

Tarea 3

Inyección y Modificación de Tráfico con Scapy

Taller de Redes y Servicios

Alexa Galaz Barahona

Sección 3

Profesor: Miguel Contreras

Ayudante: Laura Romero

2 de Julio de 2025

Índice

1. Capítulo I – Introducción al protocolo y software utilizado	2
Capítulo I – Introducción al protocolo y software utilizado	2
1.1. Introducción	2
1.2. Justificación	2
1.3. Descripción del protocolo MySQL	2
1.4. Software en el contenedor servidor	2
1.5. Software en el contenedor cliente	3
1.6. Conexión entre contenedores	3
2. Capítulo II – Análisis del tráfico e intervención con Scapy	4
Capítulo II – Análisis del tráfico e intervención con Scapy	4
2.1. Herramienta de análisis: Scapy	4
2.2. Interfaz de red Docker	4
2.3. Configuración y ejecución del script <code>mi_script_scapy.py</code>	5
2.4. Inyecciones de tráfico con fuzzing	6
2.5. Modificación de tráfico en tiempo real	7
2.6. Resumen final de captura de tráfico	9
3. Capítulo III – Análisis y resultados	10
Capítulo III – Análisis y resultados	10
3.1. Análisis de la inyección de tráfico (fuzzing)	10
3.2. Conclusiones del experimento	11
4. Conclusión general y lecciones aprendidas	12

1. Capítulo I – Introducción al protocolo y software utilizado

1.1. Introducción

En esta tarea se estudió el comportamiento del protocolo MySQL ante la inyección y modificación de tráfico no esperado, con el objetivo de observar cómo reacciona el servicio bajo condiciones anómalas o potencialmente maliciosas. A través del uso de contenedores Docker, se levantaron instancias separadas para el cliente y el servidor, permitiendo simular una arquitectura realista. Utilizando la herramienta Scapy dentro de un entorno virtual controlado, se desarrollaron scripts capaces de interceptar, alterar y enviar paquetes en tiempo real. Esta práctica permitió comprender de forma más profunda la seguridad a nivel de red y la tolerancia de servicios frente a posibles ataques o manipulaciones del protocolo.

1.2. Justificación

La presente actividad se fundamenta en la necesidad de comprender el impacto que puede tener el tráfico no esperado sobre un servicio de red en ejecución. Mediante la utilización de Scapy, se diseñaron scripts capaces de interceptar y modificar paquetes entre un cliente y un servidor MySQL desplegados en contenedores Docker. Esta simulación permitió reproducir escenarios de inyección de datos y manipulación de peticiones en tiempo real, facilitando el análisis de las posibles reacciones del software ante eventos anómalos. De este modo, se logró evaluar la robustez y comportamiento del protocolo frente a condiciones adversas, abordando aspectos clave de seguridad y funcionamiento a nivel de red.

1.3. Descripción del protocolo MySQL

MySQL es un sistema de gestión de bases de datos relacional (RDBMS) que opera sobre el protocolo de transporte TCP, utilizando por defecto el puerto 3306 para establecer conexiones entre cliente y servidor. La comunicación entre ambas partes se realiza mediante un protocolo propietario de MySQL, basado en un formato binario optimizado para el intercambio eficiente de comandos SQL y respuestas. Este protocolo permite operaciones como consultas, inserciones, actualizaciones y borrado de datos, así como el control de transacciones y autenticación de usuarios. Su estructura compacta y específica facilita el procesamiento rápido de grandes volúmenes de datos, siendo ampliamente utilizado en aplicaciones web y servicios que requieren rendimiento y escalabilidad.

1.4. Software en el contenedor servidor

El contenedor servidor fue desplegado utilizando la imagen `mysql-servidor-per`, la cual contiene una instancia funcional del sistema de gestión de bases de datos MySQL. Este contenedor expone el puerto 3306, permitiendo conexiones entrantes desde clientes que se encuentren en la misma red Docker definida. La configuración inicial incluye la creación de un usuario con permisos para realizar operaciones básicas de consulta y manipulación de datos, lo que facilita las pruebas de inyección y modificación de tráfico realizadas con Scapy. La base de datos utilizada incluye una tabla de prueba con datos ficticios, diseñada para evaluar las reacciones del servicio ante comandos manipulados.

1.5. Software en el contenedor cliente

Se utilizó la imagen `mysql-cliente-per`, la cual contiene una instalación del cliente MySQL. Desde este contenedor se ejecutaron comandos SQL dirigidos al servidor MySQL con el objetivo de generar tráfico de red controlado. Estas consultas permitieron analizar el comportamiento del protocolo, tanto en condiciones normales como al interceptar, modificar o inyectar paquetes utilizando Scapy.

