compose.

1.1.1. C1

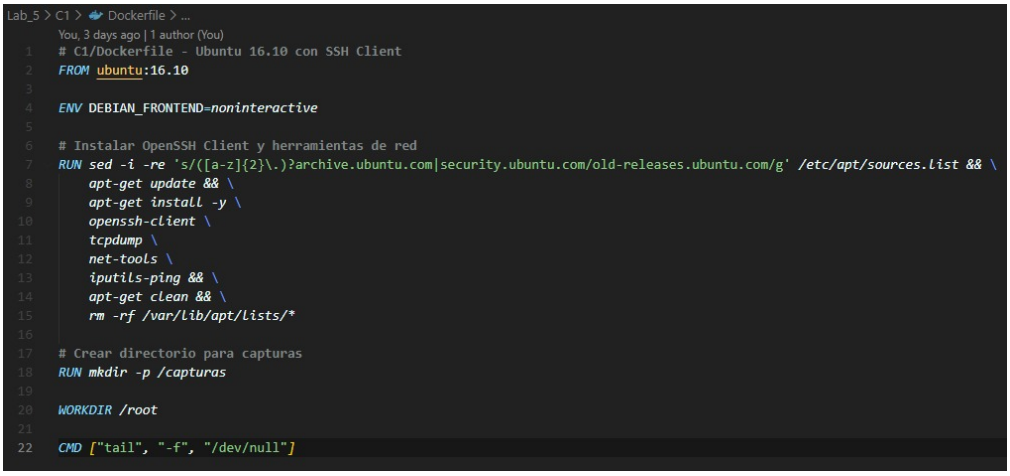


Figura 3: Dockerfile de C1 con el OS: Ubuntu 16.04.

1.1.2. C2

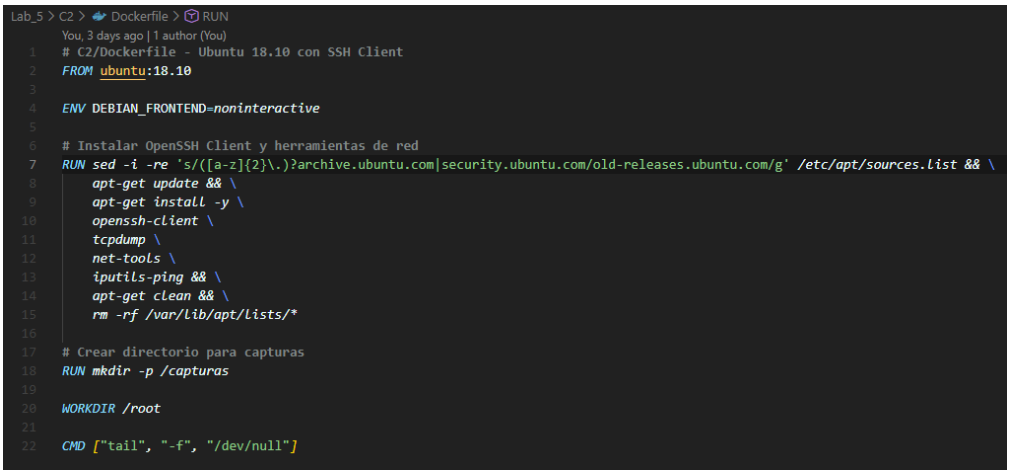


Figura 4: Dockerfile de C2 con el OS: Ubuntu 18.04.

1.1.3. C3

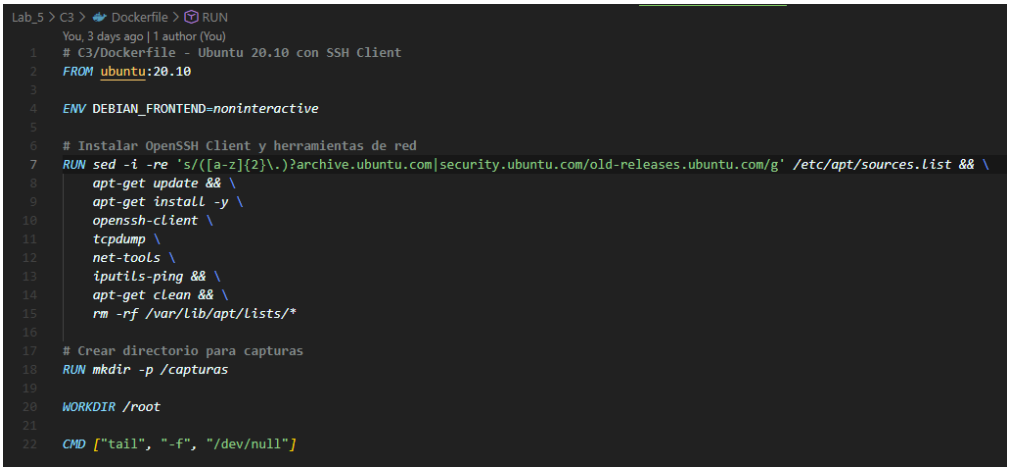


Figura 5: Dockerfile de C1 con el OS: Ubuntu 20.04.

1.1.4. C4/S1

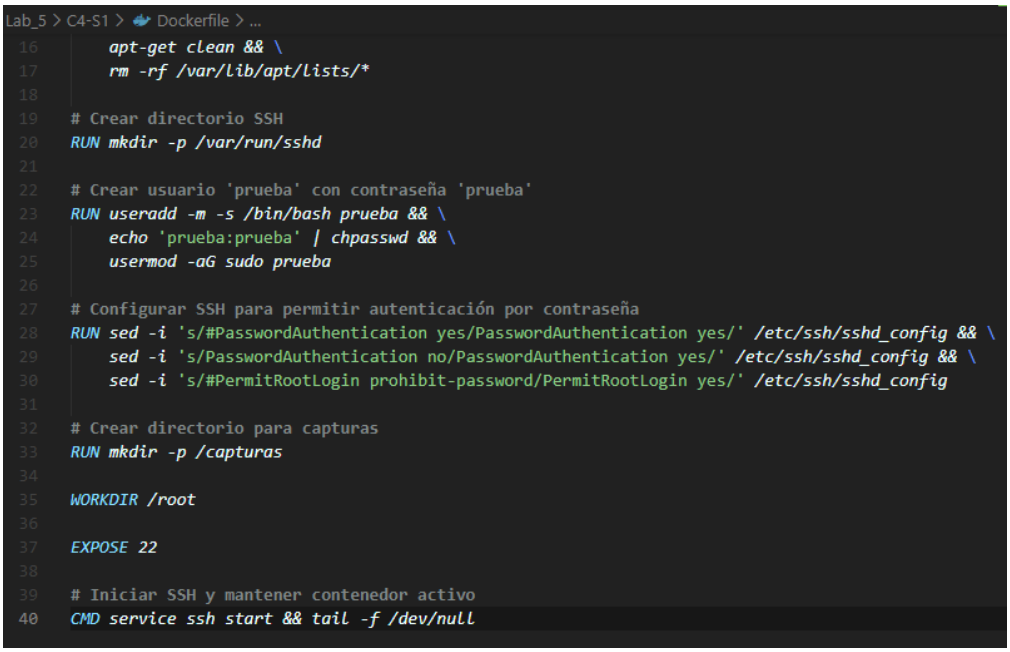


Figura 6: Dockerfile de C1 con el OS: Ubuntu 22.04.

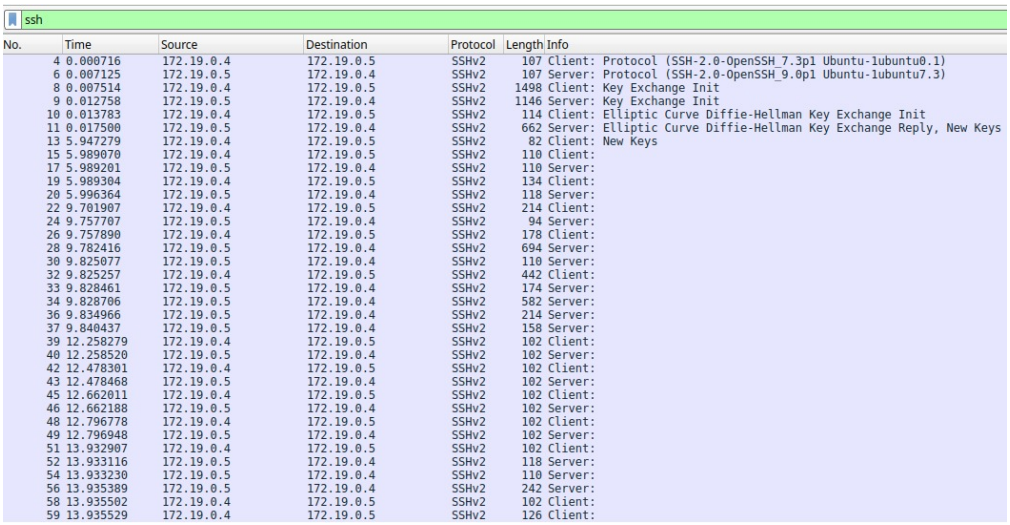
1.2. Creación de las credenciales para S1

