

Informe Laboratorio 5

Sección 2

Cristóbal Barra
cristobal.barra1@mail.udp.cl

Diciembre de 2024

Índice

Descripción de actividades	3
1. Desarrollo (Parte 1)	5
1.1. Códigos de cada Dockerfile	5
1.1.1. C1	6
1.1.2. C2	6
1.1.3. C3	6
1.1.4. C4/S1	7
1.2. Creación de las credenciales para S1	7
1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	8
1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	11
1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	14
1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)	17
1.7. Compara la versión de HASSH obtenida con la base de datos para validar si el cliente corresponde al mismo	20
1.8. Tipo de información contenida en cada uno de los paquetes generados en texto plano	21
1.8.1. C1	21
1.8.2. C2	22
1.8.3. C3	23
1.8.4. C4/S1	25
1.9. Diferencia entre C1 y C2	26
1.10. Diferencia entre C2 y C3	26
1.11. Diferencia entre C3 y C4	27
2. Desarrollo (Parte 2)	28
2.1. Identificación del cliente SSH con versión “?”	28
2.2. Replicación de tráfico al servidor (paso por paso)	28
3. Desarrollo (Parte 3)	28
3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso)	28
4. Desarrollo (Parte 4)	28
4.1. Explicación OpenSSH en general	28
4.2. Capas de Seguridad en OpenSSH	28
4.3. Identificación de que protocolos no se cumplen	29

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 A

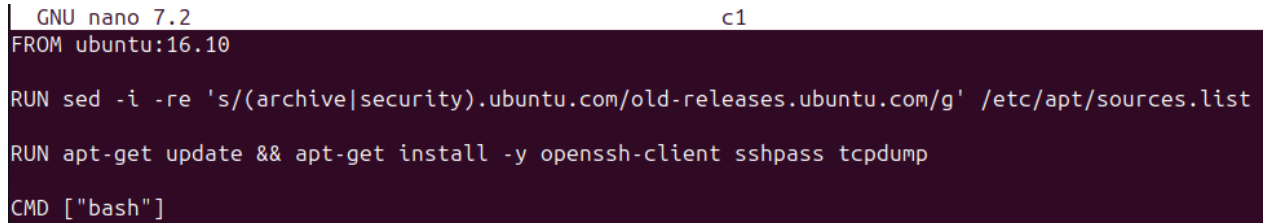
Figura 2: Captura del Key Exchange

- 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 cada Dockerfile respectivo al cliente hay que instalar *openssh-client*, mientras que para el servidor S1 hay que instalar *openssh-server*. Con el fin de poder comunicar los contenedores clientes al contenedor servidor y generar tráfico, en caso del contenedor con Ubuntu 22.10, este hará de cliente y servidor al mismo tiempo. Serán cuatro escenarios, donde cada cliente se conectará al servidor vía protocolo SSH.

1.1.1. C1A terminal window showing the content of a Dockerfile named 'c1'. The text is as follows:

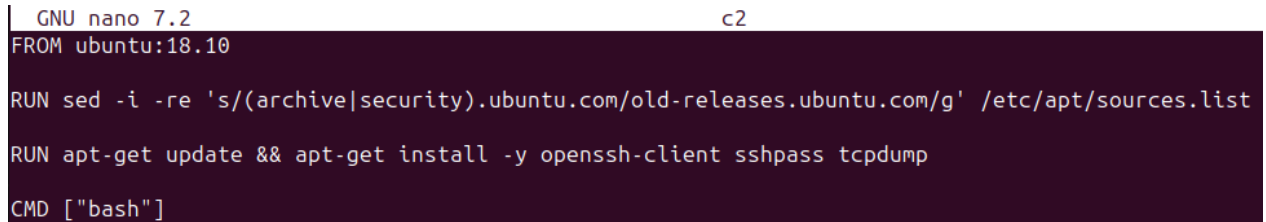
```
GNU nano 7.2 c1
FROM ubuntu:16.10

RUN sed -i -re 's/(archive|security).ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

RUN apt-get update && apt-get install -y openssh-client sshpass tcpdump

CMD ["bash"]
```

Figura 3: Dockerfile C1

1.1.2. C2A terminal window showing the content of a Dockerfile named 'c2'. The text is as follows:

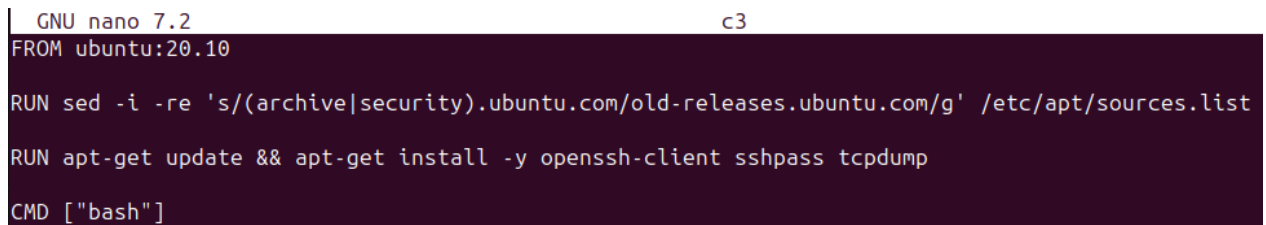
```
GNU nano 7.2 c2
FROM ubuntu:18.10

RUN sed -i -re 's/(archive|security).ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

RUN apt-get update && apt-get install -y openssh-client sshpass tcpdump

CMD ["bash"]
```

Figura 4: Dockerfile C1

1.1.3. C3A terminal window showing the content of a Dockerfile named 'c3'. The text is as follows:

```
GNU nano 7.2 c3
FROM ubuntu:20.10

RUN sed -i -re 's/(archive|security).ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

RUN apt-get update && apt-get install -y openssh-client sshpass tcpdump

CMD ["bash"]
```

Figura 5: Dockerfile C1

1.1.4. C4/S1

```
GNU nano 7.2 c4-s1
FROM ubuntu:22.10

RUN sed -i -re 's/(archive|security).ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

RUN apt-get update && apt-get install -y openssh-server openssh-client sshpass tcpdump

RUN useradd -m prueba && echo 'prueba:prueba' | chpasswd || true

EXPOSE 22

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

Figura 6: Dockerfile C1

1.2. Creación de las credenciales para S1

En la figura 6 se observa que, a través de la línea de comando `RUN useradd -m test && echo 'prueba:prueba' || chpasswd || true`, se ha realizado la creación de las credenciales **prueba:prueba**.

Para comprobar si se crearon correctamente las credenciales, primero debemos conocer la IP del servidor S1, esto se puede lograr con el siguiente comando: `sudo docker inspect -f '{{range .NetworkSettings.Networks}}.IPAddress{{end}}' c4-s1`. Véase en la siguiente figura que el comando retorna la IP del servidor en cuestión, por lo tanto para que un cliente se pueda conectar a éste, debe referirse a esta dirección.