1.6. Conexión entre contenedores

Se creó una red Docker llamada `mysql-net` y se ejecutaron ambos contenedores:

```
sudo docker network create mysql-net
sudo docker run -it --name mysql-servidor --network mysql-net mysql-servidor-per
sudo docker run -it --name mysql-cliente --network mysql-net mysql-cliente-per
```

Desde el cliente se realizó una conexión exitosa:

```
mysql -u root -p -h mysql-servidor
```

2. Capítulo II – Análisis del tráfico e intervención con Scapy

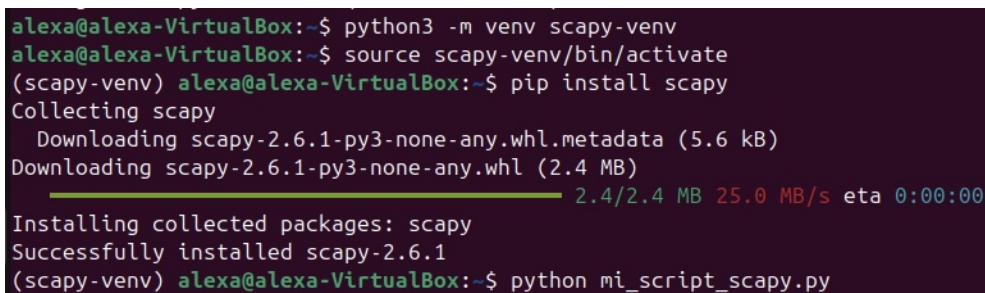
2.1. Herramienta de análisis: Scapy

Para realizar la intervención, modificación e inyección de tráfico en la red Docker, se utilizó Scapy, una herramienta de Python especializada en el análisis y manipulación de paquetes de red.

Scapy fue instalado dentro de un entorno virtual (`scapy-venv`) para aislar su ejecución del sistema base y facilitar la gestión de dependencias. A continuación, los comandos utilizados para su instalación:

```
python3 -m venv scapy-venv
source scapy-venv/bin/activate
pip install scapy
```

Una vez completada la instalación, se verificó su correcto funcionamiento.



```
alex@alex-VirtualBox:~$ python3 -m venv scapy-venv
alex@alex-VirtualBox:~$ source scapy-venv/bin/activate
(scapy-venv) alex@alex-VirtualBox:~$ pip install scapy
Collecting scapy
  Downloading scapy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
  Downloading scapy-2.6.1-py3-none-any.whl (2.4 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 MB 25.0 MB/s eta 0:00:00
Installing collected packages: scapy
Successfully installed scapy-2.6.1
(scapy-venv) alex@alex-VirtualBox:~$ python mi_script_scapy.py
```

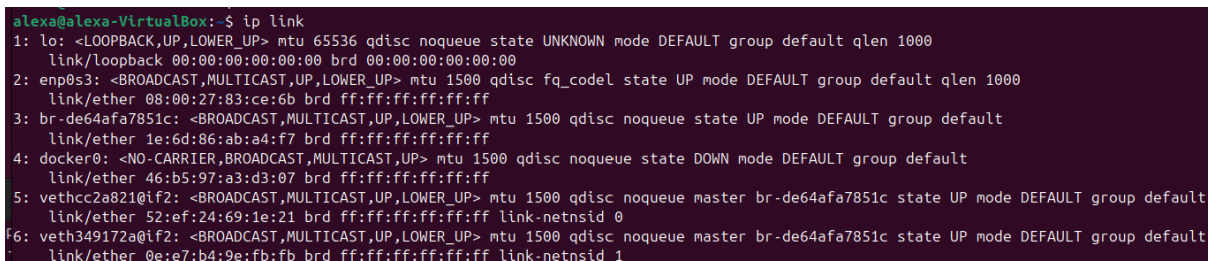
Figura 1: Instalación de Scapy en entorno virtual

2.2. Interfaz de red Docker

Para interceptar correctamente el tráfico entre los contenedores Docker, fue necesario identificar la interfaz de red utilizada por estos. Esto se logró mediante el comando:

```
ip link
```

Entre las interfaces listadas, se seleccionó `br-de64afa7851c`, ya que corresponde al bridge que conecta directamente al cliente y servidor desplegados en la red virtual `mysql-net`. Aunque existía otra interfaz llamada `docker0`, esta no estaba activa ni relacionada con el tráfico observado entre los contenedores, por lo que se descartó.