En la figura 6, se puede ver en las siguientes líneas, la creación de las credenciales de prueba que se utilizaron para el protocolo ssh, utilizando el usuario **prueba** y la contraseña **prueba**

```
RUN useradd -m -s /bin/bash prueba && \
    echo 'prueba:prueba' | chpasswd && \
    usermod -aG sudo prueba
```

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

1.3.1. Tráfico generado por C1



ssh

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000716	172.19.0.4	172.19.0.5	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)
6	0.007125	172.19.0.5	172.19.0.4	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
8	0.007514	172.19.0.4	172.19.0.5	SSHv2	1498	Client: Key Exchange Init
9	0.012758	172.19.0.5	172.19.0.4	SSHv2	1146	Server: Key Exchange Init
10	0.013783	172.19.0.4	172.19.0.5	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11	0.017500	172.19.0.5	172.19.0.4	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
13	5.947279	172.19.0.4	172.19.0.5	SSHv2	82	Client: New Keys
15	5.989070	172.19.0.4	172.19.0.5	SSHv2	110	Client:
17	5.989201	172.19.0.5	172.19.0.4	SSHv2	110	Server:
19	5.989304	172.19.0.4	172.19.0.5	SSHv2	134	Client:
20	5.996364	172.19.0.5	172.19.0.4	SSHv2	118	Server:
22	9.701907	172.19.0.4	172.19.0.5	SSHv2	214	Client:
24	9.757707	172.19.0.5	172.19.0.4	SSHv2	94	Server:
26	9.757890	172.19.0.4	172.19.0.5	SSHv2	178	Client:
28	9.782416	172.19.0.5	172.19.0.4	SSHv2	694	Server:
30	9.825077	172.19.0.5	172.19.0.4	SSHv2	110	Server:
32	9.825257	172.19.0.4	172.19.0.5	SSHv2	442	Client:
33	9.828461	172.19.0.5	172.19.0.4	SSHv2	174	Server:
34	9.828706	172.19.0.5	172.19.0.4	SSHv2	582	Server:
36	9.834966	172.19.0.5	172.19.0.4	SSHv2	214	Server:
37	9.840437	172.19.0.5	172.19.0.4	SSHv2	158	Server:
39	12.258279	172.19.0.4	172.19.0.5	SSHv2	102	Client:
40	12.258520	172.19.0.5	172.19.0.4	SSHv2	102	Server:
42	12.478301	172.19.0.4	172.19.0.5	SSHv2	102	Client:
43	12.478468	172.19.0.5	172.19.0.4	SSHv2	102	Server:
45	12.662011	172.19.0.4	172.19.0.5	SSHv2	102	Client:
46	12.662188	172.19.0.5	172.19.0.4	SSHv2	102	Server:
48	12.796778	172.19.0.4	172.19.0.5	SSHv2	102	Client:
49	12.796948	172.19.0.5	172.19.0.4	SSHv2	102	Server:
51	13.932907	172.19.0.4	172.19.0.5	SSHv2	102	Client:
52	13.933116	172.19.0.5	172.19.0.4	SSHv2	118	Server:
54	13.933230	172.19.0.5	172.19.0.4	SSHv2	110	Server:
56	13.935389	172.19.0.5	172.19.0.4	SSHv2	242	Server:
58	13.935502	172.19.0.4	172.19.0.5	SSHv2	102	Client:
59	13.935529	172.19.0.4	172.19.0.5	SSHv2	126	Client:

Figura 7: Paquetes ssh en C1.

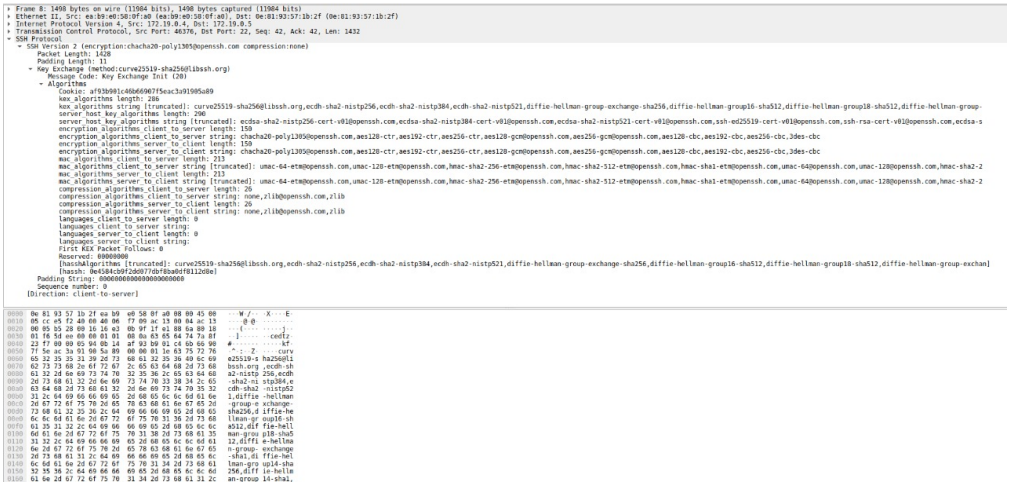


Figura 8: Detalles de la captura de C1 y el Hassh.

Versión SSH Cliente: SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1
Versión SSH Servidor: SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
Tamaños de paquetes del flujo (Cliente):

No.	Tamaño	Descripción
1	74 B	TCP SYN
3	66 B	TCP ACK
4	107 B	Protocol Version Exchange
8	1498 B	**Client: Key Exchange Init**
10	114 B	Diffie-Hellman Key Exchange Init
13	82 B	New Keys
15-59	102-442 B	Encrypted packets (autenticación y sesión)
61	66 B	TCP FIN

Tabla 1: Paquetes enviados por C1 (22 paquetes, 3631 bytes total)

HASSH Cliente: `92674389c9f72c2d6e9f5c19a4a8d40d`
Tamaño del KEI: 1428 bytes (payload)
Cookie de sesión: `af93b901c4bb66907f5eac3a91905a89`
Información en texto plano: Versión SSH/OpenSSH, lista de algoritmos criptográficos, cookie de sesión.
Información cifrada: Contraseña de autenticación, comandos ejecutados, respuestas del servidor.

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

1.4.1. Tráfico generado por C2

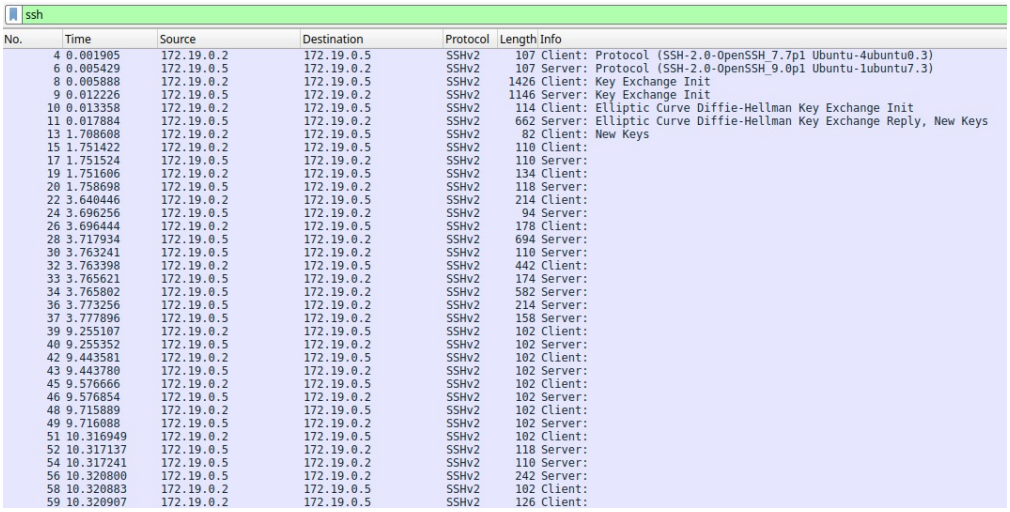


Figura 9: Paquetes ssh en C2.

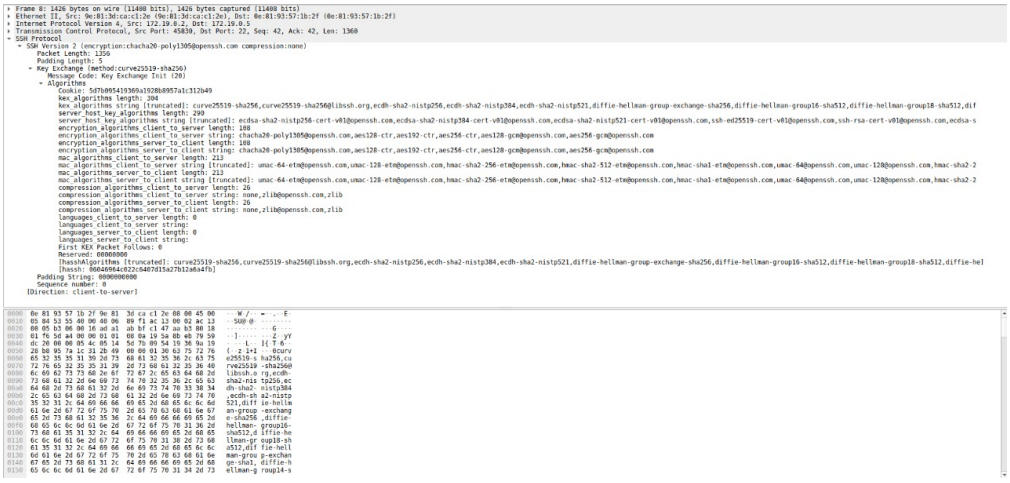


Figura 10: Detalles de la captura de C2 y el Hassh.