```
ehnr00@ehnr00:~/Cripto/Lab5$ sudo docker inspect -f '{{range
.NetworkSettings.Networks}}.IPAddress{{end}}' c4-s1
172.17.0.2
```

Figura 7: Dirección IP del servidor S1.

Ahora que se conoce la IP de S1, podemos realizar el test de conexión entre un cliente y el servidor, para este caso se escogió el cliente C1. Seguir los pasos que se ven en la siguiente figura referente a la conexión entre C1 y S1, donde el usuario es **'prueba'** y **172.17.0.2** es la IP de destino.

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

1 DESARROLLO (PARTE 1)

```
ehnr00@ehnr00:~/Cripto/Lab5$ sudo docker exec -it c1 bash
root@235a07ce522f:/# ssh prueba@172.17.0.2
prueba@172.17.0.2's password:
Welcome to Ubuntu 22.10 (GNU/Linux 6.8.0-49-generic x86_64)

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

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

To restore this content, you can run the 'unminimize' command.
Last login: Wed Dec  4 01:13:09 2024 from 172.17.0.3
```

Figura 8: Test de conexión entre C1 y S1.

En la figura 8 se ingresó la contraseña 'prueba', seguidamente, el sistema le da la bienvenida al cliente cuya IP es **172.17.0.3**. Con esto se confirma la creación de las credenciales para la realización de este laboratorio.

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

A continuación, en la figura 9, se muestra todo el tráfico generado entre el cliente C1 y el servidor S1 al momento de la conexión entre estos dos, más adelante se realizará una inspección de los paquetes más importantes para tener en cuenta, como también la obtención del HASSH generado a partir de la conexión.

Time	Source	Destination	Protocol	Length	Info
1 0.000000000	172.17.0.3	172.17.0.2	TCP	74	58504 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1288105780 TSecr=0 WS=128
2 0.000045387	172.17.0.2	172.17.0.3	TCP	74	22 → 58504 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=863434919 TSecr=1288105780 WS=128
3 0.000080283	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1288105780 TSecr=863434919
4 0.000575715	172.17.0.3	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1)
5 0.000599034	172.17.0.2	172.17.0.3	TCP	66	22 → 58504 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=863434920 TSecr=1288105781
6 0.001693575	172.17.0.2	172.17.0.3	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7 0.001733898	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=1288105782 TSecr=863434921
8 0.002508312	172.17.0.3	172.17.0.2	SSHv2	1498	Client: Key Exchange Init
9 0.004420819	172.17.0.2	172.17.0.3	SSHv2	1148	Server: Key Exchange Init
10 0.009206197	172.17.0.3	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11 0.016651232	172.17.0.2	172.17.0.3	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
12 0.018688682	172.17.0.3	172.17.0.2	SSHv2	82	Client: New Keys
13 0.059293305	172.17.0.2	172.17.0.3	TCP	66	22 → 58504 [ACK] Seq=1718 Ack=1538 Win=64128 Len=0 TSval=863434979 TSecr=1288105799
14 0.059334297	172.17.0.3	172.17.0.2	SSHv2	110	Client:
15 0.059361570	172.17.0.2	172.17.0.3	TCP	66	22 → 58504 [ACK] Seq=1718 Ack=1582 Win=64128 Len=0 TSval=863434979 TSecr=1288105840
16 0.059488763	172.17.0.2	172.17.0.3	SSHv2	110	Server:
17 0.059575939	172.17.0.3	172.17.0.2	SSHv2	134	Client:
18 0.065797666	172.17.0.2	172.17.0.3	SSHv2	118	Server:
19 0.106335428	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=1650 Ack=1814 Win=64128 Len=0 TSval=1288105887 TSecr=863434985
21 0.076594174	172.17.0.3	172.17.0.2	SSHv2	214	Client:
22 0.117253289	172.17.0.2	172.17.0.3	TCP	66	22 → 58504 [ACK] Seq=1814 Ack=1798 Win=64128 Len=0 TSval=863436037 TSecr=1288106857
23 0.125469244	172.17.0.2	172.17.0.3	SSHv2	94	Server:
24 0.125477829	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=1798 Ack=1842 Win=64128 Len=0 TSval=1288106906 TSecr=863436045
25 0.125542640	172.17.0.3	172.17.0.2	SSHv2	178	Client:
26 0.125550181	172.17.0.2	172.17.0.3	TCP	66	22 → 58504 [ACK] Seq=1842 Ack=1910 Win=64128 Len=0 TSval=863436045 TSecr=1288106906
27 0.131977591	172.17.0.2	172.17.0.3	SSHv2	694	Server:
28 0.172291069	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=1910 Ack=2470 Win=64128 Len=0 TSval=1288106953 TSecr=863436051
29 0.172328935	172.17.0.2	172.17.0.3	SSHv2	110	Server:
30 0.172349456	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=1910 Ack=2514 Win=64128 Len=0 TSval=1288106953 TSecr=863436092
31 0.172532419	172.17.0.3	172.17.0.2	SSHv2	442	Client:
32 0.174661094	172.17.0.2	172.17.0.3	SSHv2	174	Server:
33 0.175002898	172.17.0.2	172.17.0.3	SSHv2	566	Server:
34 0.175041057	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=2286 Ack=3122 Win=64128 Len=0 TSval=1288106955 TSecr=863436094
35 0.182560057	172.17.0.2	172.17.0.3	SSHv2	102	Server:
36 0.223310609	172.17.0.3	172.17.0.2	TCP	66	58504 → 22 [ACK] Seq=2286 Ack=3158 Win=64128 Len=0 TSval=1288107004 TSecr=863436102

Figura 9: Tráfico generado entre C1 y S1.

En el intercambio de claves SSH, se observan diferentes tamaños de paquetes a medida que avanza el proceso. Al inicio de la conexión, el cliente y el servidor intercambian paquetes de pequeña longitud, como el paquete **SYN** (74 bytes) y el **SYN-ACK** del servidor (74 bytes). Estos paquetes son necesarios para establecer la conexión TCP básica.

Una vez que se establece la conexión, el cliente envía su versión del protocolo SSH en un paquete con un tamaño de 107 bytes, que incluye la cadena **SSH-2.0-OpenSSH.7.3p1 Ubuntu-1ubuntu0.1**. Este paquete contiene la versión del protocolo y es relativamente pequeño, pesando cliente y servidor 107 bytes.

En el proceso de intercambio de claves, los paquetes empiezan a ser más grandes debido a la complejidad criptográfica. El cliente envía un paquete de **Key Exchange Init** con un tamaño de 1498 bytes. Este paquete incluye los parámetros necesarios para la negociación de algoritmos de cifrado y autenticación.

El servidor responde con su propio paquete, que también es grande, alcanzando los 1146 bytes. Este paquete contiene los datos necesarios para continuar el intercambio de claves.

Durante el intercambio Diffie-Hellman, se envían paquetes aún más grandes. El paquete de **Elliptic Curve Diffie-Hellman Key Exchange Init** del cliente tiene un tamaño de 114 bytes, mientras que la respuesta del servidor, **Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys**, alcanza los 662 bytes, el mayor tamaño es debido a las nuevas llaves que se generan.

Finalmente, después de completar el intercambio de claves, se envían paquetes más pequeños para confirmar que las claves de sesión han sido actualizadas correctamente. El paquete **New Keys** del cliente tiene un tamaño de 82 bytes, y la respuesta del servidor tiene un tamaño de 110 bytes. Los siguientes paquetes tienen un peso similar a estos paquetes recién visto.