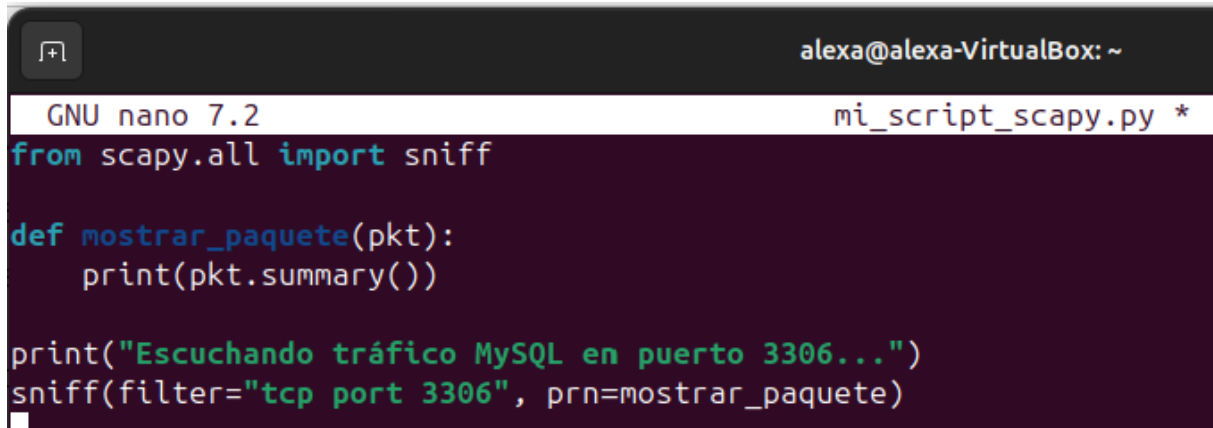


```
alex@alex-VirtualBox:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:83:ce:6b brd ff:ff:ff:ff:ff:ff
3: br-de64afa7851c: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 1e:6d:86:ab:a4:f7 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 46:b5:97:a3:d3:07 brd ff:ff:ff:ff:ff:ff
5: vethcc2a821@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-de64afa7851c state UP mode DEFAULT group default
    link/ether 52:ef:24:69:1e:21 brd ff:ff:ff:ff:ff:ff link-netnsid 0
6: veth349172a@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-de64afa7851c state UP mode DEFAULT group default
    link/ether 0e:e7:b4:9e:fb:fb brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

Figura 2: Listado de interfaces con `ip link`. Se utilizó `br-de64afa7851c` para capturar el tráfico.

2.3. Configuración y ejecución del script `mi_script_scapy.py`

Antes de iniciar la captura de tráfico, se diseñó el script `mi_script_scapy.py`, el cual hace uso de la función `sniff()` de Scapy para interceptar paquetes que circulan en la red Docker. Este script fue configurado para capturar únicamente tráfico TCP en el puerto 3306, utilizado por el servicio MySQL.



```
alexa@alexa-VirtualBox: ~
GNU nano 7.2 mi_script_scapy.py *
from scapy.all import sniff

def mostrar_paquete(pkt):
    print(pkt.summary())

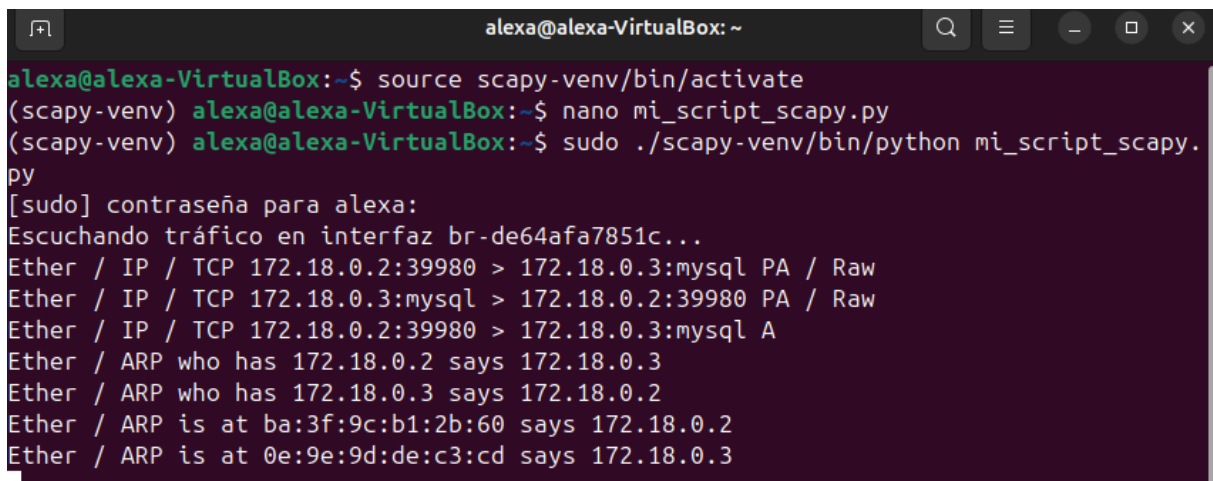
print("Escuchando tráfico MySQL en puerto 3306...")
sniff(filter="tcp port 3306", prn=mostrar_paquete)
```

Figura 3: Código fuente del script `mi_script_scapy.py`

El fragmento clave que permite la captura es el siguiente:

```
sniff(filter="tcp port 3306", prn=mostrar_paquete)
```

A continuación, se muestra la salida del script `mi_script_scapy.py` al ejecutarse exitosamente. En ella se puede observar la captura en tiempo real del tráfico entre el cliente y el servidor MySQL, incluyendo paquetes TCP y solicitudes ARP intercambiados en la red Docker.