Versión SSH Cliente: SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3
Versión SSH Servidor: SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
Tamaños de paquetes del flujo (Cliente):
HASSH Cliente: `06046964c022c6407d15a27b12a6a4fb`
Algoritmos propuestos en Key Exchange Init:

No.	Tamaño	Descripción
1	74 B	TCP SYN
3	66 B	TCP ACK
4	107 B	Protocol Version Exchange
8	1426 B	**Client: Key Exchange Init**
10	114 B	Diffie-Hellman Key Exchange Init
13	82 B	New Keys
15-59	102-442 B	Encrypted packets (autenticación y sesión)
61	66 B	TCP FIN

Tabla 2: Paquetes enviados por C2 (22 paquetes total)

Tamaño del KEI: 1356 bytes (payload)
Cookie de sesión: `5d7b095419369a1928b8957a1c312b49`

Información en texto plano: Versión SSH/OpenSSH, lista de algoritmos criptográficos, cookie de sesión.

Información cifrada: Contraseña de autenticación, comandos ejecutados, respuestas del servidor.

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

1.5.1. Tráfico generado por C3

ssh

No.	Time	Source	Destination	Protocol	Length	Info
4	0.002799	172.19.0.3	172.19.0.5	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
6	0.004900	172.19.0.5	172.19.0.3	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
8	0.005052	172.19.0.3	172.19.0.5	SSHv2	1578	Client: Key Exchange Init
10	0.011117	172.19.0.5	172.19.0.3	SSHv2	1146	Server: Key Exchange Init
11	0.012066	172.19.0.3	172.19.0.5	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
12	0.016173	172.19.0.5	172.19.0.3	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
14	2.253683	172.19.0.3	172.19.0.5	SSHv2	82	Client: New Keys
16	2.294332	172.19.0.3	172.19.0.5	SSHv2	110	Client:
18	2.294438	172.19.0.5	172.19.0.3	SSHv2	110	Server:
20	2.294529	172.19.0.3	172.19.0.5	SSHv2	134	Client:
21	2.301601	172.19.0.5	172.19.0.3	SSHv2	118	Server:
23	4.584742	172.19.0.3	172.19.0.5	SSHv2	214	Client:
25	4.640586	172.19.0.5	172.19.0.3	SSHv2	94	Server:
27	4.640762	172.19.0.3	172.19.0.5	SSHv2	178	Client:
29	4.662498	172.19.0.5	172.19.0.3	SSHv2	694	Server:
31	4.710227	172.19.0.5	172.19.0.3	SSHv2	110	Server:
33	4.710367	172.19.0.3	172.19.0.5	SSHv2	442	Client:
34	4.712253	172.19.0.5	172.19.0.3	SSHv2	174	Server:
35	4.712466	172.19.0.5	172.19.0.3	SSHv2	582	Server:
37	4.718590	172.19.0.5	172.19.0.3	SSHv2	214	Server:
38	4.723018	172.19.0.5	172.19.0.3	SSHv2	158	Server:
40	11.356162	172.19.0.3	172.19.0.5	SSHv2	102	Client:
41	11.356382	172.19.0.5	172.19.0.3	SSHv2	102	Server:
43	11.511218	172.19.0.3	172.19.0.5	SSHv2	102	Client:
44	11.511413	172.19.0.5	172.19.0.3	SSHv2	102	Server:
46	11.634606	172.19.0.3	172.19.0.5	SSHv2	102	Client:
47	11.634788	172.19.0.5	172.19.0.3	SSHv2	102	Server:
49	11.784720	172.19.0.3	172.19.0.5	SSHv2	102	Client:
50	11.784970	172.19.0.5	172.19.0.3	SSHv2	102	Server:
52	12.145290	172.19.0.3	172.19.0.5	SSHv2	102	Client:
53	12.145541	172.19.0.5	172.19.0.3	SSHv2	118	Server:
55	12.145588	172.19.0.5	172.19.0.3	SSHv2	110	Server:
57	12.149049	172.19.0.5	172.19.0.3	SSHv2	242	Server:
59	12.149169	172.19.0.3	172.19.0.5	SSHv2	102	Client:
60	12.149193	172.19.0.3	172.19.0.5	SSHv2	126	Client:

Figura 11: Paquetes ssh en C3.

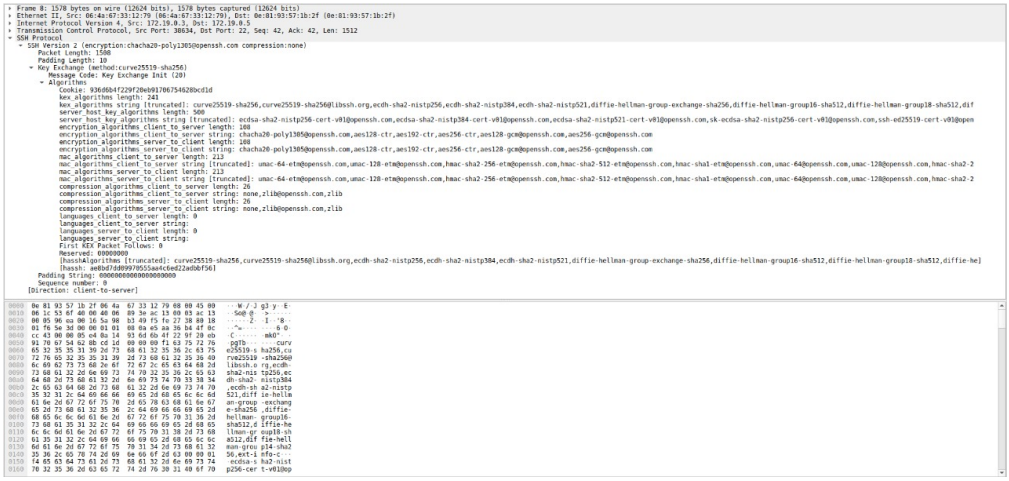


Figura 12: Detalles de la captura de C3 y el Hassh.

Versión SSH Cliente: SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1
Versión SSH Servidor: SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
Tamaños de paquetes del flujo (Cliente):

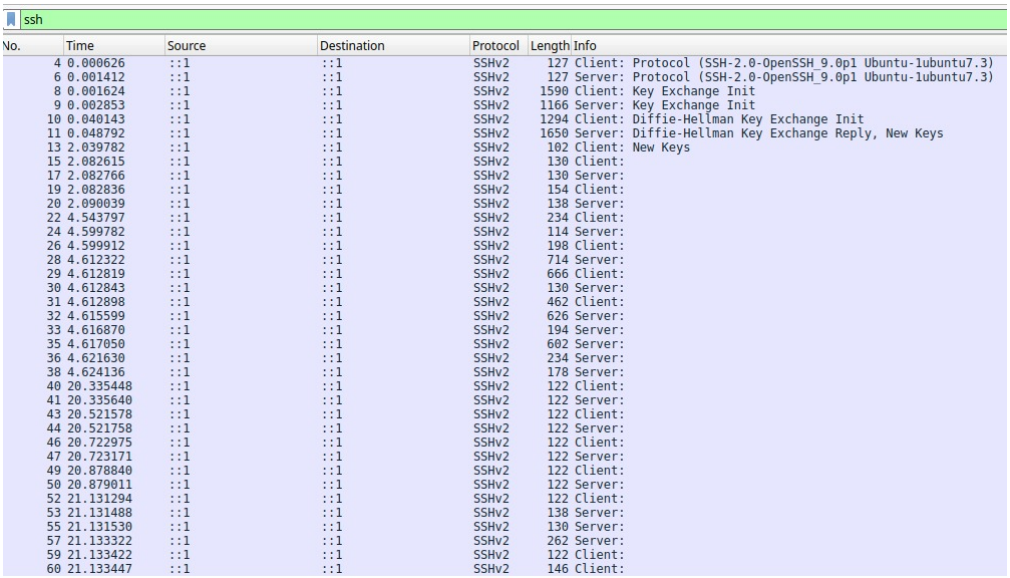
No.	Tamaño	Descripción
1	74 B	TCP SYN
3	66 B	TCP ACK
4	107 B	Protocol Version Exchange
8	1578 B	**Client: Key Exchange Init**
10	114 B	Diffie-Hellman Key Exchange Init
13	82 B	New Keys
15-59	102-442 B	Encrypted packets (autenticación y sesión)
61	66 B	TCP FIN

Tabla 3: Paquetes enviados por C3 (22 paquetes total)

HASSH Cliente: `ae80d7d609970555aa4c6ed22adbbf56`
Algoritmos propuestos en Key Exchange Init:
Tamaño del KEI: 1508 bytes (payload)
Cookie de sesión: `936d6b4f229f20eb9170675462bcbd1d`
Información en texto plano: Versión SSH/OpenSSH, lista de algoritmos criptográficos, cookie de sesión.
Información cifrada: Contraseña de autenticación, comandos ejecutados, respuestas del servidor.

