Facultad de Ingeniería y Ciencias

Escuela de Informática y Telecomunicaciones

Taller de Redes y Servicios

Tarea 3 – Scapy

Nombres: José Avello, Álvaro Valdebenito
Profesor: Miguel Contreras
Ayudante: Laura Romero
Sección: 03
Fecha: 06/07/2025

Junio de 2025

Índice general

Capítulo 1

Introducción al protocolo

1.1. Introducción

1.1.1. Descripción del protocolo

El Protocolo FTP (File Transfer Protocol) es un protocolo de red utilizado para transferir archivos entre un cliente y un servidor a través de una red basada en TCP/IP. Su objetivo principal es permitir la carga, descarga y gestión remota de archivos mediante comandos específicos, utilizando dos canales: uno de control y otro de datos. FTP se emplea comúnmente en la administración de sitios web, distribución de software y sistemas de intercambio de archivos, especialmente cuando se requiere manejar grandes volúmenes de información. Aunque su funcionamiento es eficiente, no incluye mecanismos de cifrado, por lo que es vulnerable a interceptaciones; debido a esto, en entornos que requieren mayor seguridad se suelen usar variantes como FTPS o SFTP.

1.1.2. Importancia y relevancia del protocolo

El protocolo FTP ha sido, durante décadas, una herramienta estándar para la transferencia de archivos entre equipos remotos. Su relevancia radica en que permite enviar, recibir y gestionar archivos a través de redes basadas en TCP/IP, sin requerir acceso físico a los servidores. Gracias a esto, tareas como subir, descargar o renombrar archivos se vuelven más simples y accesibles para usuarios y administradores.

FTP se ha usado ampliamente en el desarrollo y mantenimiento de sitios web, la distribución de software y la automatización de respaldos. Aunque en la actualidad existen alternativas más seguras como SFTP o FTPS, FTP sigue teniendo valor en entornos internos o controlados, donde prima la simplicidad y la compatibilidad por sobre la seguridad. Su facilidad de uso y el hecho de que funciona en múltiples plataformas hacen que aún hoy sea considerado una pieza importante en muchas infraestructuras de red.

1.2. Historia del protocolo

1.2.1. Creación y desarrollo inicial

El protocolo FTP fue creado en 1971 por un grupo de investigadores del MIT (Massachusetts Institute of Technology), en colaboración con instituciones como la Universidad de Stanford y BBN Technologies, dentro del marco del desarrollo de ARPANET,

la red precursora de Internet. Este fue diseñado para satisfacer la necesidad de transferir archivos de manera eficiente y confiable entre diferentes sistemas conectados a una red, que en ese entonces era un reto debido a la diversidad de hardware y sistemas operativos. La especificación inicial de FTP fue publicada como parte de los primeros documentos RFC (Request for Comments), específicamente el RFC 114 en abril de 1971, y posteriormente se fue perfeccionando con el RFC 765 en 1980, que estableció un estándar más formal y detallado. El desarrollo de FTP respondió a la necesidad de un protocolo universal para compartir información entre usuarios remotos, facilitando la colaboración y el intercambio de datos en un entorno aún muy limitado tecnológicamente. Desde entonces, FTP se convirtió en uno de los pilares de las comunicaciones en redes informáticas y en la base para protocolos posteriores más seguros y especializados.

1.2.2. Evolución a lo largo del tiempo

A lo largo del tiempo, el protocolo FTP ha experimentado varias etapas de evolución y mejoras para adaptarse a las necesidades cambiantes de las redes y la seguridad. Después de su especificación inicial en los años 70, FTP fue formalizado y estandarizado en el RFC 765 en 1980, donde se definieron con mayor precisión sus comandos y modos de operación. Con la expansión de Internet en los años 90, surgió la necesidad de mejorar la seguridad, ya que FTP transmite datos, incluyendo contraseñas, en texto claro, lo que lo hace vulnerable a ataques. Esto llevó al desarrollo de extensiones seguras como FTPS, que incorpora cifrado mediante SSL/TLS para proteger la transferencia de archivos. A su vez, tambien apareció SFTP, que aunque es un protocolo diferente basado en SSH, se utiliza comúnmente como alternativa segura a FTP. Además, se mejoraron los modos de conexión para superar problemas con firewalls y NAT, introduciendo el modo pasivo, que facilitó su uso en entornos modernos. Aunque FTP no ha cambiado radicalmente en su estructura básica, estas adaptaciones y extensiones han permitido que siga siendo útil en entornos actuales, especialmente cuando se combinan con medidas de seguridad y configuraciones adecuadas.

1.3. Funcionamiento del protocolo

1.3.1. Descripción del funcionamiento

Este protocolo funciona bajo un modelo cliente-servidor, donde un cliente FTP se conecta a un servidor FTP para intercambiar archivos y comandos. Este modelo implica que el cliente inicia la conexión y solicita servicios al servidor, que es el encargado de gestionar los archivos y responder a las solicitudes. FTP utiliza dos canales de comunicación distintos: un canal de control y un canal de datos. El canal de control se establece primero, generalmente en el puerto 21, y se utiliza para enviar comandos y respuestas entre el cliente y el servidor. El canal de datos, que puede abrirse en diferentes puertos dependiendo del modo de conexión (activo o pasivo), se utiliza exclusivamente para la transferencia de archivos o listados de directorios. En el modo activo, el servidor inicia la conexión del canal de datos hacia el cliente, mientras que en el modo pasivo es el cliente quien se conecta a un puerto abierto por el servidor, facilitando la comunicación a través de firewalls y NAT. Gracias a esta separación de canales, FTP puede gestionar eficientemente el control y la transferencia de datos, manteniendo la sesión y permitiendo operaciones como subir, descargar y administrar archivos remotos.

1.3.2. Detalles técnicos del protocolo

Este opera principalmente en la capa de aplicación del modelo OSI y TCP/IP, utilizando TCP como protocolo de transporte para garantizar una comunicación confiable. Para establecer conexiones, FTP utiliza dos puertos estándar: el puerto 21 para el canal de control, donde se intercambian comandos y respuestas, y el puerto 20 para el canal de datos en el modo activo, aunque en modo pasivo el canal de datos puede usar puertos dinámicos asignados por el servidor. Los comandos FTP son textos ASCII que el cliente envía al servidor para realizar diversas operaciones, entre los más comunes están USER (para enviar el nombre de usuario), PASS (para enviar la contraseña), LIST (para listar archivos en un directorio), RETR (para descargar archivos), STOR (para subir archivos) y DELE (para eliminar archivos). El servidor responde con códigos numéricos que indican el estado de la operación, similares a los códigos de estado HTTP. FTP también soporta comandos para cambiar directorios (CWD), crear directorios (MKD), renombrar (RNFR y RNTO) y cerrar sesión (QUIT). Debido a su funcionamiento sobre TCP, FTP asegura la integridad y orden de los datos transferidos, pero su diseño original no contempla cifrado ni autenticación segura, lo que motivó el desarrollo de extensiones para agregar seguridad.

1.4. Aplicaciones y clientes de

1.4.1. Descripción de las aplicaciones, clientes y servidores conocidos

El protocolo FTP cuenta con una amplia variedad de aplicaciones y herramientas que lo utilizan tanto en clientes como en servidores, disponibles para múltiples sistemas operativos y entornos. Entre los clientes FTP más populares destacan FileZilla, WinSCP, Cyberduck y Transmit, que ofrecen interfaces gráficas intuitivas para facilitar la transferencia y gestión de archivos entre usuarios y servidores. Además, muchos sistemas operativos como Windows, macOS y Linux incluyen clientes FTP integrados o accesibles mediante línea de comandos, permitiendo su uso sin necesidad de software adicional. En el ámbito de los servidores FTP, existen soluciones reconocidas como vsftpd (Very Secure FTP Daemon) en Linux, ProFTPD, Pure-FTPd y el servidor FTP incluido en Windows Server. Estas plataformas permiten configurar y administrar servicios FTP para diferentes necesidades, desde servidores públicos para distribución de archivos hasta entornos corporativos con acceso controlado. FTP también se integra en aplicaciones empresariales, sistemas de gestión de contenido y plataformas de desarrollo web para facilitar la transferencia de datos y archivos en entornos tanto locales como en la nube. Su compatibilidad con distintos sistemas y su soporte extendido garantizan su uso continuo en diversos sectores.