```
alexa@alexa-VirtualBox: ~
alexa@alexa-VirtualBox:~$ source scapy-venv/bin/activate
(scapy-venv) alexa@alexa-VirtualBox:~$ nano mi_script_scapy.py
(scapy-venv) alexa@alexa-VirtualBox:~$ sudo ./scapy-venv/bin/python mi_script_scapy.py
[sudo] contraseña para alexa:
Escuchando tráfico en interfaz br-de64afa7851c...
Ether / IP / TCP 172.18.0.2:39980 > 172.18.0.3:mysql PA / Raw
Ether / IP / TCP 172.18.0.3:mysql > 172.18.0.2:39980 PA / Raw
Ether / IP / TCP 172.18.0.2:39980 > 172.18.0.3:mysql A
Ether / ARP who has 172.18.0.2 says 172.18.0.3
Ether / ARP who has 172.18.0.3 says 172.18.0.2
Ether / ARP is at ba:3f:9c:b1:2b:60 says 172.18.0.2
Ether / ARP is at 0e:9e:9d:de:c3:cd says 172.18.0.3
```

Figura 4: Ejecución del script `mi_script_scapy.py` y captura de tráfico MySQL

2.4. Inyecciones de tráfico con fuzzing

Con el fin de evaluar cómo responde el servicio MySQL frente a paquetes anómalos, se utilizó el script `inyeccion_fuzz.py`, diseñado para realizar dos inyecciones distintas:

- **Primer intento:** Se envió un paquete TCP con una carga malformada fija ("`@@@FAKE_SQL_COMMAND@@@`"), simulando una instrucción SQL inválida.
- **Segundo intento:** Se generó una carga binaria aleatoria de 40 bytes mediante `os.urandom()` para introducir datos inesperados.

Ambos paquetes fueron contruidos usando Scapy y enviados al puerto 3306 del servidor.

A screenshot of a terminal window titled 'alexa@alexa-VirtualBox: ~'. The window shows the code of a Python script named 'inyeccion_fuzz.py' being edited in 'GNU nano 7.2'. The code imports 'IP', 'TCP', 'Raw', and 'send' from 'scapy.all', and 'random' and 'os'. It sets 'ip_destino' to '172.18.0.2' and 'puerto_destino' to '3306'. It prints 'Enviando paquetes de fuzzing...'. It then creates 'paquete1' with a TCP payload of '@@>' and sends it. Next, it generates 40 random bytes with 'os.urandom(40)', creates 'paquete2' with that data as the TCP payload, and sends it. Finally, it prints 'Paquetes enviados.'.

```
GNU nano 7.2 inyeccion_fuzz.py *
from scapy.all import IP, TCP, Raw, send
import random
import os

ip_destino = "172.18.0.2"
puerto_destino = 3306

print("Enviando paquetes de fuzzing...")

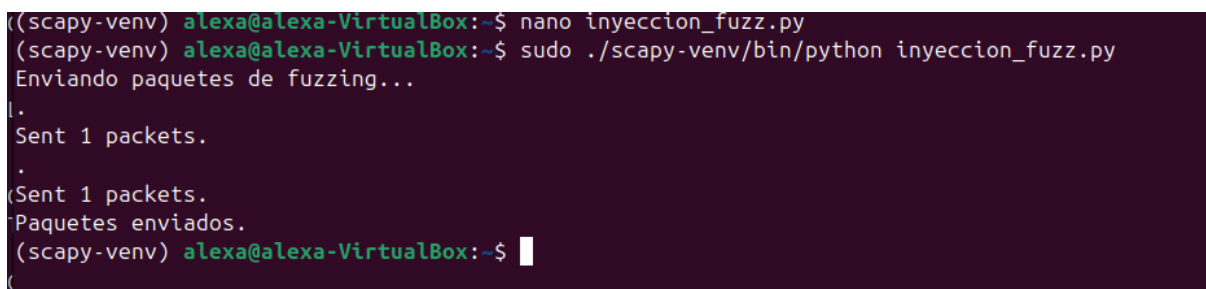
paquete1 = IP(dst=ip_destino)/TCP(dport=puerto_destino, sport=random.randint(1024,65535))/Raw(load="@>")
send(paquete1)

data_random = os.urandom(40)
paquete2 = IP(dst=ip_destino)/TCP(dport=puerto_destino, sport=random.randint(1024,65535))/Raw(load=data_random)
send(paquete2)

print("Paquetes enviados.")
```

Figura 5: Código del script `inyeccion_fuzz.py` con dos tipos de cargas malformadas

La ejecución del script evidenció que ambos paquetes fueron transmitidos correctamente al servidor, como se muestra en la siguiente terminal:

A screenshot of a terminal window showing the execution of the script. The user enters 'nano inyeccion_fuzz.py' and then 'sudo ./scapy-venv/bin/python inyeccion_fuzz.py'. The script outputs 'Enviando paquetes de fuzzing...', 'Sent 1 packets.', and 'Paquetes enviados.'.