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

1.6.1. Tráfico generado por C4 (localhost)



ssh

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000626	::1	::1	SSHv2	127	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
6	0.001412	::1	::1	SSHv2	127	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
8	0.001624	::1	::1	SSHv2	1590	Client: Key Exchange Init
9	0.002853	::1	::1	SSHv2	1166	Server: Key Exchange Init
10	0.040143	::1	::1	SSHv2	1294	Client: Diffie-Hellman Key Exchange Init
11	0.048792	::1	::1	SSHv2	1650	Server: Diffie-Hellman Key Exchange Reply, New Keys
13	2.039782	::1	::1	SSHv2	102	Client: New Keys
15	2.082615	::1	::1	SSHv2	130	Client:
17	2.082766	::1	::1	SSHv2	130	Server:
19	2.082836	::1	::1	SSHv2	154	Client:
20	2.090039	::1	::1	SSHv2	138	Server:
22	4.543797	::1	::1	SSHv2	234	Client:
24	4.599782	::1	::1	SSHv2	114	Server:
26	4.599912	::1	::1	SSHv2	198	Client:
28	4.612322	::1	::1	SSHv2	714	Server:
29	4.612819	::1	::1	SSHv2	666	Client:
30	4.612843	::1	::1	SSHv2	130	Server:
31	4.612898	::1	::1	SSHv2	462	Client:
32	4.615599	::1	::1	SSHv2	626	Server:
33	4.616870	::1	::1	SSHv2	194	Server:
35	4.617050	::1	::1	SSHv2	602	Server:
36	4.621630	::1	::1	SSHv2	234	Server:
38	4.624136	::1	::1	SSHv2	178	Server:
40	20.335448	::1	::1	SSHv2	122	Client:
41	20.335640	::1	::1	SSHv2	122	Server:
43	20.521578	::1	::1	SSHv2	122	Client:
44	20.521758	::1	::1	SSHv2	122	Server:
46	20.722975	::1	::1	SSHv2	122	Client:
47	20.723171	::1	::1	SSHv2	122	Server:
49	20.878840	::1	::1	SSHv2	122	Client:
50	20.879011	::1	::1	SSHv2	122	Server:
52	21.131294	::1	::1	SSHv2	122	Client:
53	21.131488	::1	::1	SSHv2	138	Server:
55	21.131530	::1	::1	SSHv2	130	Server:
57	21.133322	::1	::1	SSHv2	262	Server:
59	21.133422	::1	::1	SSHv2	122	Client:
60	21.133447	::1	::1	SSHv2	146	Client:

Figura 13: Paquetes ssh en C4.

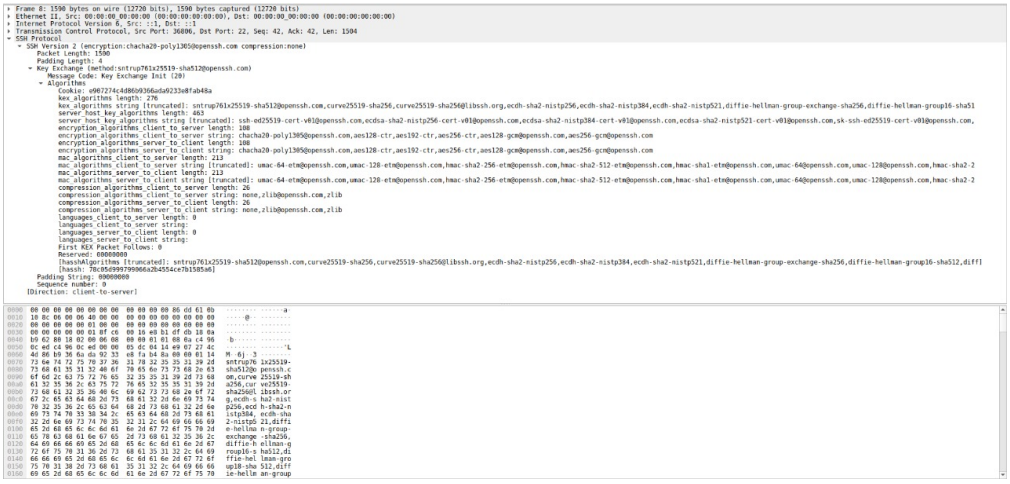


Figura 14: Detalles de la captura de C4 y el Hassh.

Versión SSH Cliente: SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
Versión SSH Servidor: SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
Interfaz: Loopback (::1 IPv6)
Nota: Cliente y servidor son la misma máquina (conexión localhost)

Tamaños de paquetes del flujo (Cliente):

No.	Tamaño	Descripción
4	127 B	Protocol Version Exchange
8	1590 B	**Client: Key Exchange Init**
10	1294 B	Diffie-Hellman Key Exchange Init
13	102 B	New Keys
15	130 B	Client: Encrypted packet
19-60	122-666 B	Encrypted packets (autenticación y sesión)

Tabla 4: Paquetes enviados por C4 hacia localhost

HASSH Cliente: `78c05d99970066aa2b4554ce7b1585a6`
Algoritmos propuestos en Key Exchange Init:
Tamaño del KEI: 1500 bytes (payload)
Cookie de sesión: `e907274c4d86b9366ada9233e8fab48a`
Información en texto plano: Versión SSH/OpenSSH, lista de algoritmos criptográficos, cookie de sesión.
Información cifrada: Contraseña de autenticación, comandos ejecutados, respuestas del servidor.

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

Sección omitida por indicaciones del profesor.

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

1.8.1. C1

Durante la captura de tráfico en C1, se identifican los siguientes paquetes generados por el cliente antes de que comience el cifrado, exponiendo la siguiente información:

- **Establecimiento TCP (Frames 1 y 3):** Se observan los paquetes SYN y ACK con los números de secuencia iniciales, estableciendo la conexión con el servidor en el puerto 22.
- **Versión del Protocolo (Frame 4):** El cliente envía su banner de identificación en texto claro: `SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1`.
- **Key Exchange Init (Frame 8):** El cliente envía sus listas de algoritmos soportados y una cookie aleatoria (`0e5b8dc8...`) para evitar ataques de replay. Entre las propuestas visibles destacan:

- KEX: `curve25519-sha256`, `ecdh-sha2-nistp256`
- Cifrado: `chacha20-poly1305@openssh.com`, `aes128-ctr`
- MAC: `umac-64-etm@openssh.com`

- **ECDH Key Exchange Init (Frame 10):** Se observa el envío de la clave pública efímera del cliente necesaria para realizar el intercambio Diffie-Hellman con el algoritmo `curve25519`.
- **New Keys (Frame 13):** Último mensaje legible que señaliza que todo el tráfico posterior estará cifrado.

1.8.2. C2

Durante la captura de tráfico en C2, se identifican los siguientes paquetes generados por el cliente antes de que comience el cifrado, exponiendo la siguiente información:

- **Establecimiento TCP (Frames 1 y 3):** Se observan los paquetes SYN y ACK con los números de secuencia iniciales, estableciendo la conexión con el servidor en el puerto 22.
- **Versión del Protocolo (Frame 4):** El cliente envía su banner de identificación en texto claro: `SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3`.
- **Key Exchange Init (Frame 8):** El cliente envía sus listas de algoritmos soportados y una cookie aleatoria (`8b0a6b5d...`) para evitar ataques de replay. Entre las propuestas visibles destacan:
  - KEX: `curve25519-sha256`, `curve25519-sha256@libssh.org`, `ecdh-sha2-nistp256`
  - Cifrado: `chacha20-poly1305@openssh.com`, `aes128-ctr`, `aes192-ctr`, `aes256-ctr`
  - MAC: `umac-64-etm@openssh.com`, `umac-128-etm@openssh.com`, `hmac-sha2-256-etm@opens`
- **ECDH Key Exchange Init (Frame 10):** Se observa el envío de la clave pública efímera del cliente necesaria para realizar el intercambio Diffie-Hellman con el algoritmo `curve25519`.
- **New Keys (Frame 13):** Último mensaje legible que señaliza que todo el tráfico posterior estará cifrado.

1.8.3. C3

Durante la captura de tráfico en C3, se identifican los siguientes paquetes generados por el cliente antes de que comience el cifrado, exponiendo la siguiente información:

- **Establecimiento TCP (Frames 1 y 3):** Se observan los paquetes SYN y ACK con los números de secuencia iniciales, estableciendo la conexión con el servidor en el puerto 22.