1.5. Actualidad del protocolo

1.5.1. Uso actual del protocolo

A pesar de que han surgido protocolos más seguros y modernos, el protocolo FTP sigue utilizándose actualmente en muchos ámbitos debido a su simplicidad, eficiencia y amplia compatibilidad. Aunque su uso ha disminuido en entornos donde la seguridad es prioritaria, FTP continúa siendo frecuente en redes internas, sistemas de respaldo, transferencia de archivos en entornos controlados y en algunos servicios de hosting web para la gestión de archivos. Además, es común en organizaciones que requieren una solución rápida y estable para intercambiar grandes volúmenes de datos sin la necesidad de configuraciones complejas. Sin embargo, en aplicaciones públicas o sensibles, es habitual reemplazar FTP por alternativas como FTPS o SFTP, que ofrecen cifrado y autenticación más robustos. En resumen, FTP mantiene un rol importante en escenarios específicos, especialmente donde la seguridad no es la principal preocupación o se complementa con otras medidas, y sigue siendo una herramienta útil y ampliamente soportada en infraestructura tecnológica actual.

1.5.2. Comunidades

El protocolo FTP, al ser un estándar ampliamente utilizado desde hace décadas, es objeto de discusión, mantenimiento y desarrollo principalmente en comunidades técnicas y organizaciones dedicadas a los estándares de Internet. La IETF (Internet Engineering Task Force) es la principal entidad responsable de la estandarización y actualización de FTP, a través de la publicación y revisión de documentos RFC que definen su funcionamiento y mejoras. En cuanto a comunidades más prácticas, existen foros especializados en redes y administración de sistemas, como Stack Overflow, Reddit (r/networking, r/sysadmin) y foros de desarrolladores donde se discuten configuraciones, problemas y mejores prácticas relacionadas con FTP y sus variantes seguras (FTPS, SFTP). También, proyectos de software libre relacionados con servidores y clientes FTP, como FileZilla o vsftpd, mantienen comunidades activas en GitHub y otros repositorios donde desarrolladores colaboran en el mantenimiento y evolución de estas herramientas. Por último, existen grupos y listas de correo específicos dentro del ámbito de administradores de sistemas y redes que intercambian información técnica y novedades sobre FTP y protocolos de transferencia en general.

1.6. Instalación del software

1.6.1. Servidor

Para esta tarea se utilizó ProFTPD, un servidor FTP. Se creó una imagen Docker personalizada basada en Ubuntu 20.04, compilando ProFTPD desde el repositorio oficial en GitHub, entregado como referencia para esta actividad. A continuacion podemos ver el Dockerfile:

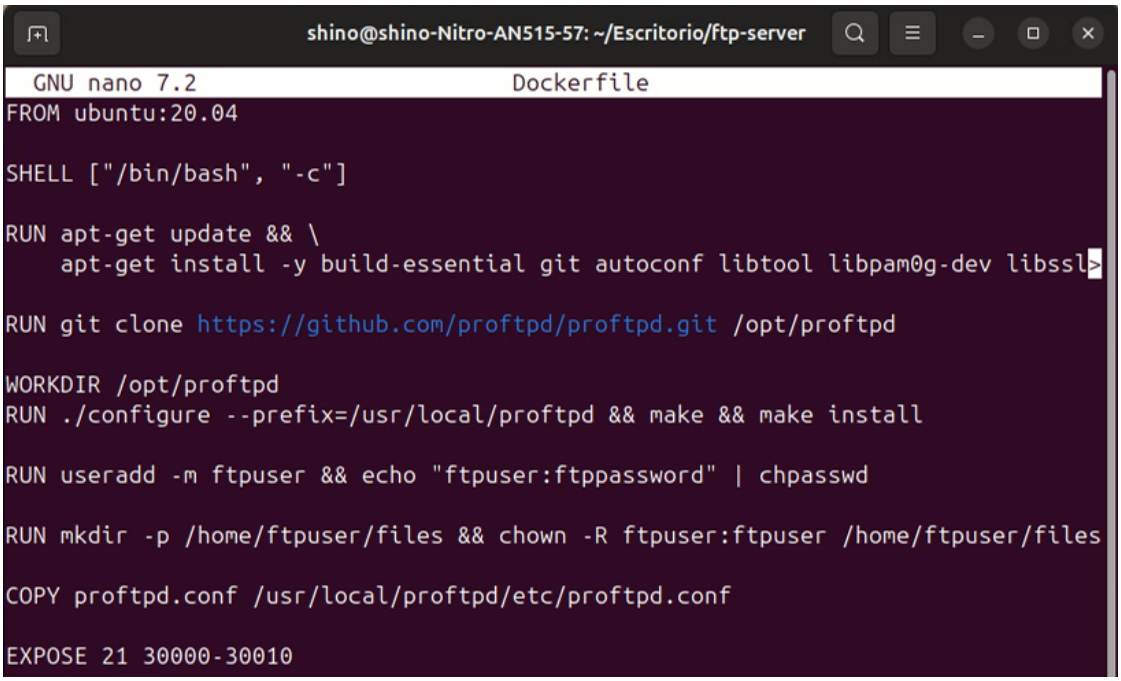


Figura 1.1: Dockerfile.

Los pasos realizados en el Dockerfile fueron los siguientes:

- 1. Se instalaron las dependencias necesarias:

  ```
  apt-get update && \
  apt-get install -y build-essential git autoconf libtool
  libpam0g-dev libssl-dev
  ```

- 2. Se importo el repositorio de GitHub mencionado anteriormente:

  ```
  git clone https://github.com/proftpd/proftpd.git /opt/proftpd
  ```

- 3. Compilacion y instalacion de ProFTPD:

  ```
  ./configure --prefix=/usr/local/proftpd && make && make install
  ```

- 4. Creación de usuario:

  ```
  useradd -m ftpuser && echo "ftpuser:ftppassword" | chpasswd
  ```

- 5. Definición de ubicación de los archivos del servidor:

```
mkdir -p /home/ftpuser/files &&
chown -R ftpuser:ftpuser /home/ftpuser/files
```

- 6. Copia del archivo de configuración para el funcionamiento del servidor:

```
proftpd.conf /usr/local/proftpd/etc/proftpd.conf
```

- 7. Exposición de los puertos necesarios:
  - Puerto 21, correspondiente al canal de control FTP.
  - Rango 30000-30010, necesario para el modo pasivo del protocolo FTP.
- 8. Ejecucion del servidor utilizando el comando CMD:

```
[/usr/local/proftpd/sbin/proftpd
-n -c /usr/local/proftpd/etc/proftpd.conf]
```

Con esta configuración, se logró implementar exitosamente un contenedor funcional como servidor FTP. Además, fue necesario liberar los puertos mencionados en el router para permitir el acceso desde redes externas, quedando preparado para recibir conexiones desde un cliente FTP ubicado en otro contenedor conectado desde una red distinta.

Figura 1.2: Puertos Liberados.

¿Por qué se libera el puerto 21?
El puerto 21 es el puerto estándar para el canal de control del protocolo FTP, el cual se utiliza para establecer una conexion inicial entre Cliente y el Servidor, ademas de enviar comandos como: [USER, PASS, LIST, RETR, STOR, etc]. Este canal permanece abierto durante toda la sesión FTP.

¿Por qué se libera un rango de puertos entre 30000 - 30010?
Durante una sesión FTP en modo pasivo (PASV), el servidor necesita abrir un segundo

canal de comunicación para la transferencia real de datos. En este modo, el cliente no recibe conexiones, sino que es él quien se conecta a un puerto aleatorio del servidor que se encuentra dentro del rango previamente definido.