```

SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:
    Packet Length: 1428
    Padding Length: 11
  Key Exchange (method:curve25519-sha256@libssh.org)
    Message Code: Key Exchange Init (20)
    Algorithms
      Cookie: c5f5690a09da1cfddf589b9bfb63040c
      kex_algorithms length: 286
      kex_algorithms string [truncated]: curve25519-sha256@libssh.
      server_host_key_algorithms length: 290
      server_host_key_algorithms string [truncated]: ecdsa-sha2-ni
      encryption_algorithms_client_to_server length: 150
      encryption_algorithms_client_to_server string: chacha20-poly
      encryption_algorithms_server_to_client length: 150
      encryption_algorithms_server_to_client string: chacha20-poly
      mac_algorithms_client_to_server length: 213
      mac_algorithms_client_to_server string [truncated]: umac-64-
      mac_algorithms_server_to_client length: 213
      mac_algorithms_server_to_client string [truncated]: umac-64-
      compression_algorithms_client_to_server length: 26
      compression_algorithms_client_to_server string: none,zlib@op
      compression_algorithms_server_to_client length: 26
      compression_algorithms_server_to_client string: none,zlib@op
      languages_client_to_server length: 0
      languages_client_to_server string:
      languages_server_to_client length: 0
      languages_server_to_client string:
      First KEX Packet Follows: 0
      Reserved: 00000000
      [hasshAlgorithms [truncated]: curve25519-sha256@libssh.org,e
      [hassh: 0e4584cb9f2dd077dbf8ba0df8112d8e]
  
```

Figura 10: HASSH generado por C1.

En la figura 10 se observa la información del paquete n° 8 referente a la inicialización del intercambio de llaves. El mensaje inicial para el intercambio de claves, que utiliza el algoritmo de intercambio `curve25519-sha256@libssh.org`, contiene varios parámetros clave como los algoritmos de intercambio de claves, cifrado y MAC (código de autenticación de mensaje). A partir de estos parámetros, se calcula el HASSH, un valor único que representa la integridad del proceso de intercambio de claves y establece la base para la autenticación de la sesión.

Al final de la figura se logra apreciar el HASSH del cliente, con el valor correspondiente a `0e4584cb9f2dd077dbf8ba0df8112d8e`, y es utilizado tanto por el cliente como por el servidor para verificar la autenticidad de la conexión, garantizando que los datos intercambiados no han sido alterados ni comprometidos durante el establecimiento de la sesión.

```

SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:
    Packet Length: 1076
    Padding Length: 7
  Key Exchange (method:curve25519-sha256@libssh.org)
    Message Code: Key Exchange Init (20)
  Algorithms
    Cookie: 7ac01f75f1e052cf4286a2184e087d85
    kex_algorithms length: 265
    kex_algorithms string [truncated]: sntrup761x25519-sha512@op
    server_host_key_algorithms length: 57
    server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256
    encryption_algorithms_client_to_server length: 108
    encryption_algorithms_client_to_server string: chacha20-poly
    encryption_algorithms_server_to_client length: 108
    encryption_algorithms_server_to_client string: chacha20-poly
    mac_algorithms_client_to_server length: 213
    mac_algorithms_client_to_server string [truncated]: umac-64-
    mac_algorithms_server_to_client length: 213
    mac_algorithms_server_to_client string [truncated]: umac-64-
    compression_algorithms_client_to_server length: 21
    compression_algorithms_client_to_server string: none,zlib@op
    compression_algorithms_server_to_client length: 21
    compression_algorithms_server_to_client string: none,zlib@op
    languages_client_to_server length: 0
    languages_client_to_server string:
    languages_server_to_client length: 0
    languages_server_to_client string:
    First KEX Packet Follows: 0
    Reserved: 00000000
    [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@c
    [hasshServer: a984ff804585fabe3cd08f4b3849024a]
  
```

Figura 11: HASSH generado por S1 para la conexión con C1.

Para el caso del HASSH generado por el servidor, es muy similar la manera de encontrar este valor, sin embargo, tiene otro nombre dentro del paquete, el cuales `hasshServer`. Pero al igual que C1, éste se encuentra al final de la figura 11, cuyo valor es `a984ff804585fabe3cd08f4b3849024a`.

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

Tanto para este cliente como los siguientes, se realizará la misma descripción de su tráfico, por lo que puede ser un poco redundante la información que se muestre, dado que son los mismo paquetes que se generan pero con ligeras diferencias.

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

1 DESARROLLO (PARTE 1)

Time	Source	Destination	Protocol	Length	Info
1 0.000000000	172.17.0.4	172.17.0.2	TCP	74	52408 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1013819296 TSecr=0 WS=128
2 0.000046664	172.17.0.2	172.17.0.4	TCP	74	22 → 52408 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=598289973 TSecr=1013819296 WS=128
3 0.000080998	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1013819296 TSecr=598289973
4 0.000451330	172.17.0.4	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3)
5 0.000472477	172.17.0.2	172.17.0.4	TCP	66	22 → 52408 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=598289973 TSecr=1013819296
6 0.001997471	172.17.0.2	172.17.0.4	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7 0.002029783	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=1013819298 TSecr=598289975
8 0.002942999	172.17.0.4	172.17.0.2	SSHv2	1426	Client: Key Exchange Init
9 0.004683162	172.17.0.2	172.17.0.4	SSHv2	1146	Server: Key Exchange Init
10 0.010171766	172.17.0.4	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
11 0.023700137	172.17.0.2	172.17.0.4	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
12 0.030182141	172.17.0.4	172.17.0.2	SSHv2	82	Client: New Keys
13 0.070843844	172.17.0.2	172.17.0.4	TCP	66	22 → 52408 [ACK] Seq=1718 Ack=1466 Win=64128 Len=0 TSval=598290044 TSecr=1013819326
14 0.070887500	172.17.0.4	172.17.0.2	SSHv2	110	Client:
15 0.070909925	172.17.0.2	172.17.0.4	TCP	66	22 → 52408 [ACK] Seq=1718 Ack=1510 Win=64128 Len=0 TSval=598290044 TSecr=1013819367
16 0.071068265	172.17.0.2	172.17.0.4	SSHv2	110	Server:
17 0.071178563	172.17.0.4	172.17.0.2	SSHv2	134	Client:
18 0.077398291	172.17.0.2	172.17.0.4	SSHv2	118	Server:
19 0.117849821	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=1578 Ack=1814 Win=64128 Len=0 TSval=1013819414 TSecr=598290050
20 1.104973698	172.17.0.4	172.17.0.2	SSHv2	214	Client:
21 1.145788344	172.17.0.2	172.17.0.4	TCP	66	22 → 52408 [ACK] Seq=1814 Ack=1726 Win=64128 Len=0 TSval=598291119 TSecr=1013820401
22 1.153877115	172.17.0.2	172.17.0.4	SSHv2	94	Server:
23 1.153888077	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=1726 Ack=1842 Win=64128 Len=0 TSval=1013820450 TSecr=598291127
24 1.153952397	172.17.0.4	172.17.0.2	SSHv2	178	Client:
25 1.153958987	172.17.0.2	172.17.0.4	TCP	66	22 → 52408 [ACK] Seq=1842 Ack=1838 Win=64128 Len=0 TSval=598291127 TSecr=1013820450
26 1.161144152	172.17.0.2	172.17.0.4	SSHv2	694	Server:
27 1.201828369	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=1838 Ack=2470 Win=64128 Len=0 TSval=1013820498 TSecr=598291134
28 1.201871738	172.17.0.2	172.17.0.4	SSHv2	110	Server:
29 1.201900899	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=1838 Ack=2514 Win=64128 Len=0 TSval=1013820498 TSecr=598291175
30 1.202106956	172.17.0.4	172.17.0.2	SSHv2	442	Client:
31 1.204414155	172.17.0.2	172.17.0.4	SSHv2	174	Server:
32 1.204852629	172.17.0.2	172.17.0.4	SSHv2	566	Server:
33 1.204907637	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=2214 Ack=3122 Win=64128 Len=0 TSval=1013820501 TSecr=598291177
34 1.212325110	172.17.0.2	172.17.0.4	SSHv2	102	Server:
35 1.252839902	172.17.0.4	172.17.0.2	TCP	66	52408 → 22 [ACK] Seq=2214 Ack=3158 Win=64128 Len=0 TSval=1013820549 TSecr=598291185

Figura 12: Tráfico generado entre C2 y S1.

El cliente C2 inicia la conexión enviando un paquete SSH que incluye la versión del protocolo: "SSH-2.0-OpenSSH.7.7p1 Ubuntu-4ubuntu0.3". Este paquete tiene un tamaño de 107 bytes en la red y se transmite al servidor con una dirección de destino 172.17.0.2.

En respuesta, el servidor envía un paquete del mismo tamaño, 107 bytes, indicando su propia versión del protocolo: "SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3". Este intercambio inicial establece la compatibilidad del protocolo entre el cliente y el servidor.

A continuación, el cliente envía un paquete más grande, de 1426 bytes, que corresponde al mensaje "Key Exchange Init". Este paquete contiene los parámetros necesarios para la negociación de algoritmos de cifrado. El servidor responde con un paquete de 1146 bytes, proporcionando su propio conjunto de parámetros para el intercambio de claves.

El proceso continúa con el cliente enviando un paquete de 114 bytes que inicia el intercambio de claves basado en Diffie-Hellman. El servidor responde con un paquete de 662 bytes, que incluye su parte del cálculo Diffie-Hellman y un mensaje "New Keys".

Para completar el establecimiento de claves, el cliente envía un paquete de 82 bytes confirmando la instalación de las nuevas claves. Poco después, el cliente y el servidor intercambian paquetes adicionales de 110 bytes cada uno, confirmando el estado final de la conexión.

Para identificar el HASSH generado por el cliente y el servidor, referente a su handshake, seguiremos el mismo procedimiento de la subsección anterior (tráfico entre C1 y S1). Donde, en el paquete 8 y 9, referente a la inicialización del intercambio de llaves, al final del paquete, se encuentran los campos que hacen alusión al HASSH generados tanto por el cliente como

por el servidor. Siendo el HASSH del cliente el valor 06046964c022c6407d15a27b12a6a4fb, mientras que para el servidor es a984ff804585fabe3cd08f4b3849024a. Estos valores se encuentran dentro de los campos **hassh** y **hasshServer** respectivamente, a continuación se encuentran las figuras en las cuales se puede apreciar lo mencionado anteriormente.