- **Versión del Protocolo (Frame 4):** El cliente envía su banner de identificación en texto claro: `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`.
- **Key Exchange Init (Frame 8):** El cliente envía sus listas de algoritmos soportados y una cookie aleatoria (`ae80d7d6...`) para evitar ataques de replay. Entre las propuestas visibles destacan:
  - KEX: `curve25519-sha256`, `curve25519-sha256@libssh.org`, `ecdh-sha2-nistp256`, `ecdh-sha2-nistp384`, `ecdh-sha2-nistp521`
  - Cifrado: `chacha20-poly1305@openssh.com`, `aes128-ctr`, `aes192-ctr`, `aes256-ctr`, `aes128-gcm@openssh.com`
  - MAC: `umac-64-etm@openssh.com`, `umac-128-etm@openssh.com`, `hmac-sha2-256-etm@opens` `hmac-sha2-512-etm@openssh.com`
- **ECDH Key Exchange Init (Frame 11):** Se observa el envío de la clave pública efímera del cliente necesaria para realizar el intercambio Diffie-Hellman con el algoritmo `curve25519`.
- **New Keys (Frame 14):** Último mensaje legible que señaliza que todo el tráfico posterior estará cifrado.

1.8.4. C4/S1

Durante la captura de tráfico en C4 conectándose a S1 en localhost (interfaz loopback), se identifican los siguientes paquetes generados por el cliente antes de que comience el cifrado, exponiendo la siguiente información:

- **Establecimiento TCP (Frames 1 y 3):** Se observan los paquetes SYN y ACK con los números de secuencia iniciales, estableciendo la conexión con el servidor en el puerto 22 a través de la interfaz loopback IPv6 (::1).
- **Versión del Protocolo (Frame 4):** El cliente envía su banner de identificación en texto claro: `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`.
- **Key Exchange Init (Frame 8):** El cliente envía sus listas de algoritmos soportados y una cookie aleatoria (`78c4d5g9...`) para evitar ataques de replay. Entre las propuestas visibles destacan:
  - KEX: `sntrup761x25519-sha512@openssh.com`, `curve25519-sha256`, `curve25519-sha256@l` `ecdh-sha2-nistp256`, `ecdh-sha2-nistp384`, `ecdh-sha2-nistp521`
  - Cifrado: `chacha20-poly1305@openssh.com`, `aes128-ctr`, `aes192-ctr`, `aes256-ctr`, `aes128-gcm@openssh.com`, `aes256-gcm@openssh.com`
  - MAC: `umac-64-etm@openssh.com`, `umac-128-etm@openssh.com`, `hmac-sha2-256-etm@opens` `hmac-sha2-512-etm@openssh.com`

- **Diffie-Hellman Key Exchange Init (Frame 10):** Se observa el envío de la clave pública efímera del cliente necesaria para realizar el intercambio con el algoritmo post-cuántico `sntrup761x25519-sha512@openssh.com`, que combina protección contra ataques de computadoras cuánticas.
- **New Keys (Frame 13):** Último mensaje legible que señaliza que todo el tráfico posterior estará cifrado.

Nota: A diferencia de los escenarios anteriores, C4 introduce el algoritmo de intercambio de claves post-cuántico `sntrup761x25519-sha512@openssh.com` como primera opción, reflejando las mejoras de seguridad implementadas en OpenSSH 9.0 para proteger contra futuras amenazas de computación cuántica.

1.9. Diferencia entre C1 y C2

Entre C1 (OpenSSH 7.3p1) y C2 (OpenSSH 7.7p1) se observan las siguientes diferencias en el tráfico SSH:

Versión del cliente:

- C1: SSH-2.0-OpenSSH 7.3p1 Ubuntu-1ubuntu0.1
- C2: SSH-2.0-OpenSSH 7.7p1 Ubuntu-4ubuntu0.3

Tamaño del Key Exchange Init:

- C1: 1498 bytes (1428 bytes payload)
- C2: 1426 bytes (1356 bytes payload)
- **Reducción de 72 bytes en el paquete KEI**

HASSH:

- C1: `92674389c9f72c2d6e9f5c19a4a8d40d`
- C2: `06046964c022c6407d15a27b12a6a4fb`

Cambios en algoritmos:
C2 **elimina** algunos algoritmos de intercambio de claves presentes en C1:

- Se removió: `diffie-hellman-group-exchange-sha1`
- Se removió: `diffie-hellman-group14-sha1`

Estos algoritmos fueron considerados menos seguros en versiones posteriores de OpenSSH, por lo que la versión 7.7p1 los eliminó de su lista de algoritmos propuestos por defecto. La cookie de sesión también cambió de `af93b901c4bb66907f5eac3a91905a89` (C1) a `5d7b095419369a1928b8957a1c312b49` (C2), lo cual es esperado ya que se genera aleatoriamente en cada conexión.

1.10. Diferencia entre C2 y C3

Entre C2 (OpenSSH 7.7p1) y C3 (OpenSSH 8.3p1) se observan las siguientes diferencias en el tráfico SSH:

Versión del cliente:

- C2: SSH-2.0-OpenSSH 7.7p1 Ubuntu-4ubuntu0.3
- C3: SSH-2.0-OpenSSH 8.3p1 Ubuntu-1ubuntu0.1

Tamaño del Key Exchange Init:

- C2: 1426 bytes (1356 bytes payload)
- C3: 1578 bytes (1508 bytes payload)
- **Incremento de 152 bytes en el paquete KEI**

HASSH:

- C2: `06046964c022c6407d15a27b12a6a4fb`
- C3: `ae80d7d609970555aa4c6ed22adbbf56`

Cambios en algoritmos:
C3 **añade** nuevos algoritmos MAC no presentes en C2:

- Se agregó: `umac-64@openssh.com`
- Se agregó: `umac-128@openssh.com`
- Se agregó: `hmac-sha2-256`
- Se agregó: `hmac-sha2-512`

Estos algoritmos MAC adicionales (sin el sufijo `-etm`) proporcionan compatibilidad con versiones anteriores que no soportan el modo Encrypt-then-MAC (ETM). OpenSSH 8.3p1 amplía la lista de algoritmos MAC para mejorar la interoperabilidad. La cookie de sesión cambió a `936d6b4f229f20eb9170675462bcbd1d`.

1.11. Diferencia entre C3 y C4

Entre C3 (OpenSSH 8.3p1) y C4 (OpenSSH 9.0p1) se observan las siguientes diferencias en el tráfico SSH:

Versión del cliente:

- C3: SSH-2.0-OpenSSH 8.3p1 Ubuntu-1ubuntu0.1
- C4: SSH-2.0-OpenSSH 9.0p1 Ubuntu-1ubuntu7.3

Tamaño del Key Exchange Init:

- C3: 1578 bytes (1508 bytes payload)
- C4: 1590 bytes (1500 bytes payload en captura cliente)
- **Incremento de 12 bytes en el paquete KEI**

HASSH:

- C3: `ae80d7d609970555aa4c6ed22adbbf56`
- C4: `78c05d99970066aa2b4554ce7b1585a6`

Cambios en algoritmos:
C4 **añade** un nuevo algoritmo de intercambio de claves al inicio de la lista:

- Se agregó en primera posición: `sntrup761x25519-sha512@openssh.com`

Este es un algoritmo de intercambio de claves **post-cuántico** que combina la encapsulación de claves Streamlined NTRU Prime con curve25519, diseñado para resistir ataques de computadoras cuánticas futuras. OpenSSH 9.0 introdujo este algoritmo como el predeterminado y más preferido en el proceso de negociación. La cookie de sesión para esta conexión localhost fue `e907274c4d86b9366ada9233e8fab48a`.

Nota adicional: C4 se conectó a sí mismo (localhost vía IPv6 ::1), lo que explica por qué tanto cliente como servidor reportan la misma versión OpenSSH 9.0p1.

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión "?"

El cliente utilizado es un cliente personalizado, lo cual se descubre al analizar el paquete de intercambio de protocolo SSH. Los clientes OpenSSH estándar envían por defecto su versión exacta durante la conexión inicial, sin embargo, en este caso se observa la versión `SSH-2.0-OpenSSH_?`, donde el carácter ¿reemplaza el número de versión real. Esta anomalía indica que el informante modificó intencionalmente el string de versión para ocultar la implementación específica de su cliente SSH.

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

```python
import paramiko
import socket
import time
import sys

# Configuración
HOST_IP = "172.19.0.5"
PUERTO = 22
VERSION_A_FALSIFICAR = "SSH-2.0-OpenSSH_?"
USUARIO = "prueba"
PASSWORD = "prueba"

def conectar_ssh_falsificado():
    sock = None
    transport = None

    try:
        print("="*60)
        print("  REPLICACIÓN SSH CON VERSIÓN FALSIFICADA")
        print("="*60)
        print(f"[*] Servidor: {HOST_IP}:{PUERTO}")
        print(f"[*] Usuario: {USUARIO}")
        print(f"[*] Versión a falsificar: {VERSION_A_FALSIFICAR}")
        print()

        # Paso 1: Crear socket
        print("[1/5] Creando socket TCP...")
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(5)  # ← IMPORTANTE: timeout de 5 segundos
        print("      ✓ Socket creado")

        # Paso 2: Conectar
        print(f"[2/5] Conectando a {HOST_IP}:{PUERTO}...")
        sock.connect((HOST_IP, PUERTO))
        print("      ✓ Conexión TCP establecida")

        # Paso 3: Crear transporte SSH y falsificar versión
        print("[3/5] Creando transporte SSH...")
        transport = paramiko.Transport(sock)
        transport.local_version = VERSION_A_FALSIFICAR
```

Figura 15: Código parcial del script de ssh en Python.