La configuración realizada para ProFTPD fue la siguiente:

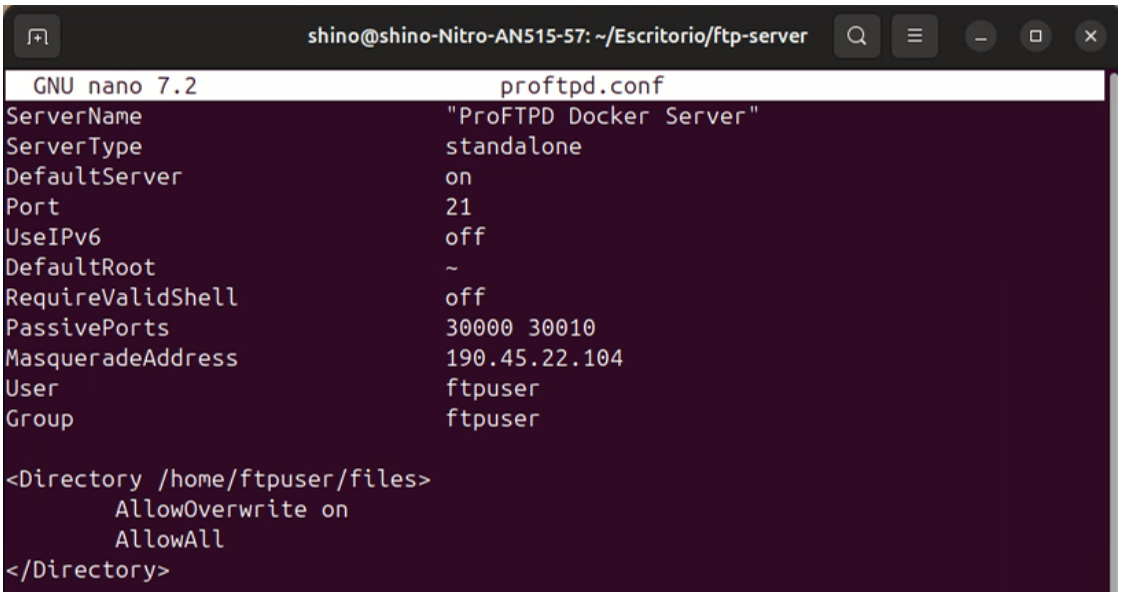


Figura 1.3: Configuracion ProFTPD.

Como se puede apreciar en la imagen, tenemos distintos tipos de campos especificos los cuales fueron configurados de la mejor forma para trabajar con el.

- 1. Primero se tiene el nombre del servidor FTP:

  ServerName ProFTPD Docker Server

- 2. Luego se tiene el tipo de servidor, a modo que este trabaje de forma independiente:

  ServerType standalone

- 3. Se tiene el puerto 21 para la conexión al servidor:

  Port 21

- 4. Se desactiva IPv6 para evitar problemas con el servidor y sus conexiones:

  UseIPv6 off

- 5. Se declara el DefaultRoot a modo de evitar la navegación entre carpetas del sistema por parte del cliente:

  DefaultRoot ~

- 6. Se especifican los puertos pasivos del servidor:

```
PassivePorts            30000 30010
```

- 6. Direccion IPv4 a la cual los clientes se conectan:

```
MasqueradeAddress       190.45.22.104
```

- 6. Finalmente se especifican las credenciales del cliente:

```
USER                    ftpuser
GROUP                   ftpuser
```

Cabe destacar que para el apartado de **MasqueradeAddress** se utiliza la ip publica de la red, para esto se utilizó el siguiente comando tal como muestra la imagen:

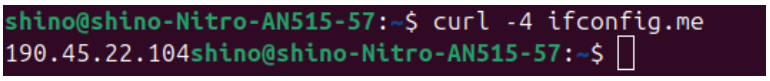


Figura 1.4: IPv4 publica de la red.

Una vez realizadas las configuraciones correspondientes para el Dockerfile y ProFTP, pasamos a la construcción del contenedor.

- Para esto se realizo el siguiente comando:

```
docker build --no-cache -t proftpd-server .
```

El cual podemos ver en la siguiente imagen:

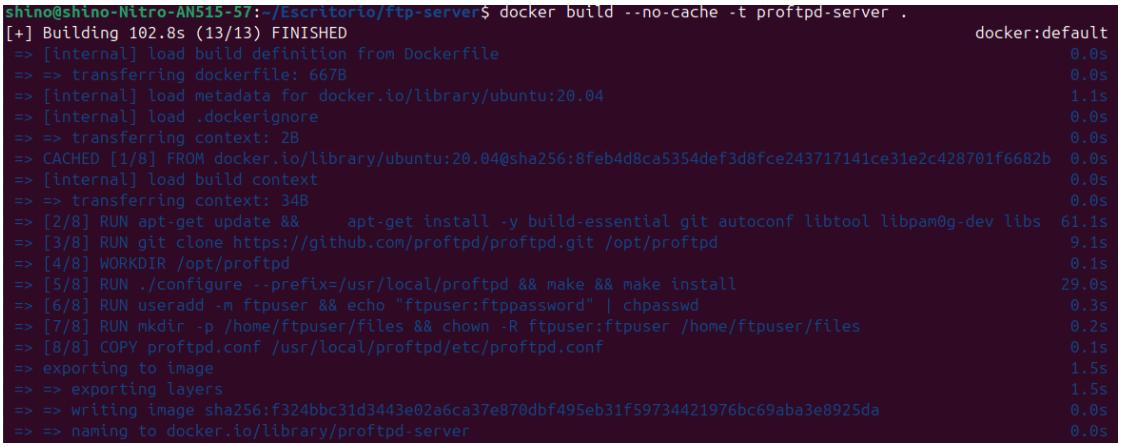


Figura 1.5: Docker build.

- Luego de esto, una vez cargados los archivos, corremos el contenedor con el siguiente comando:

```
docker run -d -p 21:21 -p 30000-30010:30000-30010
--name proftpd-server proftpd-server
```

```
shino@shino-Nitro-AN515-57:~/Escritorio/ftp-server$ docker run -d -p 21:21 -p 30000-30010:30000-30010 --name proftpd-
server proftpd-server
ce594acffc8230cfdc1380921d0c3667aee4043dab8e028baefa5372e48e0232
shino@shino-Nitro-AN515-57:~/Escritorio/ftp-server$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED         STATUS         PORTS
                                                                  NAMES
ce594acffc82   proftpd-server   "/usr/local/proftpd/…"   3 seconds ago   Up 3 seconds   0.0.0.0:21->21/tcp, :::21->21
/tcp, 0.0.0.0:30000-30010->30000-30010/tcp, :::30000-30010->30000-30010/tcp   proftpd-server
```

Figura 1.6: Docker Run - Docker ps

De la imagen podemos ver que el contenedor fue creado correctamente gracias a la configuración previa realizada. Se puede comprobar facilmente la existencia de este contenedor como se ve, gracias al comando **docker ps**. De esta forma nos aseguramos que está todo en orden en cuanto al servidor.

Finalmente, una vez terminada la configuracion del servidor FTP, se sube un archivo de texto, el cual contiene el mensaje: **Hola Cliente FTP**. Esto se realiza a modo de que el Cliente pueda interactuar con el, ya sea realizando una vista dentro del servidor, como descargar el archivo.

```
shino@shino-Nitro-AN515-57:~/Escritorio/ftp-server$ echo "Hola cliente FTP" > mensaje.txt
shino@shino-Nitro-AN515-57:~/Escritorio/ftp-server$ docker cp mensaje.txt proftpd-server:/home/ftpuser/files/
```

Figura 1.7: Creacion de archivo .txt

1.6.2. Cliente

Para conectarse al servidor FTP se utilizó **LFTP**, un cliente compatible con múltiples protocolos, entre ellos FTP, FTPS, HTTP y SFTP.

Se creó un contenedor Docker basado en `Ubuntu 20.04`, en el cual se instaló `lftp` utilizando el gestor de paquetes `apt`. El `Dockerfile` utilizado fue el siguiente:

```
virtualx@virtualx-VMware-Virtual-Platform: ~/Escrito...
  GNU nano 7.2                    Dockerfile *
FROM ubuntu:20.04

RUN apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y lftp

CMD ["/bin/bash"]

^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar      ^J Justificar
```

Figura 1.8: Dockerfile Cliente.