```

SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1356
    Padding Length: 5
    Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      Algorithms
        Cookie: 15fcd1e98c25a04e145132d212bd7523
        kex_algorithms length: 304
        kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha
        server_host_key_algorithms length: 290
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@op
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@op
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@ope
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@ope
        compression_algorithms_client_to_server length: 26
        compression_algorithms_client_to_server string: none,zlib@openssh.c
        compression_algorithms_server_to_client length: 26
        compression_algorithms_server_to_client string: none,zlib@openssh.c
        languages_client_to_server length: 0
        languages_client_to_server string:
        languages_server_to_client length: 0
        languages_server_to_client string:
        First KEX Packet Follows: 0
        Reserved: 00000000
        [hasshAlgorithms [truncated]: curve25519-sha256,curve25519-sha256@l
        [hassh: 06046964c022c6407d15a27b12a6a4fb]
        Padding String: 0000000000
        Sequence number: 0
        [Direction: client-to-server]

```

Figura 13: HASSH generado por C2 para la conexión con el servidor.

```

SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
    Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      Algorithms
        Cookie: 621db886b766c86595632438f361588f
        kex_algorithms length: 265
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.c
        server_host_key_algorithms length: 57
        server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@op
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@op
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@ope
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@ope
        compression_algorithms_client_to_server length: 21
        compression_algorithms_client_to_server string: none,zlib@openssh.c
        compression_algorithms_server_to_client length: 21
        compression_algorithms_server_to_client string: none,zlib@openssh.c
        languages_client_to_server length: 0
        languages_client_to_server string:
        languages_server_to_client length: 0
        languages_server_to_client string:
        First KEX Packet Follows: 0
        Reserved: 00000000
        [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.
        [hasshServer: a984ff804585fabe3cd08f4b3849024a]
        Padding String: 0000000000000000
        Sequence number: 0
        [Direction: server-to-client]

```

Figura 14: HASSH generado por el servidor para la conexión con C2.

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

Con respecto al tráfico generado entre el cliente C3 y el servidor S1, se sigue el mismo procedimiento para el flujo de intercambio de llaves, a continuación se describirán los pesos de los paquetes más importantes en este handshake y también la obtención de los HASSH generados por parte de ambos.

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

1 DESARROLLO (PARTE 1)

Time	Source	Destination	Protocol	Length	Info
3 0.000162745	172.17.0.5	172.17.0.2	TCP	74	59502 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=53942496 TSecr=0 WS=128
4 0.000209697	172.17.0.2	172.17.0.5	TCP	74	22 → 59502 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1595607405 TSecr=53942496 WS=128
5 0.000244547	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=53942496 TSecr=1595607405
6 0.000844846	172.17.0.5	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1)
7 0.000866184	172.17.0.2	172.17.0.5	TCP	66	22 → 59502 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=1595607406 TSecr=53942497
8 0.001907581	172.17.0.2	172.17.0.5	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
9 0.001943253	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=53942498 TSecr=1595607407
10 0.002853799	172.17.0.5	172.17.0.2	SSHv2	1578	Client: Key Exchange Init
11 0.004607646	172.17.0.2	172.17.0.5	SSHv2	1146	Server: Key Exchange Init
12 0.010264038	172.17.0.5	172.17.0.2	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
13 0.020582138	172.17.0.2	172.17.0.5	SSHv2	662	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
14 0.025420750	172.17.0.5	172.17.0.2	SSHv2	82	Client: New Keys
15 0.065474480	172.17.0.2	172.17.0.5	TCP	66	22 → 59502 [ACK] Seq=1718 Ack=1618 Win=64128 Len=0 TSval=1595607471 TSecr=53942521
16 0.065494079	172.17.0.5	172.17.0.2	SSHv2	110	Client:
17 0.065504481	172.17.0.2	172.17.0.5	TCP	66	22 → 59502 [ACK] Seq=1718 Ack=1662 Win=64128 Len=0 TSval=1595607471 TSecr=53942562
18 0.065581588	172.17.0.2	172.17.0.5	SSHv2	110	Server:
19 0.065631384	172.17.0.5	172.17.0.2	SSHv2	134	Client:
20 0.071813745	172.17.0.2	172.17.0.5	SSHv2	118	Server:
21 0.112510933	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=1730 Ack=1814 Win=64128 Len=0 TSval=53942609 TSecr=1595607477
22 1.361461938	172.17.0.5	172.17.0.2	SSHv2	214	Client:
23 1.403498461	172.17.0.2	172.17.0.5	TCP	66	22 → 59502 [ACK] Seq=1814 Ack=1878 Win=64128 Len=0 TSval=1595608809 TSecr=53943857
24 1.410386184	172.17.0.2	172.17.0.5	SSHv2	94	Server:
25 1.410396522	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=1878 Ack=1842 Win=64128 Len=0 TSval=53943906 TSecr=1595608815
26 1.410473768	172.17.0.5	172.17.0.2	SSHv2	178	Client:
27 1.410480355	172.17.0.2	172.17.0.5	TCP	66	22 → 59502 [ACK] Seq=1842 Ack=1990 Win=64128 Len=0 TSval=1595608816 TSecr=53943907
28 1.415961405	172.17.0.2	172.17.0.5	SSHv2	694	Server:
29 1.456579161	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=1990 Ack=2470 Win=64128 Len=0 TSval=53943953 TSecr=1595608821
30 1.456619390	172.17.0.2	172.17.0.5	SSHv2	110	Server:
31 1.456640076	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=1990 Ack=2514 Win=64128 Len=0 TSval=53943953 TSecr=1595608862
32 1.456817281	172.17.0.5	172.17.0.2	SSHv2	442	Client:
33 1.458930174	172.17.0.2	172.17.0.5	SSHv2	174	Server:
34 1.459301867	172.17.0.2	172.17.0.5	SSHv2	566	Server:
35 1.459368199	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=2366 Ack=3122 Win=64128 Len=0 TSval=53943955 TSecr=1595608864
36 1.466439419	172.17.0.2	172.17.0.5	SSHv2	102	Server:
37 1.508467348	172.17.0.5	172.17.0.2	TCP	66	59502 → 22 [ACK] Seq=2366 Ack=3158 Win=64128 Len=0 TSval=53944005 TSecr=1595608871

Figura 15: Tráfico generado entre C3 y S1.

El cliente C3 inicia la conexión enviando un paquete SSH que indica la versión del protocolo: "SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1". Este paquete tiene un tamaño de 107 bytes y se dirige al servidor en la dirección 172.17.0.2.

El servidor responde con un paquete también de 107 bytes, confirmando su versión del protocolo: "SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3". Esto establece la compatibilidad inicial entre cliente y servidor.

Posteriormente, el cliente envía un paquete de 1578 bytes que contiene el mensaje "Key Exchange Init", iniciando la negociación de los algoritmos de cifrado. El servidor responde con un paquete de 1146 bytes que incluye sus parámetros para el intercambio de claves.

El cliente prosigue con un paquete de 114 bytes que inicia el intercambio de claves basado en Diffie-Hellman. El servidor responde con un paquete de 662 bytes, donde incluye su parte del cálculo y el mensaje "New Keys".

Para finalizar el proceso, el cliente envía un paquete de 82 bytes confirmando la instalación de las nuevas claves. Finalmente, ambos extremos intercambian paquetes de 110 bytes cada uno, validando la conexión establecida.

En los paquetes 10 y 11 mostrados en la figura 15, se encuentran los HASSH que generan ambas partes para el intercambio de llaves.