```
((scapy-venv) alexa@alexa-VirtualBox:~$ nano inyeccion_fuzz.py
((scapy-venv) alexa@alexa-VirtualBox:~$ sudo ./scapy-venv/bin/python inyeccion_fuzz.py
Enviando paquetes de fuzzing...
.
Sent 1 packets.
.
Sent 1 packets.
Paquetes enviados.
((scapy-venv) alexa@alexa-VirtualBox:~$
```

Figura 6: Ejecución del script y confirmación del envío de paquetes de fuzzing

2.5. Modificación de tráfico en tiempo real

Con el objetivo de alterar el comportamiento del protocolo MySQL, se implementaron tres intentos de modificación del contenido de los paquetes interceptados. Estas pruebas se realizaron mediante el script `modificar_trafico.py`, utilizando la biblioteca `Scapy`.

Modificación 1: Alteración de valores en UPDATE

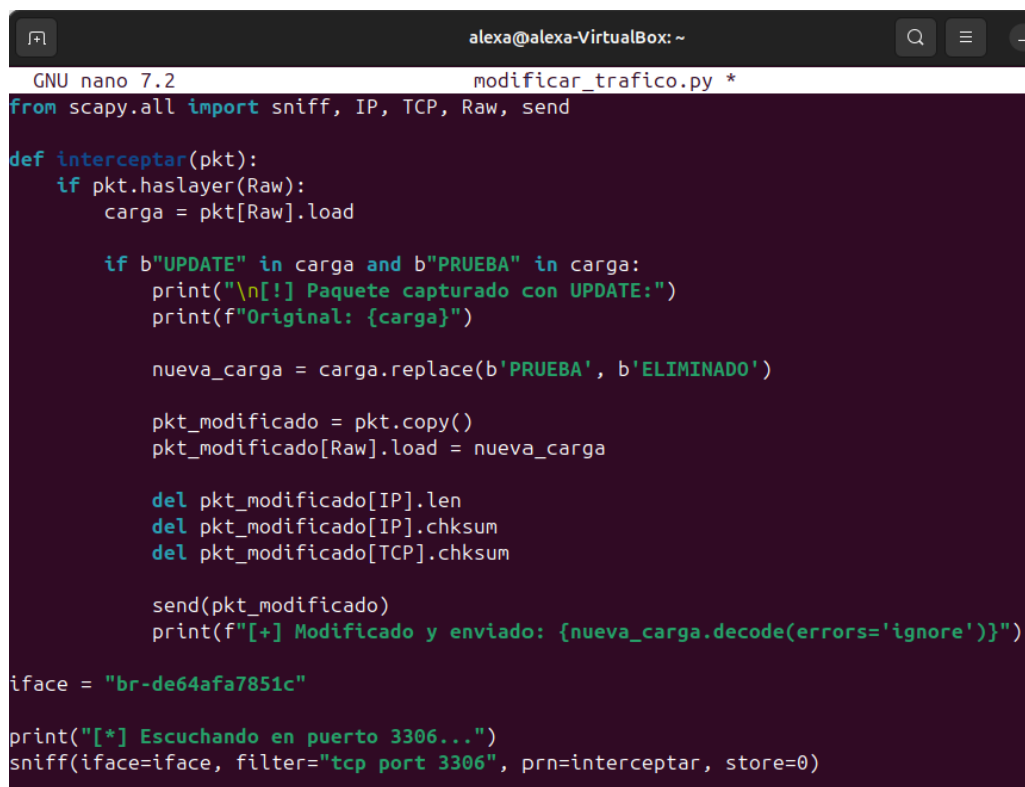
Esta modificación fue parcialmente exitosa. Se interceptaron paquetes con instrucciones `UPDATE` y se reemplazó el valor asignado en el campo `nombre`. Por ejemplo, la consulta original `UPDATE clientes SET nombre = PRUEBA WHERE id = 1` fue interceptada y transformada en `ELIMINADO`. Si bien el paquete fue capturado, modificado y reenviado correctamente (como se verificó en la consola), el cambio no se reflejó en la base de datos, posiblemente por cómo MySQL valida la conexión y la integridad del paquete.

Modificación 2: Inyección hacia tabla inexistente

En esta prueba se interceptó el mismo tipo de instrucción, pero se alteró la consulta para apuntar a una tabla inexistente, como `UPDATE usuarios SET nombre = "x"` con el fin de provocar un error controlado. El servidor respondió con un error indicando que la tabla no existía, lo que demuestra que la modificación fue procesada por el motor MySQL, aunque no ejecutada exitosamente.

Modificación 3: Instrucción inválida

Se intentó reemplazar parte del contenido con una instrucción no válida o malformada (por ejemplo, fragmentos incompletos o aleatorios). Aunque el paquete fue reenviado, el servidor no realizó ninguna acción visible, probablemente debido al rechazo silencioso de paquetes con sintaxis inválida.