Se ejecutó con:

- `sudo docker build -t cliente-lftp .`
- `sudo docker run -it --network ftp-net cliente-lftp`

1.6.3. Conexión Cliente-Servidor

Dentro del contenedor, se estableció la conexión al servidor FTP mediante el siguiente comando:

- `lftp ftp://ftpuser:ftppassword@<ip-servidor>`

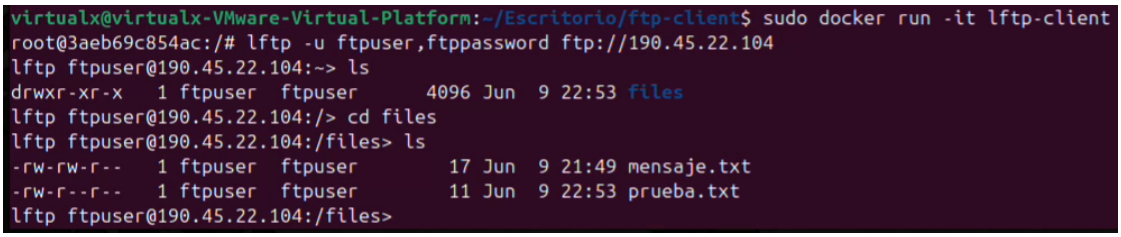


Figura 1.9: Conexión cliente-servidor.

Donde `<ip-servidor>` corresponde a la dirección IP asignada al contenedor del servidor FTP, que en este caso fue la 190.45.22.104. Una vez conectado, se utilizaron los comandos estándar de LFTP, tales como:

- `ls` para listar los archivos disponibles en el servidor.
- `get <archivo>` para descargar archivos desde el servidor.
- `put <archivo>` para subir archivos al servidor.
- `exit` para cerrar la sesión FTP.

Esta configuración permitió establecer una comunicación efectiva entre cliente y servidor, lo cual fue verificado mediante análisis de tráfico de red con Wireshark posteriormente.

Capítulo 2

Análisis de tráfico

2.1. Tráfico observado

2.1.1. Paquete de Conexión al servidor

Se identificó el establecimiento de la conexión TCP entre el cliente `192.168.10.60` y el servidor `190.22.97.179` a través del puerto `21` (FTP). Este proceso se observa mediante el protocolo TCP en tres pasos conocidos como **Three-way Handshake**, presentes en los siguientes paquetes:

- **Paquete 2289 – SYN**: El cliente inicia la conexión con el servidor enviando un paquete con la bandera `SYN`, desde el puerto `49882` hacia el puerto `21`. Se establece el número de secuencia inicial y se indican opciones TCP como `MSS`, `SACK PERM`, `Timestamps` y `Window Scale`.
- **Paquete 2290 – SYN, ACK**: El servidor responde aceptando la conexión. Este paquete incluye las banderas `SYN` y `ACK`, reconociendo el número de secuencia del cliente y enviando el propio.
- **Paquete 2292 – ACK**: El cliente responde con un paquete con la bandera `ACK`, finalizando el proceso de sincronización. La conexión TCP queda así establecida.

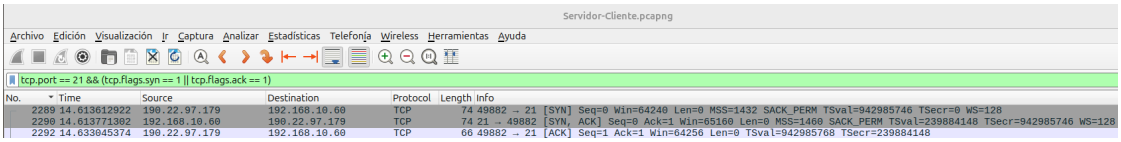


Figura 2.1: Three-way Handshake entre cliente y servidor FTP observado en Wireshark.

Filtro Wireshark
La explicación de la elección de dichos filtros son los siguientes:

- **1. Puerto de control FTP**: El protocolo FTP utiliza siempre el puerto TCP 21 para su canal de control. Y mediante el filtro:

  `tcp.port == 21`

  Obtenemos un enfoque especifico solo en este trafico.

- **2. Enfoque en el Three-Way Handshake**:
  - `tcp.flags.syn == 1`

    Esta parte del filtro busca paquetes donde el flag SYN está activado (es igual a 1). Los paquetes SYN son el primer paso del three-way handshake.

- `tcp.flags.ack == 1`

  Esta parte del filtro busca paquetes donde el flag ACK está activado (es igual a 1). ya que los paquetes ACK son cruciales, por que confirman la recepción de datos.

2.1.2. Paquetes de autenticación del servidor

Método de login:
El protocolo FTP utiliza un método de autenticación basado en el envío explícito de un nombre de usuario (USER) y una contraseña (PASS) para acceder al servidor.

Estructura del paquete:
El proceso de autenticación consta de dos paquetes principales capturados durante la sesión:

- **Paquete USER:**
  - Origen (Source): 190.22.97.179 (IP del cliente que intenta conectarse)
  - Destino (Destination): 192.168.10.60 (IP del servidor FTP)
  - Protocolo: FTP
  - Longitud (Length): 80 bytes
  - Solicitud (Request): USER `ftpuser`

  Este paquete contiene el comando `USER` seguido del nombre de usuario enviado.
- **Paquete PASS:**
  - Origen (Source): 190.22.97.179
  - Destino (Destination): 192.168.10.60
  - Protocolo: FTP
  - Longitud (Length): 84 bytes
  - Solicitud (Request): PASS `ftppassword`

  Este paquete contiene el comando `PASS` seguido de la contraseña correspondiente, también enviada en texto claro.

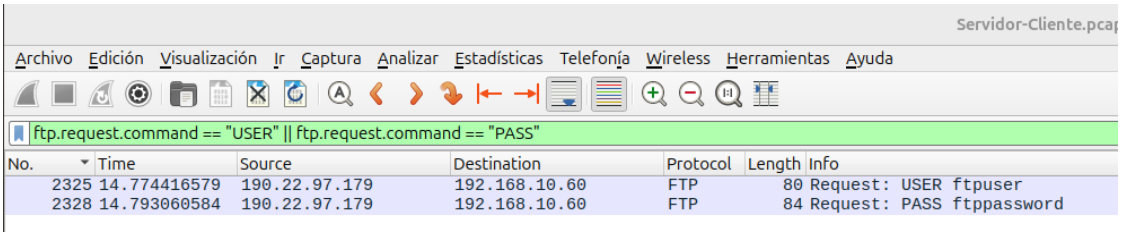


Figura 2.2: Three-way Handshake entre cliente y servidor FTP observado en Wireshark.

Filtro Wireshark
La explicación de la elección de dichos filtros son los siguientes:

- **1.**

  `ftp.request.command == "USER"`

  En FTP el primer comando que suele enviar para la autenticación es **USER**. Al incluir esta parte del filtro, Wireshark buscará y mostrará cualquier paquete FTP donde el comando de solicitud sea **USER**.

- **2.**

  `ftp.request.command == "PASS"`

  Luego de la recepción del comando USER por el cliente FTP viene su correspondiente paquete que trae el **PASS**(password) la cual es necesaria para establecer la conexión. Al incluir esta parte del filtro, Wireshark también buscará y mostrará cualquier paquete FTP donde el comando de solicitud sea **PASS**.

2.1.3. Paquetes de Listado de Archivos

Durante la sesión capturada en Wireshark, se identificaron comandos FTP utilizados para solicitar listados de archivos en el servidor. Los comandos observados fueron `LIST` y `MLSD`, ambos empleados para obtener el contenido del directorio.

- `LIST`: Devuelve un listado de archivos en formato tradicional.
- `MLSD`: Devuelve un listado en formato estructurado y uniforme, más adecuado para procesamiento automático.

Los paquetes observados fueron:

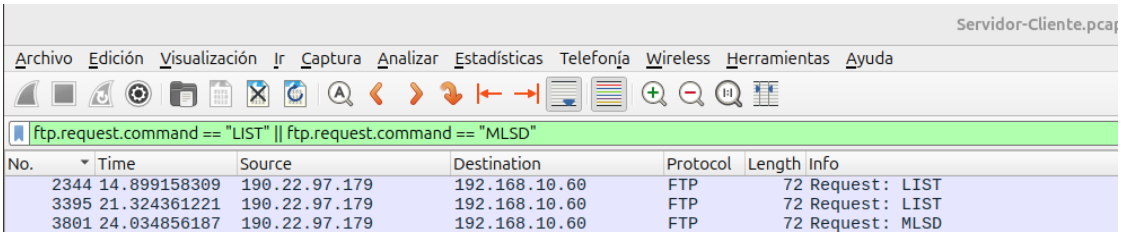


Figura 2.3: Three-way Handshake entre cliente y servidor FTP observado en Wireshark (paquetes 2289, 2290, 2292).