```

SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1508
    Padding Length: 10
    Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      Algorithms
        Cookie: 5a46b677c9b166a508d643cab037cea4
        kex_algorithms length: 241
        kex_algorithms string [truncated]: curve25519-sha256,curve25519-sha
        server_host_key_algorithms length: 500
        server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@op
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@op
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@ope
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@ope
        compression_algorithms_client_to_server length: 26
        compression_algorithms_client_to_server string: none,zlib@openssh.c
        compression_algorithms_server_to_client length: 26
        compression_algorithms_server_to_client string: none,zlib@openssh.c
        languages_client_to_server length: 0
        languages_client_to_server string:
        languages_server_to_client length: 0
        languages_server_to_client string:
        First KEX Packet Follows: 0
        Reserved: 00000000
        [hasshAlgorithms [truncated]: curve25519-sha256,curve25519-sha256@l
        [hassh: ae8bd7dd09970555aa4c6ed22adbbf56]
        Padding String: 00000000000000000000
        Sequence number: 0
        [Direction: client-to-server]

```

Figura 16: HASSH generado por C3 para la conexión con el servidor.


```

SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
    Key Exchange (method:curve25519-sha256)
      Message Code: Key Exchange Init (20)
      Algorithms
        Cookie: 5e9f40f8b2697922544bd681133040c2
        kex_algorithms length: 265
        kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.c
        server_host_key_algorithms length: 57
        server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-
        encryption_algorithms_client_to_server length: 108
        encryption_algorithms_client_to_server string: chacha20-poly1305@op
        encryption_algorithms_server_to_client length: 108
        encryption_algorithms_server_to_client string: chacha20-poly1305@op
        mac_algorithms_client_to_server length: 213
        mac_algorithms_client_to_server string [truncated]: umac-64-etm@ope
        mac_algorithms_server_to_client length: 213
        mac_algorithms_server_to_client string [truncated]: umac-64-etm@ope
        compression_algorithms_client_to_server length: 21
        compression_algorithms_client_to_server string: none,zlib@openssh.c
        compression_algorithms_server_to_client length: 21
        compression_algorithms_server_to_client string: none,zlib@openssh.c
        languages_client_to_server length: 0
        languages_client_to_server string:
        languages_server_to_client length: 0
        languages_server_to_client string:
        First KEX Packet Follows: 0
        Reserved: 00000000
        [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.
        [hasshServer: a984ff804585fabe3cd08f4b3849024a]
        Padding String: 00000000000000
        Sequence number: 0
        [Direction: server-to-client]

```

Figura 17: HASSH generado por el servidor para la conexión con C3.

Refiriéndose a la obtención de los HASSH generados por el cliente C3 y el servidor S1, se pueden observar en las figuras 16 y 17. Donde el HASSH generado por el cliente es ae8bd7dd09970555aa4c6ed22adbbf56, mientras que el generado por el servidor es el siguiente: a984ff804585fabe3cd08f4b3849024a.

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

El tráfico entre C4 y S1 se debe capturar desde la interfaz **Loopback**, y no **docker0** como se hizo con anterioridad, puesto que el tráfico se genera en un solo contenedor y no entre dos contenedores como antes. El tráfico en este caso sigue el mismo patrón, con la única diferencia de que es solo una IP la que está haciendo el trabajo, en este caso, solo la IP **172.17.0.2**, correspondiente a la dirección del contenedor.

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

1 DESARROLLO (PARTE 1)

Time	Source	Destination	Protocol	Length	Info
1 0.000000	172.17.0.2	172.17.0.2	TCP	74	33902 → 22 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4049610321 TSecr=0 WS=128
2 0.000025	172.17.0.2	172.17.0.2	TCP	74	22 → 33902 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=4049610321 TSecr=4049610321 WS=128
3 0.000133	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4049610321 TSecr=4049610321
4 0.000559	172.17.0.2	172.17.0.2	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
5 0.000569	172.17.0.2	172.17.0.2	TCP	66	22 → 33902 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=4049610322 TSecr=4049610322
6 0.001800	172.17.0.2	172.17.0.2	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3)
7 0.001879	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=42 Ack=42 Win=65536 Len=0 TSval=4049610323 TSecr=4049610323
8 0.002385	172.17.0.2	172.17.0.2	SSHv2	1570	Client: Key Exchange Init
9 0.004821	172.17.0.2	172.17.0.2	SSHv2	1146	Server: Key Exchange Init
10 0.045059	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=1546 Ack=1122 Win=65536 Len=0 TSval=4049610367 TSecr=4049610326
11 0.051266	172.17.0.2	172.17.0.2	SSHv2	1274	Client: Diffie-Hellman Key Exchange Init
12 0.058758	172.17.0.2	172.17.0.2	SSHv2	1630	Server: Diffie-Hellman Key Exchange Reply, New Keys
13 0.058762	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=2754 Ack=2686 Win=65536 Len=0 TSval=4049610380 TSecr=4049610380
14 2.143357	172.17.0.2	172.17.0.2	SSHv2	82	Client: New Keys
15 2.184088	172.17.0.2	172.17.0.2	TCP	66	22 → 33902 [ACK] Seq=2686 Ack=2770 Win=65536 Len=0 TSval=4049612506 TSecr=4049612465
16 2.184117	172.17.0.2	172.17.0.2	SSHv2	110	Client:
17 2.184130	172.17.0.2	172.17.0.2	TCP	66	22 → 33902 [ACK] Seq=2686 Ack=2814 Win=65536 Len=0 TSval=4049612506 TSecr=4049612506
18 2.184339	172.17.0.2	172.17.0.2	SSHv2	110	Server:
19 2.184363	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=2814 Ack=2730 Win=65536 Len=0 TSval=4049612506 TSecr=4049612506
20 2.184448	172.17.0.2	172.17.0.2	SSHv2	134	Client:
21 2.191104	172.17.0.2	172.17.0.2	SSHv2	118	Server:
22 2.232088	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=2882 Ack=2782 Win=65536 Len=0 TSval=4049612554 TSecr=4049612513
23 4.201825	172.17.0.2	172.17.0.2	SSHv2	214	Client:
24 4.242103	172.17.0.2	172.17.0.2	TCP	66	22 → 33902 [ACK] Seq=2782 Ack=3030 Win=65536 Len=0 TSval=4049614564 TSecr=4049614523
25 4.254369	172.17.0.2	172.17.0.2	SSHv2	94	Server:
26 4.254376	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=3030 Ack=2810 Win=65536 Len=0 TSval=4049614576 TSecr=4049614576
27 4.254441	172.17.0.2	172.17.0.2	SSHv2	178	Client:
28 4.254445	172.17.0.2	172.17.0.2	TCP	66	22 → 33902 [ACK] Seq=2810 Ack=3142 Win=65536 Len=0 TSval=4049614576 TSecr=4049614576
29 4.264006	172.17.0.2	172.17.0.2	SSHv2	694	Server:
30 4.264476	172.17.0.2	172.17.0.2	SSHv2	646	Client:
31 4.264487	172.17.0.2	172.17.0.2	SSHv2	110	Server:
32 4.264534	172.17.0.2	172.17.0.2	SSHv2	442	Client:
33 4.269963	172.17.0.2	172.17.0.2	SSHv2	606	Server:
34 4.278357	172.17.0.2	172.17.0.2	SSHv2	174	Server:
35 4.278397	172.17.0.2	172.17.0.2	TCP	66	33902 → 22 [ACK] Seq=4098 Ack=4130 Win=65536 Len=0 TSval=4049614600 TSecr=4049614591
36 4.278775	172.17.0.2	172.17.0.2	SSHv2	782	Server:
37 4.285264	172.17.0.2	172.17.0.2	SSHv2	102	Server:

Figura 18: Tráfico generado entre C4 y S1.

Tal como se ve en la figura 18, el cliente envió inicialmente su mensaje de protocolo SSH, SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3, con un peso de 107 bytes. El servidor respondió con un mensaje de protocolo equivalente, también de 107 bytes.

Posteriormente, el cliente envió un paquete de inicialización que contenía las preferencias de algoritmos para el intercambio de claves, cifrado y compresión. Este paquete tenía un peso de 1570 bytes. El servidor respondió con un paquete de inicialización similar, cuyo peso fue de 1146 bytes.

En el intercambio de claves Diffie-Hellman, el cliente envió un paquete con su clave pública, cuyo peso fue de 1160 bytes. El servidor, en respuesta, envió un paquete de 1630 bytes que incluía su clave pública.

Finalmente, el cliente envió un paquete **New Keys** para confirmar el éxito del intercambio de claves. Con un peso de 82 bytes.

Ahora, con respecto a los HASSH del cliente y el servidor, estos se pueden obtener de la misma manera que vista en las tres contenedores anteriores. El valor del HASSH que se generó para el cliente C4 es 78c05d999799066a2b4554ce7b1585a6, mientras que para el servidor S1 es a984ff804585fabe3cd08f4b3849024a. Estos valores se pueden encontrar en las figuras 19 y 20 que se muestran a continuación.