Para replicar el tráfico se utilizó *Paramiko* ejecutado desde dentro del contenedor Docker:

1. **Creación del script:** Se implementó `main.py` con `transport.local_version = "SSH-2.0-OpenS` para falsificar la versión.

2. **Transferencia al contenedor:**
   `docker cp ssh_main.py C4-S1:/tmp/`

3. **Instalación de Paramiko:**
   `docker exec -it C4-S1 bash -c .ªpt-get update && apt-get install -y python3-pip && pip3 install paramiko"`

4. **Inicio de captura (Terminal 1):**
   `docker exec -it C4-S1 tcpdump -i lo -w /capturas/test3.pcap port 22`

5. **Ejecución del script (Terminal 2):**
   `docker exec -it C4-S1 python3 /tmp/ssh_main.py`

6. **Detención de captura:** Ctrl+C en Terminal 1

7. **Extracción:**
   `docker cp C4-S1:/capturas/test3.pcap ./test3.pcap`

8. **Verificación:**
   `tshark -r test3.pcap -Y "sshT fields -e ssh.protocol`

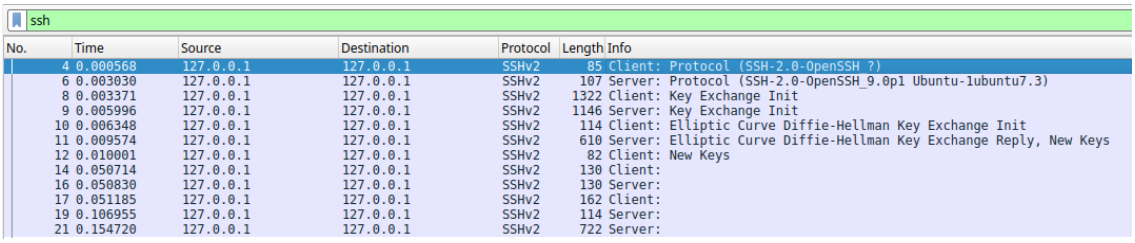


Figura 16: Captura .pcap con la recreación del paquete ssh simulado.

Resultado: La captura muestra exitosamente `SSH-2.0-OpenSSH_?` en el paquete Protocol Version Exchange del cliente, replicando el tráfico del informante. Se utilizó la interfaz loopback (`lo`) porque la conexión se realizó desde el contenedor hacia localhost.

3. Desarrollo (Parte 3)

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

Para lograr que el paquete Key Exchange Init sea de 300 bytes o menos, se deben limitar los algoritmos criptograficos de la configuración de OpenSSH. por lo que se instala un editor de texto, en este caso *nano* para editar la configuración de OpenSSH. Luego, aplicamos la nueva configuración y realizamos la captura desde alguna máquina, en este caso se usó **C2**.

```
# Acceder al contenedor
docker exec -it C4-S1 bash

# Instalar nano (desde el contendeor C4-S1)
apt-get update
apt-get install -y nano

# Editar configuracion SSH (desde el contenedor C4-S1)
nano /etc/ssh/sshd_config
```

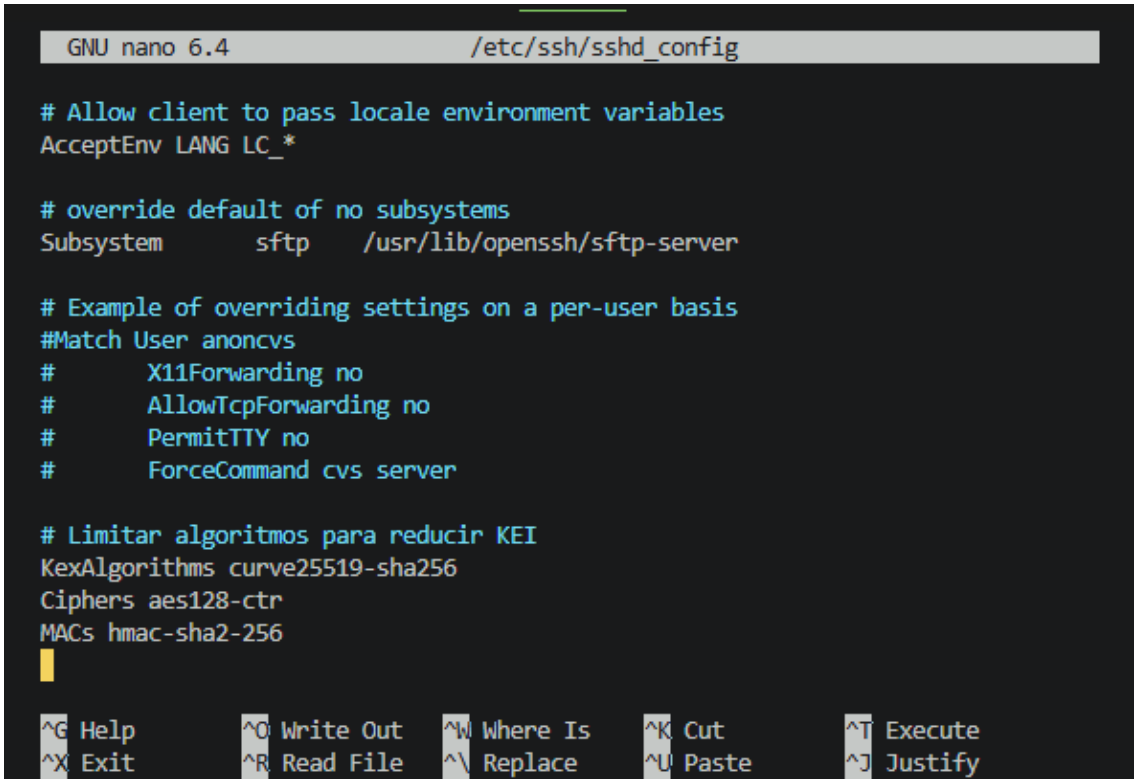


Figura 17: Modificando config OpenSSH.

```
# Reiniciar SSH dentro del contenedor (C4-S1)
service ssh restart

# Salir del contenedor
exit
```

Ahora, con la configuración modificada, se procede a realizar la captura como se hizo anteriormente. De C2 a C4-S1. mientras, en otra terminal hacemos el login ssh al usuario de prueba en C4-S1, para salir inmediatamente y ver el registro de la captura en el visor de wireshark.

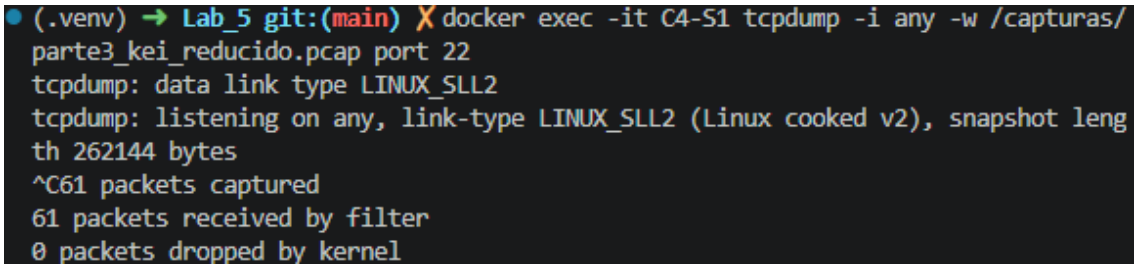


Figura 18: Captura en Servidor C4-S1 de C2.

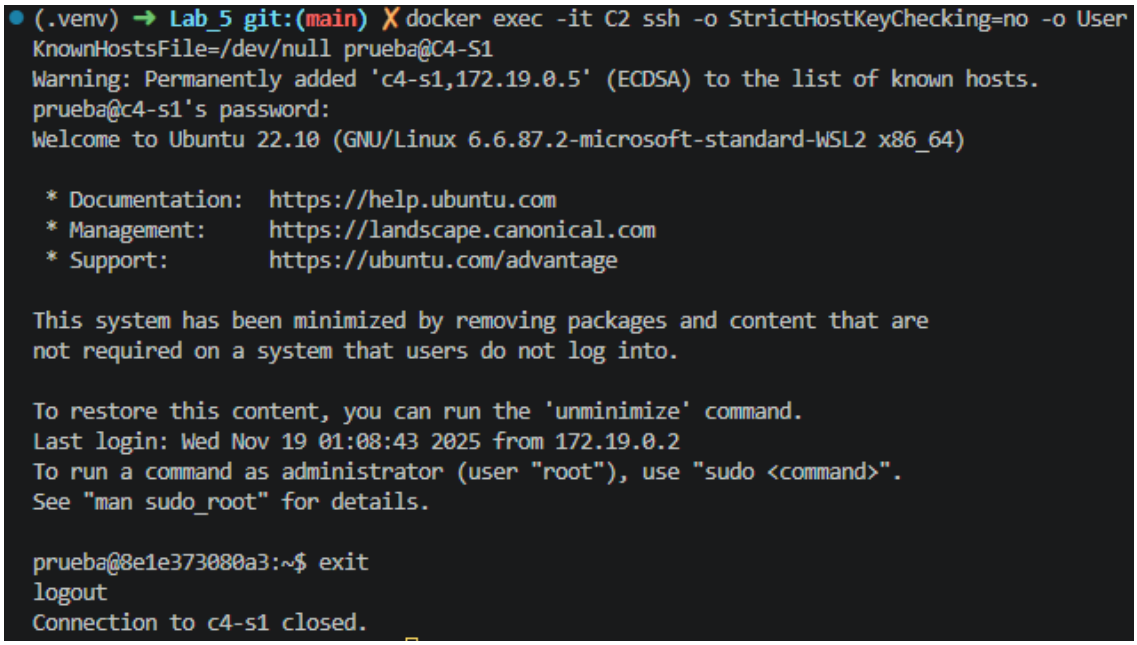


Figura 19: Entrando a C4-S1 desde C2 (modificado).