Filtro Wireshark
La explicación de la elección de dichos filtros son los siguientes:

- **1.**

  `ftp.request.command == "LIST"`

  El comando LIST es uno de los comandos FTP que se utiliza para solicitar al servidor FTP un listado detallado de los archivos y directorios, equivalente al

ls en linux o **dir** respectivamente en windows. Al incluir este filtro, Wireshark mostrará cualquier paquete donde el cliente esté solicitando esta información.

- **2.**

  `ftp.request.command == "MLSD"`

  MLSD es un comando FTP más moderno para solicitar listados de directorios. Dependiendo del cliente FTP se puede preferir **MLSD** si el servidor lo soporta. Al incluir MLSD, se puede capturar también estos comandos en caso de que el cliente o el servidor FTP lo estén utilizando.

Todos los comandos fueron transmitidos por el protocolo FTP mediante el canal de control, utilizando TCP como protocolo de transporte. Los paquetes tienen una longitud similar (72 bytes), lo que indica que el comando se transmite de forma compacta y eficiente. Una vez enviado el comando, el servidor responde a través del canal de datos con el contenido solicitado.

2.1.4. Paquetes de Descarga de archivo

Se capturó un paquete con el comando `RETR`, utilizado para iniciar la descarga de un archivo desde el servidor hacia el cliente. En este caso, el archivo solicitado fue `mensaje.txt`.

- **Comando utilizado**: `RETR mensaje.txt`
- **Tamaño del paquete**: 84 bytes
- **Protocolo**: FTP sobre TCP
- **Transferencia**: A través del canal de datos una vez establecida la instrucción

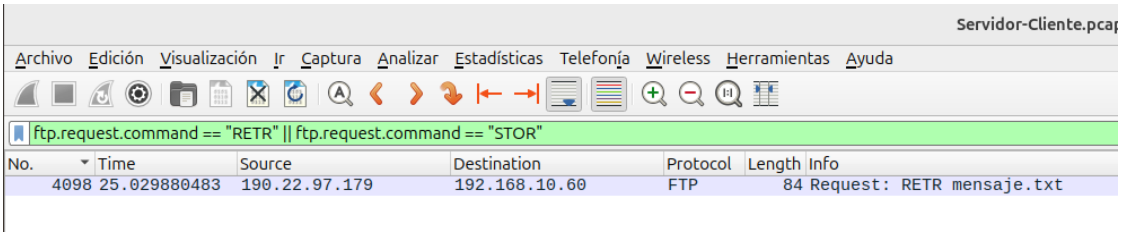


Figura 2.4: Three-way Handshake entre cliente y servidor FTP observado en Wireshark (paquetes 2289, 2290, 2292).

Filtro Wireshark
La explicación de la elección de dichos filtros son los siguientes:

- **1.**

  `ftp.request.command == "RETR"`

  RETR es el comando FTP que un cliente envía al servidor cuando desea descargar un archivo desde el servidor, este es acompañado del nombre del archivo seleccionado por el cliente para poder realizar la descarga. Por lo que utilizar este filtro nos sirve para capturar las solicitudes de descarga en el servidor.

- **2.**

  `ftp.request.command == "STOR"`

  STOR es el comando FTP que un cliente envía al servidor cuando desea subir un archivo al servidor, igualmente va acompañado de el nombre del archivo a subir en el servidor. Al usar este filtro obtenemos paquetes donde se realice la carga de archivos.

Este paquete representa una solicitud del cliente para recuperar el archivo desde el servidor. Posteriormente, la transferencia del archivo se realiza mediante una conexión TCP adicional establecida por el canal de datos FTP, asegurando la entrega ordenada y completa del contenido solicitado.

2.2. Suposiciones sobre posibles repercusiones en el software al modificar el tráfico

Durante el análisis del tráfico FTP, se pueden formular diversas suposiciones sobre el comportamiento del software cliente-servidor al producirse alteraciones en los paquetes transmitidos.

Una primera suposición es que si se interrumpe o modifica el *three-way handshake* (por ejemplo, si el paquete ACK final nunca llega), la conexión TCP no podrá establecerse, impidiendo cualquier comunicación posterior entre cliente y servidor. Esta condición impediría el uso del protocolo FTP desde el principio.

Si se interceptan o alteran los paquetes de **autenticación** (comandos `USER` o `PASS`), podrían producirse errores como autenticación fallida o acceso denegado. Dado que estos comandos viajan en texto plano en FTP, también podrían ser fácilmente explotados por atacantes mediante técnicas de *sniffing*, lo cual representa una amenaza crítica para la seguridad.

En cuanto al tráfico de **listado de archivos** o **transferencias**, si se manipulan los paquetes de control (como `LIST` o `RETR`), el cliente podría recibir respuestas incompletas, inconsistentes o vacías. Además, alterar los paquetes del canal de datos podría provocar archivos corruptos, interrupciones de descarga o incluso errores de sincronización que lleven a la desconexión de la sesión FTP.

Finalmente, si un atacante realiza una inyección de paquetes maliciosos o modifica el tráfico legítimo, podrían desencadenarse fallos inesperados en el cliente o servidor, dependiendo de su robustez y validación de datos. En entornos sin cifrado, estas alteraciones pueden representar un riesgo significativo.

Capítulo 3

Tarea 3

3.1. Introducción y Configuración del Entorno

Considerando tu informe anterior de la Tarea 2 sobre el protocolo FTP y las capturas de pantalla que has proporcionado para la Tarea 3, a continuación te detallo la sección que debes agregar a tu informe. Esta sección se enfocará en las actividades realizadas con Scapy, las inyecciones de tráfico (fuzzing) y las modificaciones de campos específicos del protocolo FTP, junto con sus respectivas repercusiones.

3. Tarea 3: Inyección y Modificación de Tráfico en el Protocolo FTP 3.1. Introducción y Configuración del Entorno En esta sección se detallan las actividades realizadas para interceptar, inyectar y modificar el tráfico del protocolo FTP, basándose en el entorno cliente-servidor establecido en la Tarea 2. El objetivo principal fue analizar cómo el servicio responde a tráficos no esperados o manipulados, utilizando la herramienta Scapy.

Para la ejecución de esta tarea, se mantuvo el entorno Docker con el servidor y cliente FTP configurados previamente. Se añadió un contenedor o se instaló localmente la última versión de Scapy para interactuar con el tráfico de red.

Configuración del entorno Scapy: Se utilizó un entorno Docker basado en Python para la ejecución de Scapy. Como se puede observar (Fig. 3.1).

```
1  FROM python:3.10-slim
2
3  RUN apt-get update && apt-get install -y tcpdump iproute2 iputils-ping net-tools
4
5  RUN pip install --no-cache-dir scapy
6
7  CMD ["/bin/bash"]
```

Figura 3.1: Dockerfile Scapy.

A continuación, se muestra el estado inicial del entorno Docker.

Ejecución del entorno Docker

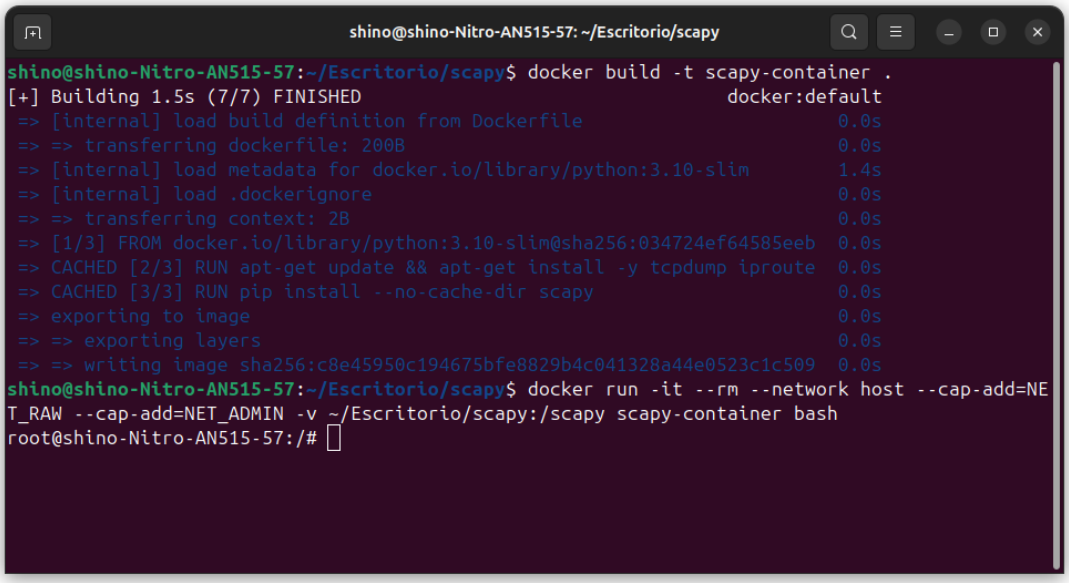


Figura 3.2: Docker Build-Run Scapy.