```
alex@alexa-VirtualBox: ~
GNU nano 7.2                                modificar_trafico.py *
from scapy.all import sniff, IP, TCP, Raw, send

def interceptar(pkt):
    if pkt.haslayer(Raw):
        carga = pkt[Raw].load

        if b"UPDATE" in carga and b"PRUEBA" in carga:
            print("\n[!] Paquete capturado con UPDATE:")
            print(f"Original: {carga}")

            nueva_carga = carga.replace(b'PRUEBA', b'ELIMINADO')

            pkt_modificado = pkt.copy()
            pkt_modificado[Raw].load = nueva_carga

            del pkt_modificado[IP].len
            del pkt_modificado[IP].chksum
            del pkt_modificado[TCP].chksum

            send(pkt_modificado)
            print(f"[+] Modificado y enviado: {nueva_carga.decode(errors='ignore')}")

iface = "br-de64afa7851c"

print("[*] Escuchando en puerto 3306...")
sniff(iface=iface, filter="tcp port 3306", prn=interceptar, store=0)
```

Figura 7: Código del script `modificar_trafico.py` para interceptar y modificar paquetes

Durante la ejecución del script, se verificó en consola que los paquetes eran efectivamente interceptados, modificados y reenviados. En el caso de la modificación #1, se observó el reemplazo de la cadena dentro de la instrucción SQL, aunque la base de datos no aplicó el cambio, posiblemente por restricciones del motor o el control de sesiones.

```
(scapy-venv) alexa@alexa-VirtualBox:~$ sudo ./scapy-venv/bin/python modificar_trafico.py
[sudo] contraseña para alexa:
[*] Escuchando en puerto 3306...

[!] Paquete capturado con UPDATE:
Original: b'3\x00\x00\x00\x03UPDATE clientes SET nombre = "PRUEBA" WHERE id = 1'
WARNING: MAC address to reach destination not found. Using broadcast.
.
Sent 1 packets.
[+] Modificado y enviado: 3UPDATE clientes SET nombre = "ELIMINADO" WHERE id = 1
```

Figura 8: Ejecución del script con modificación activa del tráfico SQL

Fundamentación de las modificaciones

A continuación, se detallan las tres modificaciones implementadas al tráfico MySQL, indicando el resultado observado y la justificación teórica para cada una:

- **Modificación 1: Alteración del valor en UPDATE**

Fundamento: Al interceptar un paquete que contenía una instrucción UPDATE y modificar el valor de la asignación (SET nombre = "PRUEBA" → ELIMINADO), se esperaba que el motor MySQL ejecutara la instrucción modificada, alterando el contenido de la base de datos. Esto simula cómo un atacante podría modificar datos de forma silenciosa en tránsito.

- **Modificación 2: Inyección hacia tabla inexistente**

Fundamento: Modificar la consulta para apuntar a una tabla inexistente permite comprobar que el paquete fue procesado por el servidor. El error recibido (Table 'usuarios' doesn't exist) evidencia que la modificación fue reconocida, aunque no se completó con éxito.

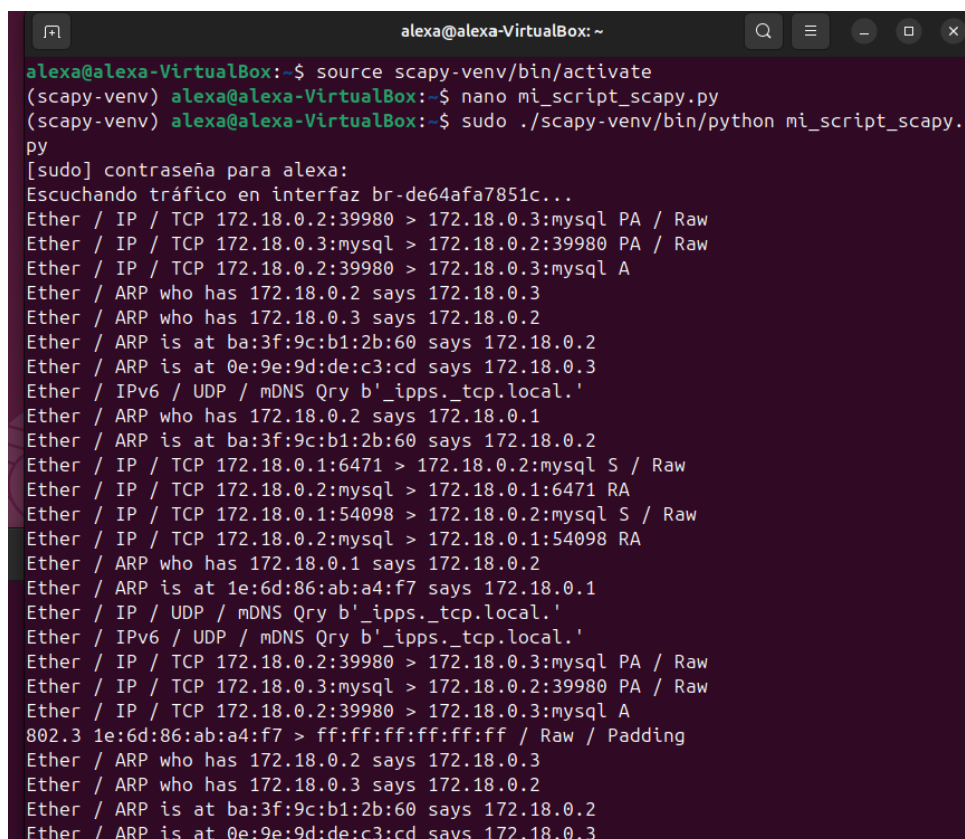
- **Modificación 3: Reemplazo por instrucción inválida**

Fundamento: Al introducir una consulta malformada, se simula un escenario de corrupción de tráfico o ataque con código inválido. Se esperaba que el servidor la rechazara, lo que fue confirmado al no obtenerse ninguna respuesta visible. Esto prueba cómo el protocolo maneja errores de sintaxis a nivel de red.

2.6. Resumen final de captura de tráfico

Al realizar los scripts de captura, modificación e inyección de paquetes con Scapy, se ejecutó el script `mi_script_scapy.py` para verificar el comportamiento del tráfico en la red virtual configurada.

Se logró capturar en tiempo real múltiples tipos de paquetes entre los contenedores, incluyendo tráfico TCP de MySQL, consultas ARP y mensajes mDNS. Esto confirma la efectividad del entorno Docker junto a la herramienta Scapy para analizar y manipular el tráfico con éxito.