```

    ▾ SSH Protocol
      ▾ SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
        Packet Length: 1500
        Padding Length: 4
      ▾ Key Exchange (method:sntrup761x25519-sha512@openssh.com)
        Message Code: Key Exchange Init (20)
        ▾ Algorithms
          Cookie: 7799d91b02fdce1400248acc9ffc17dd
          kex_algorithms length: 276
          kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.c
          server_host_key_algorithms length: 463
          server_host_key_algorithms string [truncated]: ssh-ed25519-cert-v01
          encryption_algorithms_client_to_server length: 108
          encryption_algorithms_client_to_server string: chacha20-poly1305@op
          encryption_algorithms_server_to_client length: 108
          encryption_algorithms_server_to_client string: chacha20-poly1305@op
          mac_algorithms_client_to_server length: 213
          mac_algorithms_client_to_server string [truncated]: umac-64-etm@ope
          mac_algorithms_server_to_client length: 213
          mac_algorithms_server_to_client string [truncated]: umac-64-etm@ope
          compression_algorithms_client_to_server length: 26
          compression_algorithms_client_to_server string: none,zlib@openssh.c
          compression_algorithms_server_to_client length: 26
          compression_algorithms_server_to_client string: none,zlib@openssh.c
          languages_client_to_server length: 0
          languages_client_to_server string:
          languages_server_to_client length: 0
          languages_server_to_client string:
          First KEX Packet Follows: 0
          Reserved: 00000000
          [hasshAlgorithms [truncated]: sntrup761x25519-sha512@openssh.com,cu
          [hassh: 78c05d999799066a2b4554ce7b1585a6]
          Padding String: 00000000
          Sequence number: 0
          [Direction: client-to-server]
    
```

Figura 19: HASSH generado por C4 para la conexión con el servidor.