Gracias a esto se logró la ejecución del contenedor de Scapy con el cual se realizaron los siguientes pasos para la inyección y modificación del tráfico FTP.

3.2. Intercepción y Manipulación de Tráfico con Scapy

Intercepción de Tráfico con Scapy

Se utilizó Scapy para interceptar el tráfico entre el cliente y el servidor FTP. mediante el siguiente script que se encarga de "escuchar" el tráfico TCP específicamente en el puerto 21 (FTP). Entregando en la terminal el tráfico que intercepta (Fig 3.3).

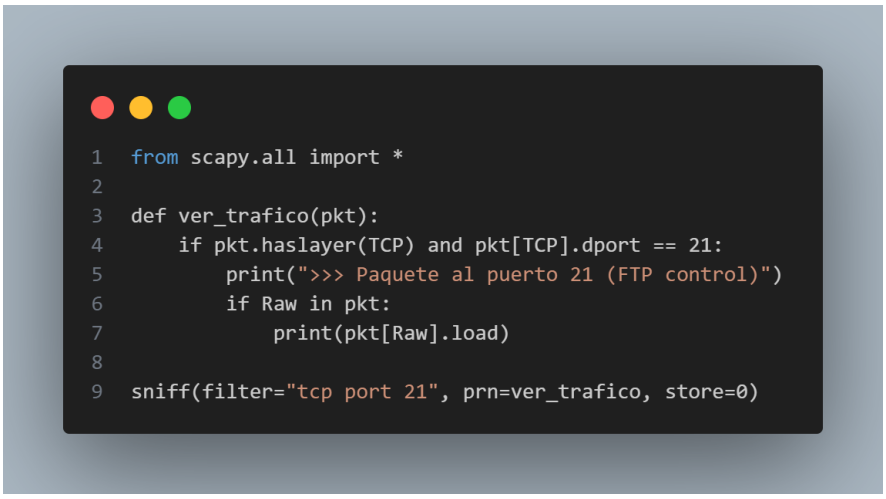


Figura 3.3: Script trafico Scapy.

Inyección de Tráfico con Fuzzing

Se realizaron inyecciones de tráfico utilizando técnicas de fuzzing para probar la robustez del servidor FTP ante entradas inesperadas o malformadas. Ya que mediante el fuzzing que su principal función es enviar datos aleatorios o inesperados a una aplicación para descubrir vulnerabilidades.
Con el siguiente script que genera paquetes del tipo SYN, SYN-ACK, ACK:

```python
from scapy.all import *
import random
import string

ip = "190.45.22.104"   # IP del servidor FTP
port = 21              # Puerto FTP

def comando_fuzz():
    return ''.join(random.choices(string.ascii_uppercase + string.digits, k=8)).encode() + b"\r\n"

SYN = IP(dst=ip)/TCP(dport=port, flags='S')
SYNACK = sr1(SYN, timeout=2)

if SYNACK:
    ACK = IP(dst=ip)/TCP(dport=port, sport=SYN[TCP].sport, seq=100, ack=SYNACK.seq + 1, flags='A')
    send(ACK)

    for _ in range(2):
        fuzz_data = comando_fuzz()
        pkt = IP(dst=ip)/TCP(dport=port, sport=SYN[TCP].sport, seq=101, ack=SYNACK.seq + 1, flags='PA')/Raw(load=fuzz_data)
        send(pkt)
        print(f"Inyectado: {fuzz_data}")
else:
    print("No hubo respuesta al SYN")
```

Figura 3.4: Script paquetes Fuzzing.

A continuación se ejecuta el script el cual realiza las dos inyecciones de trafico, junto a la captura de trafico correspondiente del servidor (Fig. 3.5).

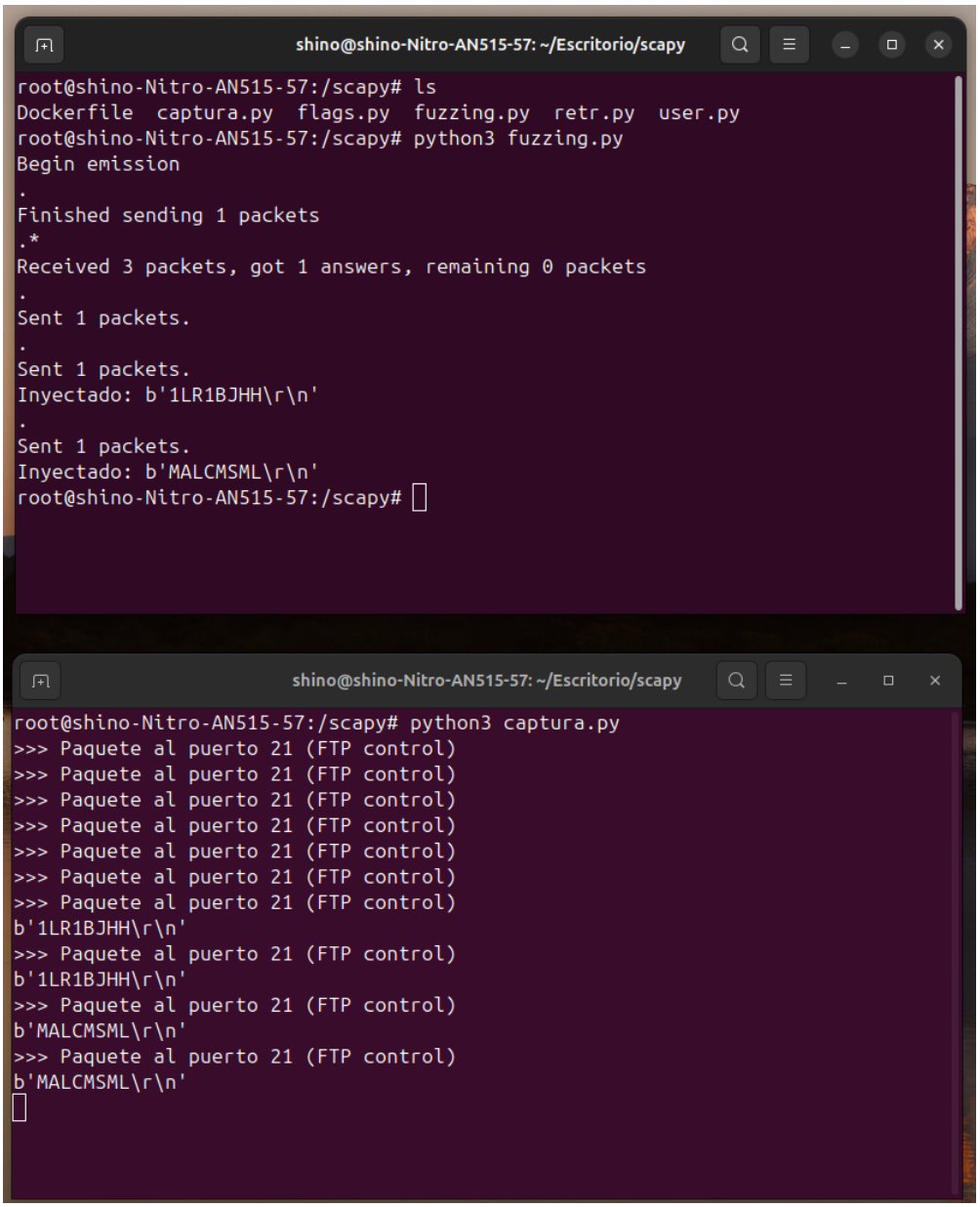


Figura 3.5: Ejecucion script paquetes Fuzzing.