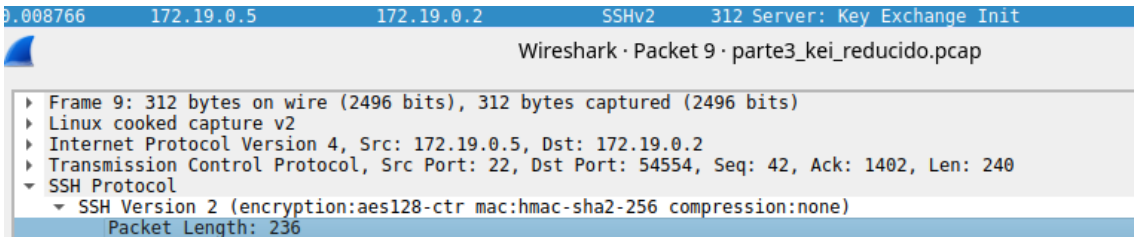


Figura 20: Captura de wireshark del paquete reducido.

Se puede notar que se redujo drásticamente el paquete init del servidor a comparación de como era anterior mente (1076 vs 236 bytes).

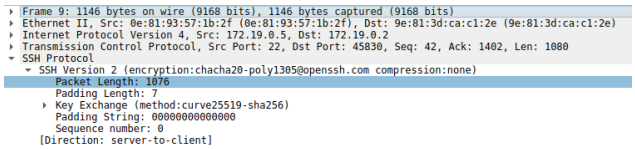


Figura 21: Paquete ssh init antes de reducir.

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es una implementación de código abierto del protocolo SSH que proporciona comunicación cifrada y autenticada sobre redes no seguras. Opera sobre TCP en el puerto 22 y establece un canal seguro mediante un handshake estructurado.

A partir del análisis del tráfico interceptado en las capturas C1, C2, C3 y C4, se observa que el establecimiento de conexión SSH sigue la siguiente secuencia:

Fase 1: Establecimiento TCP. Se realiza el three-way handshake estándar (SYN, SYN-ACK, ACK). En las capturas corresponde a los primeros 3 paquetes, con tamaños entre 72-94 bytes según IPv4 o IPv6.

Fase 2: Protocol Version Exchange. Cliente y servidor intercambian versiones en texto plano. En las capturas se observaron las siguientes versiones:

- C1: `SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1`
- C2: `SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3`
- C3: `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`
- C4: `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`
- Servidor: `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`

Fase 3: Key Exchange Init (KEXINIT). Ambas partes envían listas de algoritmos soportados para key exchange, cifrado, MAC y compresión. Los tamaños de estos paquetes varían según la versión:

- C1: 1498 bytes (cliente), 1152 bytes (servidor)
- C2: 1426 bytes (cliente), 1152 bytes (servidor)
- C3: 1578 bytes (cliente), 1152 bytes (servidor)
- C4: 1590 bytes (cliente), 1166 bytes (servidor)

- Parte 3 (modificado): 236 bytes (servidor reducido)

Fase 4: Key Exchange. Se intercambian claves usando ECDH (120 bytes en C1, C2) o Diffie-Hellman (1294 bytes en C3, C4). En C4 se observa el algoritmo post-cuántico `sntrup761x25519-sha512@openssh.com` como primera opción.

Fase 5: New Keys. Mensaje indicando que el tráfico posterior usará cifrado (88-102 bytes).

Fase 6: Sesión Cifrada. Toda comunicación posterior está cifrada, visible como "Encrypted packet" en las capturas con tamaños variables (102-666 bytes).

4.2. Capas de Seguridad en OpenSSH

El protocolo OpenSSH implementa múltiples capas de seguridad para garantizar los principios fundamentales de seguridad de la información. A continuación se analiza cada principio basándose en el tráfico interceptado:

Confidencialidad

Se garantiza mediante cifrado simétrico post-handshake. En las capturas se observa que todo el tráfico después de "New Keys" está cifrado. El principio se cumple para el contenido de sesión, aunque metadatos como versiones de software y timing permanecen expuestos.

Algoritmos de cifrado observados en las capturas:

- C1: `chacha20-poly1305@openssh.com`, `aes128-ctr`
- C2: `chacha20-poly1305@openssh.com`, `aes128-ctr`, `aes192-ctr`, `aes256-ctr`
- C3: `chacha20-poly1305@openssh.com`, `aes128-gcm@openssh.com`, `aes256-ctr`
- C4: `chacha20-poly1305@openssh.com`, `aes128-gcm@openssh.com`, `aes256-gcm@openssh.com`

En Wireshark, después del paquete "New Keys", todos los paquetes aparecen como "Encrypted packet" sin posibilidad de inspeccionar su contenido, confirmando el cifrado efectivo de credenciales, comandos y respuestas del servidor.

Limitaciones observadas:

- Versiones de software expuestas en texto plano (Fase 2)
- Metadatos no protegidos: IPs, puertos, timestamps, tamaños de paquetes
- Posible análisis de tráfico mediante patrones de comunicación

Integridad

Protegida mediante MAC (Message Authentication Code) sobre cada paquete cifrado. Los cifrados AEAD como ChaCha20-Poly1305 y AES-GCM combinan cifrado e integridad en una sola operación. El principio se cumple completamente ya que cualquier modificación será detectada.

Algoritmos MAC observados en las capturas:

- C1: `umac-64-etm@openssh.com`, `umac-128-etm@openssh.com`
- C2: `umac-64-etm@openssh.com`, `umac-128-etm@openssh.com`, `hmac-sha2-256-etm@openssh.com`
- C3: `umac-64-etm@openssh.com`, `hmac-sha2-256-etm@openssh.com`, `hmac-sha2-512-etm@openss`
- C4: `umac-64-etm@openssh.com`, `hmac-sha2-256-etm@openssh.com`, `hmac-sha2-512-etm@openss`

El sufijo "`-etm`" (Encrypt-then-MAC) observado en todas las versiones indica que el MAC se calcula sobre el texto cifrado, proporcionando protección adicional contra ataques de timing y manipulación. Cualquier modificación de un paquete resulta en fallo de verificación y terminación inmediata de la conexión.

Autenticidad

Garantizada bilateralmente mediante criptografía asimétrica. El principio se cumple para ambas partes.

Autenticación del servidor: Se observa en el paquete "Diffie-Hellman Key Exchange Reply" o "ECDH Reply":

- C1-C2: 668 bytes (ECDH Reply)
- C3-C4: 1650 bytes (Diffie-Hellman Reply)

Este paquete contiene la clave pública de host del servidor y una firma digital del intercambio de claves. En las capturas se observaron los siguientes tipos de claves de host:

- C1-C3: `ssh-rsa`, `ecdsa-sha2-nistp256`, `ssh-ed25519`
- C4: `rsa-sha2-512`, `rsa-sha2-256`, `ssh-ed25519`
- Parte 3 (restringido): únicamente `ssh-ed25519`

Autenticación del cliente: Se observa en paquetes cifrados posteriores a "New Keys" (tamaños de 102-442 bytes en las capturas). En el laboratorio se utilizó autenticación por contraseña (usuario "prueba", contraseña "prueba"), transmitida de forma cifrada dentro de la sesión SSH.

La combinación de intercambio Diffie-Hellman/ECDH con firma digital del servidor proporciona protección contra ataques man-in-the-middle (MITM).

Disponibilidad

SSH no implementa mecanismos específicos contra DoS, redundancia o balanceo de carga. Aunque incluye keep-alive y timeouts, la disponibilidad debe garantizarse mediante infraestructura adicional a nivel de red y sistema operativo.

Vulnerabilidades potenciales identificables del tráfico:

- **Connection flooding:** Múltiples conexiones TCP al puerto 22 pueden agotar recursos
- **Cryptographic DoS:** Operaciones criptográficas costosas (como las de 1294 bytes observadas en C3-C4) pueden sobrecargar la CPU del servidor
- **Slowloris SSH:** Mantener conexiones en fase de handshake sin completarlas

Mitigaciones externas necesarias:

- Firewalls y rate limiting (`iptables`, `fail2ban`)
- Configuración de `MaxStartups` en `sshd_config`
- Balanceadores de carga y redundancia de servidores
- Sistemas de detección de intrusiones (IDS/IPS)

El principio de disponibilidad no es objetivo primario de SSH, que se enfoca en la seguridad de la comunicación, no en la continuidad del servicio.