```

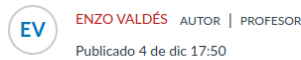
SSH Protocol
  SSH Version 2 (encryption:chacha20-poly1305@openssh.com compression:none)
    Packet Length: 1076
    Padding Length: 7
  Key Exchange (method:sntrup761x25519-sha512@openssh.com)
    Message Code: Key Exchange Init (20)
  Algorithms
    Cookie: 1c908d6e8559b33c90a136c10eb42a3d
    kex_algorithms string [truncated]: sntrup761x25519-sha512@openssh.c
    server_host_key_algorithms length: 57
    server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ecdsa-
    encryption_algorithms_client_to_server length: 108
    encryption_algorithms_client_to_server string: chacha20-poly1305@op
    encryption_algorithms_server_to_client length: 108
    encryption_algorithms_server_to_client string: chacha20-poly1305@op
    mac_algorithms_client_to_server length: 213
    mac_algorithms_client_to_server string [truncated]: umac-64-etm@ope
    mac_algorithms_server_to_client length: 213
    mac_algorithms_server_to_client string [truncated]: umac-64-etm@ope
    compression_algorithms_client_to_server length: 21
    compression_algorithms_client_to_server string: none,zlib@openssh.c
    compression_algorithms_server_to_client length: 21
    compression_algorithms_server_to_client string: none,zlib@openssh.c
    languages_client_to_server length: 0
    languages_client_to_server string:
    languages_server_to_client length: 0
    languages_server_to_client string:
    First KEX Packet Follows: 0
    Reserved: 00000000
    [hasshServerAlgorithms [truncated]: sntrup761x25519-sha512@openssh.
    [hasshServer: a984ff804585fabe3cd08f4b3849024a]
    Padding String: 00000000000000
    Sequence number: 0
    [Direction: server-to-client]
  
```

Figura 20: HASSH generado por el servidor para la conexión con C4.

Si algo se logra ver, después de analizar el tráfico de los cuatro contenedores, es que para todos los handshakes con los clientes, el servidor S1 usó el mismo HASSH. Esto se debe a que el servidor fue configurado de una sola manera, tanto algoritmos de cifrado como de intercambio de llaves, por lo que el servidor mantiene su HASSH a pesar de tratar con diferentes clientes.

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

Como se anunció hace unos días, el profesor pidió omitir esta sección, por lo que no se realizará el procedimiento de esta parte del laboratorio y se pasará a la siguiente sección correspondiente.



Error en Lab 5

Estimados,

Les pido omitir la instrucción que pide lo siguiente:

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.

Disculpen los inconvenientes.

Figura 21: Omisión de la subsección 1.7.

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

Antes de pasar a la descripción detallada de cada paquete de cada cliente, cabe destacar que los campos a describir son los mismos, aunque puede que varíen sus valores en algunos casos, por lo que la información mostrada a continuación puede parecer redundante o duplicada, pero es simplemente los valores que muestran cada paquete del intercambio entre cliente y servidor.

1.8.1. C1

■ Paquete 4

- Versión de Protocolo: SSH-2.0-OpenSSH.7.3p1 Ubuntu-1ubuntu0.1

■ Paquete 6

- Versión de Protocolo: SSH-2.0-OpenSSH.9.0p1 Ubuntu-1ubuntu7.3

■ Paquete 8 (Key Exchange Init - Cliente envía mensaje de inicio de intercambio de llaves):

● Algoritmos utilizados:

- Algoritmo de intercambio de claves: `diffie-hellman-group14-sha1`
- Algoritmo de cifrado para la sesión: `aes128-ctr`
- Algoritmo de autenticación de mensaje: `hmac-sha1`

■ Paquete 9 (Key Exchange Init - Servidor responde al intercambio):

● Algoritmos negociados:

- Algoritmo de intercambio de claves: `diffie-hellman-group14-sha1`
- Algoritmo de cifrado para la sesión: `aes128-ctr`
- Algoritmo de autenticación de mensaje: `hmac-sha1`

- **Paquete 10 (Key Exchange y Diffie-Hellman - Cliente envía su parte de la clave Diffie-Hellman):**
 - **Clave Diffie-Hellman:**
 - Se genera una clave pública de largo 32 bytes
 - El cliente envía su clave pública como parte del intercambio de claves Diffie-Hellman.
 - La verificación de la integridad de los datos se realiza utilizando el algoritmo `hmac-sha1`.
- **Paquete 11 (Key Exchange y Diffie-Hellman, New Keys - Servidor responde con su parte de la clave Diffie-Hellman, también responde con las nuevas claves):**
 - **Clave Diffie-Hellman:**
 - Se genera una clave pública de largo 32 bytes
 - El servidor envía su clave pública como parte del intercambio de claves Diffie-Hellman.
 - Se verifica la integridad de los datos utilizando `hmac-sha1`.
- **Paquete 12 (New Keys - Cliente envía mensaje de claves de cifrado):**
 - **Algoritmos de cifrado para la sesión:**
 - Cifrado de sesión: `aes128-ctr`
 - Autenticación de mensaje: `hmac-sha1`
 - El cliente indica que ha generado las claves para el intercambio.

1.8.2. C2

- **Paquete 4**
 - **Versión de Protocolo:** SSH-2.0-OpenSSH.7.7p1 Ubuntu-1ubuntu0.3
- **Paquete 6**
 - **Versión de Protocolo:** SSH-2.0-OpenSSH.9.0p1 Ubuntu-1ubuntu7.3
- **Paquete 8 (Key Exchange Init - Cliente envía mensaje de inicio de intercambio de llaves):**
 - **Algoritmos negociados:**
 - Algoritmo de intercambio de claves: `diffie-hellman-group14-sha1`
 - Algoritmo de cifrado para la sesión: `aes128-ctr`

- Algoritmo de autenticación de mensaje: `hmac-sha1`
- **Paquete 9 (Key Exchange Init - Servidor responde al intercambio):**
 - **Algoritmos negociados:**
 - Algoritmo de intercambio de claves: `diffie-hellman-group14-sha1`
 - Algoritmo de cifrado para la sesión: `aes128-ctr`
 - Algoritmo de autenticación de mensaje: `hmac-sha1`
- **Paquete 10 (Key Exchange y Diffie-Hellman - Cliente envía parte de su clave Diffie-Hellman):**
 - **Claves Diffie-Hellman:**
 - Se comparte una llave pública de peso 32 bytes.
 - El cliente envía su clave pública como parte del intercambio Diffie-Hellman.
 - La verificación de la integridad de los datos se realiza utilizando el algoritmo `hmac-sha1`.
- **Paquete 11 (Key Exchange y Diffie-Hellman, New Keys - Servidor responde con su parte de la clave Diffie-Hellman, responde también con las nuevas claves):**
 - **Claves Diffie-Hellman:**
 - Se comparte una llave pública con un largo de 32 bytes.
 - El servidor envía su clave pública como parte del intercambio Diffie-Hellman.
 - Verificación de la integridad de los datos utilizando `hmac-sha1`.
 - Se envían las claves de sesión generadas como parte del reply.
- **Paquete 12 (New Keys - Cliente envía mensaje de claves de cifrado):**
 - **Claves de sesión:**
 - Cifrado de sesión: `aes128-ctr`
 - Autenticación de mensaje: `hmac-sha1`
 - **Otros datos del paquete:** El cliente confirma el intercambio de claves y está listo para iniciar una comunicación cifrada segura.

1.8.3. C3

- **Paquete 6**
 - **Versión de Protocolo:** SSH-2.0-OpenSSH.8.3p1 Ubuntu-1ubuntu0.1
- **Paquete 8**

- **Versión de Protocolo:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Paquete 10 (Key Exchange Init - Cliente envía mensaje de inicio de intercambio de llaves):**
 - **Algoritmos negociados:**
 - Algoritmo de intercambio de claves: `diffie-hellman-group14-sha1`
 - Algoritmo de cifrado para la sesión: `aes128-ctr`
 - Algoritmo de autenticación de mensaje: `hmac-sha1`
 - **Opciones adicionales:** El cliente indica soporte para compresión y otras extensiones.
- **Paquete 11 (Key Exchange Init - Servidor responde al intercambio):**
 - **Algoritmos negociados:**
 - Algoritmo de intercambio de claves: `diffie-hellman-group14-sha1`
 - Algoritmo de cifrado para la sesión: `aes128-ctr`
 - Algoritmo de autenticación de mensaje: `hmac-sha1`
 - Algoritmo de compresión: `none`
 - **Opciones adicionales:** El servidor también especifica soporte para opciones de compresión y autenticación.
- **Paquete 12 (Key Exchange y Diffie-Hellman, New Keys - Cliente envía su parte de la clave Diffie-Hellman):**
 - **Claves Diffie-Hellman:**
 - Llave pública de largo 32 bytes.
 - El cliente envía su clave pública como parte del intercambio Diffie-Hellman.
 - La verificación de la integridad de los datos se realiza utilizando el algoritmo `hmac-sha1`.
- **Paquete 13 (Key Exchange y Diffie-Hellman, New Keys - Servidor responde con su parte de la clave Diffie-Hellman, responde también con las nuevas claves):**
 - **Claves Diffie-Hellman:**
 - Llave pública compartida de tamaño 32 bytes.
 - El servidor envía su clave pública como parte del intercambio Diffie-Hellman.
 - Verificación de la integridad de los datos utilizando `hmac-sha1`.
 - Se envían las claves de sesión generadas como parte de la respuesta.
- **Paquete 14 (New Keys - Cliente envía mensaje de claves de cifrado:**

- **Claves de sesión:**