Ejecutando el script con el contenedor Scapy se logró observar efectivamente la inyección de los paquetes, tanto su ejecución como en la captura de trafico. Sin embargo, el servidor ignora estos comandos y continua funcionando sin errores, demostrando estabilidad ante tráfico inesperado (Fig. 3.5).

Modificación del comando USER

A continuacion, se realiza la modificacion del comando **USER**, el cual busca alterar el proceso de autenticación del protocolo FTP, modificando el comando USER para interrumpir el inicio de sesión del cliente. Esto es posible gracias al siguiente script (Fig 3.6).

```python
from scapy.all import *

ip_server = "190.45.22.104"

pkt = IP(dst=ip_server)/TCP(dport=21, sport=RandShort(), flags="S")

syn_ack = sr1(pkt, timeout=2)

if syn_ack is None:
    print("Servidor no respondiÃ³ al SYN")
    exit()

ack = IP(dst=ip_server)/TCP(dport=21, sport=pkt[TCP].sport,
                            seq=syn_ack.ack, ack=syn_ack.seq + 1, flags="A")
send(ack)


payload = b"USEA ftpuser\r\n"
psh = IP(dst=ip_server)/TCP(dport=21, sport=pkt[TCP].sport,
                            seq=syn_ack.ack, ack=syn_ack.seq + 1, flags="PA")/Raw(load=payload)
send(psh)

print("Comando 'USEA ftpuser' enviado")
```

Figura 3.6: script USER.

El script intercepta el paquete que contenía el comando USER ftpuser y modifica el campo del protocolo para que diga USEA ftpuser, el cual no es un comando válido en FTP. A continuacion se ejecuta el codigo junto a su captura de tráfico (Fig 3.7).

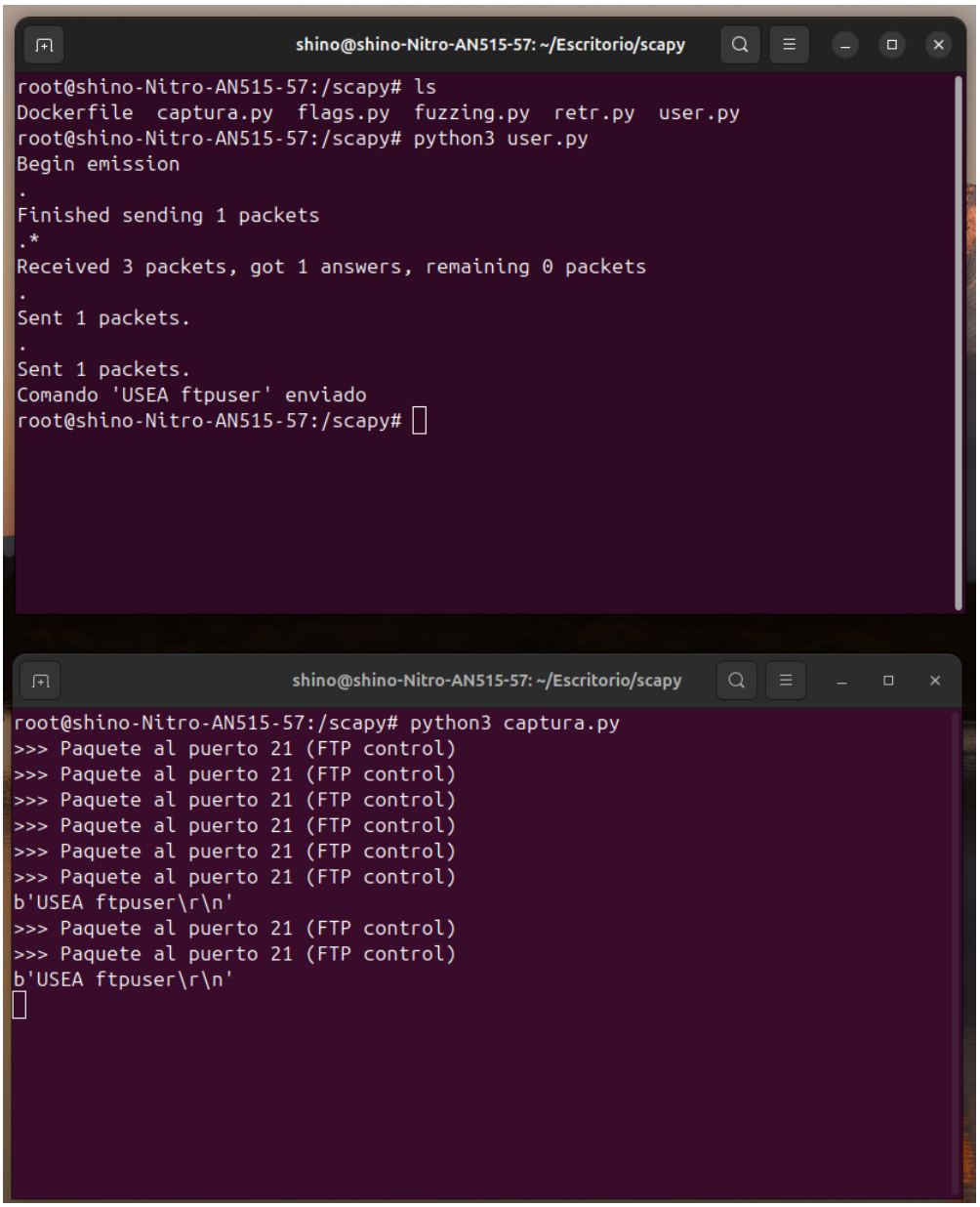
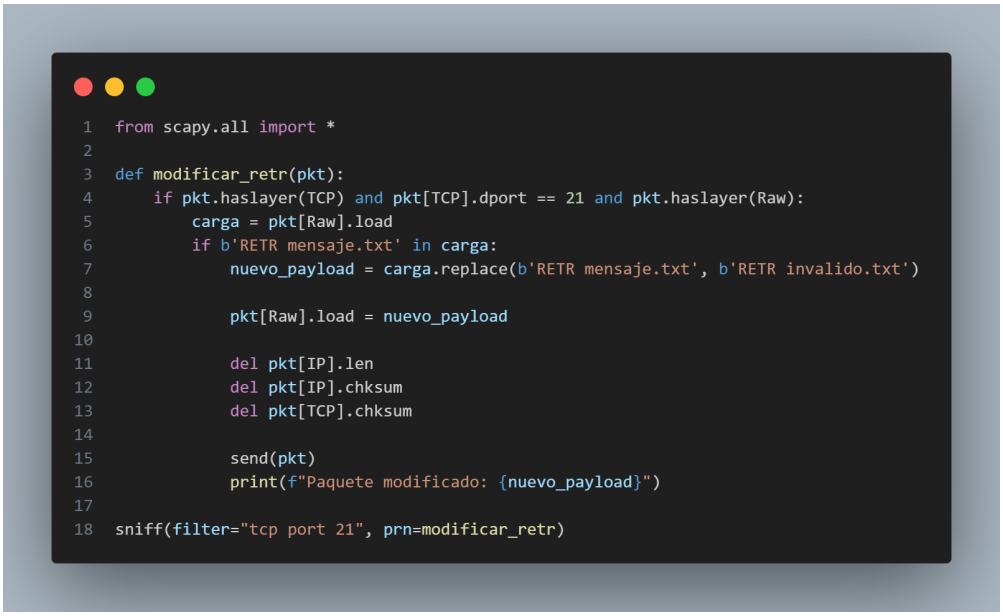


Figura 3.7: Ejecucion y captura de script USER.

Como se puede ver, el sniffer (captura de tráfico) capturó correctamente el paquete modificado (USEA ftpuser), pero el paquete original con USER ftpuser también llegó al servidor. Como resultado, el servidor procesó el original y permitió el inicio de sesión. Por lo que la modificación del campo fue exitosa y visible, pero sin efecto funcional debido a que el paquete original no fue bloqueado. Sin embargo el campo fue interceptado y alterado correctamente.