No Repudio

SSH no implementa timestamping criptográfico ni infraestructura PKI robusta. Las firmas digitales usadas para autenticación no proporcionan no repudio completo por falta de terceros certificadores, logs inmutables y mecanismos de revocación estándar.

Limitaciones identificadas:

1. **Ausencia de timestamping criptográfico:** Los timestamps visibles en las capturas son añadidos por tcpdump, no por SSH mismo.
2. **Sin infraestructura PKI formal:** Las claves de host SSH no están certificadas por autoridades de certificación (CA), no tienen fechas de expiración y se verifican mediante "Trust On First Use" (TOFU), no mediante cadena de confianza.
3. **Logs no firmados:** Aunque SSH genera logs de conexiones, estos no están firmados criptográficamente y pueden ser modificados por administradores con acceso root.
4. **Claves compartibles:** Las claves privadas SSH pueden ser copiadas a múltiples dispositivos, haciendo imposible demostrar que una acción específica fue realizada por una persona específica.

5. **Sesiones no auditables por terceros:** Del tráfico capturado se observa que después de "New Keys", todo está cifrado (paquetes de 102-666 bytes). No hay forma de que un tercero independiente pueda verificar qué comandos se ejecutaron o confirmar la identidad del operador.

El principio no se cumple. Las firmas digitales en SSH son solo para autenticación en tiempo real, no para prueba legal posterior. Para lograr no repudio se requeriría infraestructura adicional externa (servicios de sellado de tiempo RFC 3161, SIEM con logs protegidos, certificados SSH con CA).

4.3. Identificación de protocolos que no se cumplen

Basándose en el análisis del tráfico SSH interceptado y la arquitectura del protocolo, se identifican los siguientes resultados:

Principios que SÍ se cumplen completamente:

Confidencialidad: Cifrado robusto de toda la sesión

Integridad: MAC/AEAD en todos los paquetes

Autenticidad: Firmas digitales y autenticación bidireccional

Principios que NO se cumplen o se cumplen parcialmente:

× **Disponibilidad:** No es objetivo del protocolo, requiere infraestructura adicional

× **No Repudio:** Falta de PKI, timestamping criptográfico y logging inmutable

4.3.1. Vulnerabilidades identificables en el tráfico

Del análisis de las capturas C1, C2, C3 y C4 se identifican las siguientes vulnerabilidades:

1. Exposición de versiones en texto plano

Todas las capturas revelan las versiones exactas de cliente y servidor durante la Fase 2 (Protocol Version Exchange). Esto permite:

- Identificación de vulnerabilidades conocidas (CVEs específicos)
- Ataques dirigidos a versiones particulares
- Fingerprinting de sistemas

En la Parte 2 se demostró cómo ocultar esta información modificando el banner a `SSH-2.0-OpenSSH_?`, dificultando el reconocimiento.

2. Fingerprinting mediante HASSH

Los valores HASSH únicos observados permiten identificar y rastrear clientes específicos:

- C1: `92674389c9f72c2d6e9f5c19a4a8d40d`

- C2: `06046964c022c6407d15a27b12a6a4fb`
- C3: `ae80d7d609970555aa4c6ed22adbbf56`
- C4: `78c05d99970066aa2b4554ce7b1585a6`

Esta huella digital única posibilita el tracking de usuarios y dispositivos específicos a través de diferentes conexiones y redes.

3. Algoritmos legacy en versiones antiguas

En C1 (OpenSSH 7.3p1) se observaron algoritmos considerados débiles o vulnerables:

- `diffie-hellman-group14-sha1` (vulnerable a ataques de colisión SHA-1)
- `hmac-sha1` (algoritmo de hash obsoleto)
- `3des-cbc` (cifrado débil)

Estos algoritmos fueron progresivamente eliminados en versiones posteriores (C2, C3, C4), demostrando la evolución del protocolo hacia mayor seguridad.

4. Análisis de tráfico (Traffic Analysis)

Aunque el contenido está cifrado, los metadatos visibles en todas las capturas incluyen:

- Timestamps de cada paquete
- Tamaños de paquetes (102-666 bytes observados en fase cifrada)
- Frecuencia y patrones de comunicación
- Duración de sesiones

El análisis de estos patrones puede revelar:

- Tipo de actividades realizadas (transferencia de archivos vs. comandos interactivos)
- Horarios y patrones de comportamiento de usuarios
- Identificación de sesiones automatizadas vs. interactivas

5. Tamaños de KEI reveladores

La variación en tamaños de Key Exchange Init revela información sobre configuraciones:

- C1: 1498 bytes → aproximadamente 29 algoritmos
- C2: 1426 bytes → aproximadamente 27 algoritmos (eliminación de algoritmos inseguros)
- C3: 1578 bytes → aproximadamente 33 algoritmos (mayor compatibilidad)
- C4: 1590 bytes → aproximadamente 35 algoritmos (incluyendo post-cuánticos)
- Parte 3 reducido: 236 bytes → solo 4 algoritmos (configuración restringida)

Un KEI reducido disminuye la superficie de ataque pero puede afectar la compatibilidad con clientes antiguos.

4.3.2. Conclusión

El análisis del tráfico capturado en las cuatro versiones de OpenSSH (7.3, 7.7, 8.3, 9.0) demuestra que el protocolo cumple robustamente con los principios de **Confidencialidad**, **Integridad** y **Autenticidad** mediante:

- Cifrado simétrico moderno (AES-CTR, AES-GCM, ChaCha20-Poly1305)
- Códigos de autenticación de mensajes (MAC/AEAD) en modo Encrypt-then-MAC
- Autenticación bidireccional mediante firmas digitales y claves asimétricas
- Intercambio seguro de claves (ECDH, Diffie-Hellman)

La evolución observada entre versiones muestra una eliminación progresiva de algoritmos inseguros (diffie-hellman-group14-sha1, hmac-sha1, 3des-cbc) y la incorporación de protecciones modernas, incluyendo algoritmos post-cuánticos (`sntrup761x25519-sha512@openssh.com`) en OpenSSH 9.0.

Las modificaciones realizadas en las Partes 2 y 3 del laboratorio demuestran que:

- La **ocultación de versiones** (Parte 2) dificulta el reconocimiento y fingerprinting
- La **reducción de KEI a 236 bytes** (Parte 3) minimiza la superficie de ataque mediante restricción de algoritmos

Sin embargo, los principios de **Disponibilidad** y **No Repudio** no son objetivos del protocolo SSH y requieren infraestructura adicional externa:

- Disponibilidad: Firewalls, IDS/IPS, balanceadores de carga, rate limiting
- No Repudio: PKI formal, timestamping criptográfico (RFC 3161), logs firmados e inmutables

OpenSSH es un protocolo robusto y maduro que proporciona comunicación segura, pero la configuración adecuada es esencial para minimizar vulnerabilidades como exposición de versiones, fingerprinting HASSH y uso de algoritmos legacy.

Conclusiones y comentarios

El laboratorio permitió comprender el funcionamiento del protocolo SSH mediante análisis de tráfico real. Se observó la evolución del protocolo a través de cuatro versiones de OpenSSH (7.3p1, 7.7p1, 8.3p1 y 9.0p1), identificando mejoras progresivas en algoritmos criptográficos: eliminación de `diffie-hellman-group14-sha1` en C2, ampliación de algoritmos MAC en C3, e incorporación de protección post-cuántica (`sntrup761x25519-sha512@openssh.com`) en C4.

La técnica de fingerprinting mediante HASSH demostró efectividad para identificar versiones específicas de clientes SSH, observándose valores únicos para cada versión analizada. El análisis del tráfico capturado comprobó que SSH proporciona garantías sólidas de confidencialidad (cifrado simétrico AES/ChaCha20), integridad (MAC/AEAD en modo Encrypt-then-MAC) y autenticidad (firmas digitales bidireccionales), pero no cubre completamente disponibilidad (vulnerable a DoS) y no repudio (ausencia de PKI formal y timestamping criptográfico).

La modificación del código fuente en la Parte 2 permitió generar tráfico con versiones personalizadas (`SSH-2.0-OpenSSH_?`), demostrando que la información visible en el handshake es modificable para dificultar reconocimiento. La reducción del Key Exchange Init en la Parte 3 (de 1152 bytes a 236 bytes) mediante restricción de algoritmos mostró el trade-off entre compatibilidad, tamaño de paquetes y seguridad, evidenciando que configuraciones restringidas minimizan la superficie de ataque a costa de interoperabilidad con clientes antiguos.

El laboratorio demostró que, aunque SSH es criptográficamente robusto, la configuración adecuada del servidor (`sshd_config`) es esencial para mitigar vectores de ataque como exposición de versiones, fingerprinting HASSH y uso de algoritmos legacy. Las versiones más recientes de OpenSSH reflejan un desarrollo continuo hacia mayor seguridad, incorporando protecciones contra amenazas emergentes como computación cuántica.