 - Cifrado de sesión: `aes128-ctr`
 - Autenticación de mensaje: `hmac-sha1`
- El cliente confirma el intercambio de claves..

1.8.4. C4/S1

- **Paquete 6**
 - **Versión de Protocolo:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Paquete 8**
 - **Versión de Protocolo:** SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3
- **Paquete 10 (Key Exchange Init - Cliente envía mensaje de inicio de intercambio de llaves):**
 - **Algoritmos negociados:**
 - Algoritmo de intercambio de claves: `diffie-hellman-group-exchange-sha256`
 - Algoritmo de cifrado para la sesión: `aes256-ctr`
 - Algoritmo de autenticación de mensaje: `hmac-sha2-256`
- **Paquete 11 (Key Exchange Init - Servidor responde al intercambio):**
 - **Algoritmos negociados:**
 - Algoritmo de intercambio de claves: `diffie-hellman-group-exchange-sha256`
 - Algoritmo de cifrado para la sesión: `aes256-ctr`
 - Algoritmo de autenticación de mensaje: `hmac-sha2-256`
- **Paquete 12 (Key Exchange y Diffie-Hellman, New Keys - Cliente envía su parte de la clave Diffie-Hellman):**
 - **Claves Diffie-Hellman:**
 - En este caso no es posible visibilizar el largo de la llave pública que se comparte, debido a algún error.
 - El cliente envía su clave pública como parte del intercambio Diffie-Hellman.
 - La verificación de la integridad de los datos se realiza utilizando el algoritmo `hmac-sha2-256`.
- **Paquete 13 (Key Exchange y Diffie-Hellman, New Keys - Servidor responde con su parte de la clave Diffie-Hellman, responde también con las nuevas claves):**
 - **Claves Diffie-Hellman:**

- Por la misma razón del cliente, tampoco se es posible ver la llave compartida.
 - El servidor envía su clave pública como parte del intercambio Diffie-Hellman.
 - Verificación de la integridad de los datos utilizando `hmac-sha2-256`.
 - Se envían las claves de sesión generadas como parte del reply generado.
- **Paquete 14 (New Keys - Cliente envía mensaje de claves de cifrado):**
- **Claves de sesión:**
 - Cifrado de sesión: `aes256-ctr`
 - Autenticación de mensaje: `hmac-sha2-256`
 - El cliente confirma el intercambio de claves.

1.9. Diferencia entre C1 y C2

Las principales diferencias entre los clientes C1 y C2 se encuentran en los algoritmos utilizados y en la versión de OpenSSH implementada.

En cuanto a la versión del protocolo, C1 utiliza `OpenSSH_7.3p1 Ubuntu-1ubuntu0.1`, mientras que C2 utiliza una versión más reciente, `OpenSSH_7.7p1 Ubuntu-1ubuntu0.3`. Lo que conlleva mejoras en el protocolo.

Con respecto a los algoritmos de cifrado y autenticación, C1 emplea `diffie-hellman-group14-sha1` para el intercambio de claves y `aes128-ctr` para el cifrado dentro de la sesión, mientras que C2 utiliza un intercambio de claves más seguro, `diffie-hellman-group-exchange-sha256`, junto con `aes256-ctr` para un cifrado más fuerte. Además, C2 usa el algoritmo `hmac-sha2-256` para la autenticación de mensajes, que es más seguro que `hmac-sha1` usado por C1.

Aunque ambos clientes siguen un proceso similar de intercambio de claves Diffie-Hellman y verificación de integridad de los datos, el Cliente C2 tiene una ventaja de seguridad debido a la elección de algoritmos más robustos. Esto mejora la protección de la comunicación cifrada en comparación con el Cliente C1.

1.10. Diferencia entre C2 y C3

Entre los clientes C2 y C3, se observan diferencias las mismas diferencias que en la subsección anterior.

El Cliente C2 emplea `OpenSSH_7.7p1 Ubuntu-1ubuntu0.3`, mientras que el Cliente C3 utiliza una versión más nueva de `OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`. Esto implica que el Cliente C2 podría beneficiarse de mejoras de seguridad y optimización presentes en la versión más reciente del software.

En términos de seguridad, los clientes C2 y C3 usan los mismo algoritmos, tanto para cifrado como para autenticación, por lo tanto no hay mejora en este aspecto.

Se indica que gracias a una versión mejorada de OpenSSH presente en C3, este puede presentar mejoras sustanciales.

1.11. Diferencia entre C3 y C4

Las diferencias entre los clientes C3 y C4 se centran principalmente en los algoritmos de cifrado y las versiones del protocolo SSH.

Primero, C3 utiliza `OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`, mientras que el cliente C4 emplea una versión más reciente `OpenSSH_9.0p1 Ubuntu-1ubuntu7.3`.

En cuanto a los algoritmos de intercambio de claves, ambos clientes utilizan como algoritmo `diffie-hellman-group14-sha1`, pero el C4 también indica soporte para el algoritmo `diffie-hellman-group-exchange-sha256`, lo que sugiere una mayor flexibilidad y un enfoque más seguro para la negociación de claves en comparación con el cliente C3.

Respecto al cifrado de la sesión, el Cliente C4 utiliza `aes256-ctr`, lo que proporciona un nivel de seguridad más alto en comparación con el `aes128-ctr` utilizado por C3. Además, en cuanto a la autenticación de mensajes, el cliente C4 emplea `hmac-sha2-256`, que es más seguro que el `hmac-sha1` utilizado por el cliente C3.

Estas diferencias reflejan que el cliente C4 es más seguro y moderno en comparación con C3, con versiones de software más recientes y algoritmos de seguridad más robustos.

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión “?”

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

3. Desarrollo (Parte 3)

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

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH es una implementación del protocolo de seguridad SSH, este proporciona herramientas para una autenticación segura, comunicación cifrada y un manejo de sistema remoto. Cuando un usuario se conecta a un sistema usando OpenSSH, su proceso comienza con la autenticación tanto con contraseña como con fingerprint. Una vez iniciada la conexión, se da paso a traspaso de información encriptada por parte de ambas entidades. El cliente es capaz de ejecutar comandos, transferir archivos o administrar el sistema remoto.

Las ventajas que esta implementación presenta incluyen una alta seguridad en temas de cifrado, autenticación, su facilidad para administrar y transferir archivos también son su componente fuerte. Termina siendo una pieza esencial para administración remota segura de sistemas, como también el traspaso de datos cifrados, es altamente usado en entornos de red y sistemas operativos de la actualidad.

4.2. Capas de Seguridad en OpenSSH

OpenSSH ofrece capas tanto de cifrado como de autenticación como de seguridad en cuanto a cifrado. La primera capa corresponde al cifrado, la cual protege los datos utilizando algoritmos como AES y 3DES, esto asegura que toda la comunicación entre cliente y servidor sea segura.

La segunda capa es de autenticación, la cual autoriza a los usuarios para acceder a los sistemas remotos. OpenSSH soporta métodos como contraseñas, llaves públicas o fingerprints. Con esto también tiene que ver el intercambio de llaves, la cual es la tercera capa, donde cliente y servidor intercambian llaves para asegurar un traspaso seguro de información. Utilizando algoritmos como Diffie-Hellman y ECDH para establecer las llaves.

Existen otras capas, referentes a la integridad de los mensajes, que asegura que los datos transferidos no hayan sido manipulados durante las transmisión. Esta implementación utiliza

algoritmos de hash criptográficos como HMAC-SHA256 para verificar la integridad de los mensajes.

4.3. Identificación de que protocolos no se cumplen

Conclusiones y comentarios

Para terminar, OpenSSH es una implementación robusta del protocolo SSH, diseñada para acceder remotamente a los sistemas. Al emplear múltiples capas de seguridad, incluyendo cifrado, autenticación, intercambio de claves, integridad de mensajes, OpenSSH asegura que todas las comunicaciones entre clientes y servidores estén protegidas contra escuchas, manipulación y acceso no autorizado.

Lo anterior mencionado se pudo ver en el análisis de tráfico generado entre los clientes C1 a C4 con el servidor S1, donde cada uno proporciona sus propios algoritmos de cifrado e intercambio de llaves, aunque siendo de diferentes versiones, el intercambio de igual manera se puede llevar a cabo, sin embargo al no estar actualizados algunos protocolos como en el caso de C1, puede que las implementaciones no sean tan seguras como si fuera C4.

De igual manera, OpenSSH es una herramienta esencial para administradores de sistemas y personas que deseen acceder a sistemas seguros, que también quieran realizar trasposos de información de manera segura. Al ser usado ampliamente en la gran mayoría de entornos, se recalca la fiabilidad y efectividad de esta herramienta.