```
alex@alexa-VirtualBox: ~  
alex@alexa-VirtualBox:~$ source scapy-venv/bin/activate  
(scapy-venv) alexa@alexa-VirtualBox:~$ nano mi_script_scapy.py  
(scapy-venv) alexa@alexa-VirtualBox:~$ sudo ./scapy-venv/bin/python mi_script_scapy.py  
[sudo] contraseña para alexa:  
Escuchando tráfico en interfaz br-de64afa7851c...  
Ether / IP / TCP 172.18.0.2:39980 > 172.18.0.3:mysql PA / Raw  
Ether / IP / TCP 172.18.0.3:mysql > 172.18.0.2:39980 PA / Raw  
Ether / IP / TCP 172.18.0.2:39980 > 172.18.0.3:mysql A  
Ether / ARP who has 172.18.0.2 says 172.18.0.3  
Ether / ARP who has 172.18.0.3 says 172.18.0.2  
Ether / ARP is at ba:3f:9c:b1:2b:60 says 172.18.0.2  
Ether / ARP is at 0e:9e:9d:de:c3:cd says 172.18.0.3  
Ether / IPv6 / UDP / mDNS Qry b'_ipps._tcp.local.'  
Ether / ARP who has 172.18.0.2 says 172.18.0.1  
Ether / ARP is at ba:3f:9c:b1:2b:60 says 172.18.0.2  
Ether / IP / TCP 172.18.0.1:6471 > 172.18.0.2:mysql S / Raw  
Ether / IP / TCP 172.18.0.2:mysql > 172.18.0.1:6471 RA  
Ether / IP / TCP 172.18.0.1:54098 > 172.18.0.2:mysql S / Raw  
Ether / IP / TCP 172.18.0.2:mysql > 172.18.0.1:54098 RA  
Ether / ARP who has 172.18.0.1 says 172.18.0.2  
Ether / ARP is at 1e:6d:86:ab:a4:f7 says 172.18.0.1  
Ether / IP / UDP / mDNS Qry b'_ipps._tcp.local.'  
Ether / IPv6 / UDP / mDNS Qry b'_ipps._tcp.local.'  
Ether / IP / TCP 172.18.0.2:39980 > 172.18.0.3:mysql PA / Raw  
Ether / IP / TCP 172.18.0.3:mysql > 172.18.0.2:39980 PA / Raw  
Ether / IP / TCP 172.18.0.2:39980 > 172.18.0.3:mysql A  
802.3 1e:6d:86:ab:a4:f7 > ff:ff:ff:ff:ff:ff / Raw / Padding  
Ether / ARP who has 172.18.0.2 says 172.18.0.3  
Ether / ARP who has 172.18.0.3 says 172.18.0.2  
Ether / ARP is at ba:3f:9c:b1:2b:60 says 172.18.0.2  
Ether / ARP is at 0e:9e:9d:de:c3:cd says 172.18.0.3
```

Figura 9: Captura final del tráfico de red tras la ejecución de todos los scripts

3. Capítulo III – Análisis y resultados

3.1. Análisis de la inyección de tráfico (fuzzing)

Con el objetivo de observar el comportamiento del protocolo MySQL ante la recepción de datos malformados, se implementó el script `inyeccion_fuzz.py`. Este script genera y envía paquetes TCP dirigidos al puerto 3306 del servidor, pero con cargas no válidas que no siguen la estructura esperada por el servicio.

```
IP(dst=ip destino)/TCP(dport=puerto)/Raw(load="@@@FAKE SQL COMMAND@@@")
```

Durante la ejecución, se envió paquetes con contenido anómalo como parte de una técnica básica de fuzzing. Estos paquetes no fueron procesados por el servidor MySQL, lo cual es un comportamiento esperado, ya que MySQL está diseñado para ignorar o cerrar conexiones con datos que no respeten su protocolo binario.

Aunque no se observó una interrupción directa del servicio, esta prueba permitió validar que el servidor mantiene su estabilidad frente a entradas malformadas, lo cual demuestra cierto nivel de robustez ante este tipo de ataques simples.

Modificación en tiempo real

Para esta prueba, se diseñó un nuevo script llamado `modificar_trafico.py`, el cual intercepta en tiempo real los paquetes que contienen comandos `UPDATE` y reemplaza dinámicamente el valor asignado al campo `nombre`. Esta modificación se realiza usando la función `sniff()` de la biblioteca Scapy, lo que permite capturar e intervenir el tráfico antes de que sea procesado por el servidor MySQL.

```
if b'UPDATE' in carga:
    nueva_carga = carga.replace(b'PRUEBA', b'ELIMINADO')
    nuevo_pkt[Raw].load = nueva_carga
    send(nuevo_pkt)
```

El objetivo de esta modificación fue alterar de forma silenciosa el contenido de una operación legítima enviada por el cliente, modificando el valor a escribir en la base de datos. Si el paquete modificado llega correctamente al servidor y es ejecutado, se cambiaría el valor del campo `nombre` a `ELIMINADO`, sin que el cliente perciba el cambio.

La Figura 8 muestra cómo se intercepta un paquete que contenía la cadena "PRUEBA", se modifica su contenido y luego se envía el nuevo paquete con la palabra `ELIMINADO`. Esta técnica permite realizar ataques tipo *Man-in-the-Middle* (MitM), demostrando la capacidad de Scapy para modificar comandos SQL en tránsito.

Después de enviar paquetes modificados con `modificar_trafico.py`, se monitoreó el tráfico del servidor y el cliente. En la consola se visualizó que el paquete fue capturado, modificado y reenviado correctamente. Scapy notificó que no encontró una dirección MAC de destino válida y procedió a enviar el paquete por broadcast.

Aunque no se observó un efecto visible en la base de datos (el cambio no se aplicó), se validó que la manipulación del paquete ocurrió exitosamente a nivel de red. Esta limitación puede deberse a validaciones internas del motor MySQL, como la autenticación de sesión o la verificación del origen del paquete.

3.2. Conclusiones del experimento

La experimentación con Scapy permitió comprobar que es posible interceptar, modificar e inyectar tráfico en una red Docker simulada entre cliente y servidor MySQL.

Se lograron los siguientes resultados:

- Se interceptó exitosamente el tráfico en el puerto 3306 utilizando la interfaz **bridge** de Docker.
- Se diseñaron scripts capaces de modificar el contenido de paquetes **UPDATE** en tiempo real, reemplazando valores como "PRUEBA" por **ELIMINADO**.
- Se confirmaron las capturas y modificaciones mediante la salida de Scapy y **tcpdump**, observando que los paquetes eran efectivamente interceptados, alterados y reenviados.

Si bien no se observó una modificación visible en la base de datos, el experimento demostró que la manipulación del paquete fue exitosa a nivel de red. Esto indica que el servidor MySQL podría estar rechazando el paquete modificado por razones como autenticación de sesión, integridad del paquete o violaciones al protocolo binario.

Estas pruebas evidencian vulnerabilidades en protocolos que no utilizan cifrado, especialmente cuando no se implementan medidas de seguridad adicionales. Se refuerza la importancia de utilizar mecanismos como TLS/SSL, validación estricta de comandos y monitoreo de tráfico para prevenir ataques de tipo *Man-in-the-Middle*.

Hipótesis ante posibles fallos: en los casos en que el servidor no aplicó la modificación enviada, se presume que el motor MySQL descartó los paquetes por no provenir de una conexión autenticada, por no cumplir el formato binario completo del protocolo, o por contener alteraciones detectadas como anómalas.

4. Conclusión general y lecciones aprendidas

El desarrollo de esta tarea permitió profundizar en el entendimiento del tráfico de red entre aplicaciones cliente-servidor, así como en las implicancias de su manipulación. El uso de Scapy facilitó la interceptación, modificación e inyección de paquetes dentro de un entorno controlado, revelando aspectos clave sobre la seguridad en redes.

A lo largo del trabajo se logró:

- Capturar tráfico específico del protocolo MySQL en una red Docker virtualizada.
- Modificar comandos SQL legítimos en tiempo real, como `UPDATE`, alterando valores dentro de la carga útil del paquete.
- Observar el envío exitoso de paquetes modificados desde el punto de vista de red, a pesar de que el servidor no procesó las modificaciones.

Este ejercicio evidenció que servicios que no cuentan con medidas de cifrado ni autenticación pueden ser vulnerables a ataques de tipo *Man-in-the-Middle*, aunque las protecciones internas del protocolo también juegan un rol relevante. Asimismo, se destacó la utilidad de herramientas como Scapy para simular escenarios de prueba y estudiar el comportamiento de aplicaciones frente a paquetes manipulados.

Como aprendizaje, se concluye que incluso redes locales o ambientes controlados pueden presentar vectores de ataque si no se implementan medidas de seguridad adecuadas. La experimentación también fortaleció habilidades en el uso de contenedores, redes virtuales y scripting de bajo nivel para pruebas de seguridad.