Modificación del comando RETR

En este caso se busca interrumpir la transferencia de archivos modificando el nombre del archivo solicitado para que el servidor intente enviar un archivo inexistente. El script a utilizar es el siguiente (Fig. 3.8).



```python
from scapy.all import *

def modificar_retr(pkt):
    if pkt.haslayer(TCP) and pkt[TCP].dport == 21 and pkt.haslayer(Raw):
        carga = pkt[Raw].load
        if b'RETR mensaje.txt' in carga:
            nuevo_payload = carga.replace(b'RETR mensaje.txt', b'RETR invalido.txt')

            pkt[Raw].load = nuevo_payload

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

            send(pkt)
            print(f"Paquete modificado: {nuevo_payload}")

sniff(filter="tcp port 21", prn=modificar_retr)
```

Figura 3.8: Script para la modificacion de archivos.

El script intercepta el paquete con el comando RETR mensaje.txt y se modifica a RETR invalido.txt. A continuacion se ejecuta el script junto a su captura de trafico.

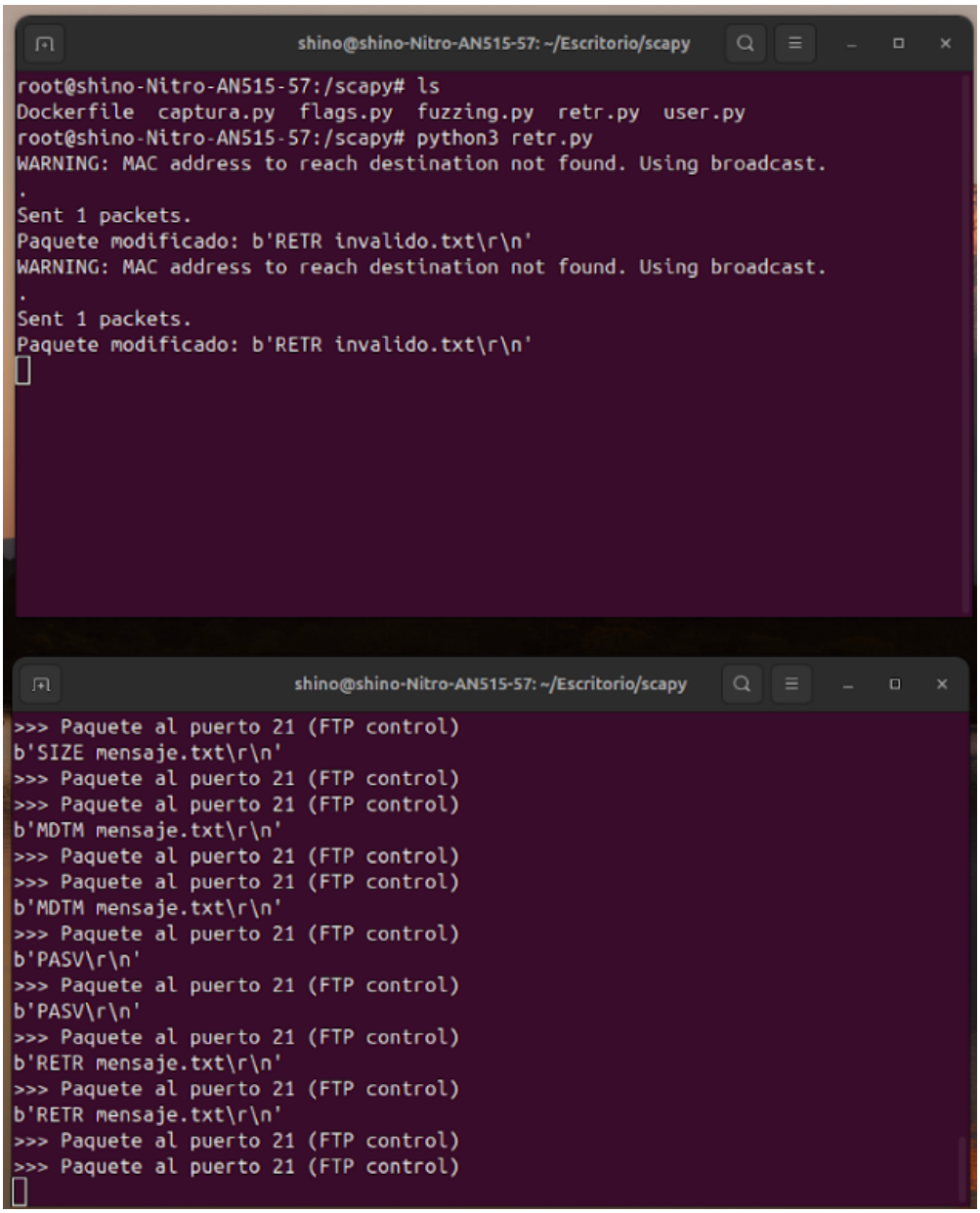


Figura 3.9: Ejecución del script RETR.

Como se puede ver, dentro de la ejecucion se muestra que el paquete fue modificado correctamente, por lo que a continuación se muestra como responde el cliente ante al script (Fig 3.10).

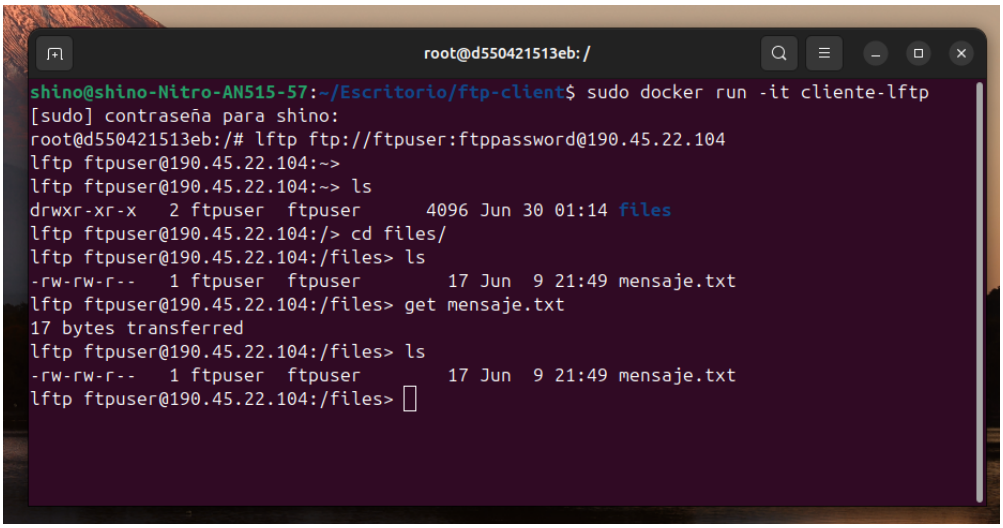


Figura 3.10: Cliente FTP.

Finalmente podemos ver que el archivo mensaje.txt se descargó correctamente, lo que indica que el paquete original con RETR mensaje.txt fue procesado por el servidor antes que el modificado. Por lo que el sniffer (Fig 3.9) no mostró respuesta al paquete falso. Por lo que aunque el campo haya sido modificado, la alteracion de este no tuvo impacto en la funcionalidad del servidor.

Modificación de flags TCP en paquete FTP

El protocolo TCP, es la base de FTP, el cual utiliza flags (como SYN, ACK, PSH, FIN, etc.) para controlar la conexión, garantizar la entrega de datos y sincronizar estados entre cliente y servidor.

Modificar estos flags puede interrumpir la comunicación o provocar comportamientos no deseados si el paquete modificado llega a ser procesado. Esto es posible gracias al siguiente script (Fig. 3.11).

```python
from scapy.all import *

ip_dst = "190.45.22.104"  # IP del servidor
puerto = 21  # Puerto FTP

pkt = IP(dst=ip_dst)/TCP(sport=RandShort(), dport=puerto, flags="P")/Raw(load="PWD\r\n")

send(pkt)

print("Paquete con flags modificadas enviado (solo PSH, sin ACK)")
```

Figura 3.11: Script para la modificación de los FLAGS.

Gracias al scrpit se puede interceptar un paquete FTP válido desde el cliente, por ejemplo, con el comando PWD ˚ el cual consulta la ruta actual del usuario. A continuacion se realiza la ejecución del mismo.

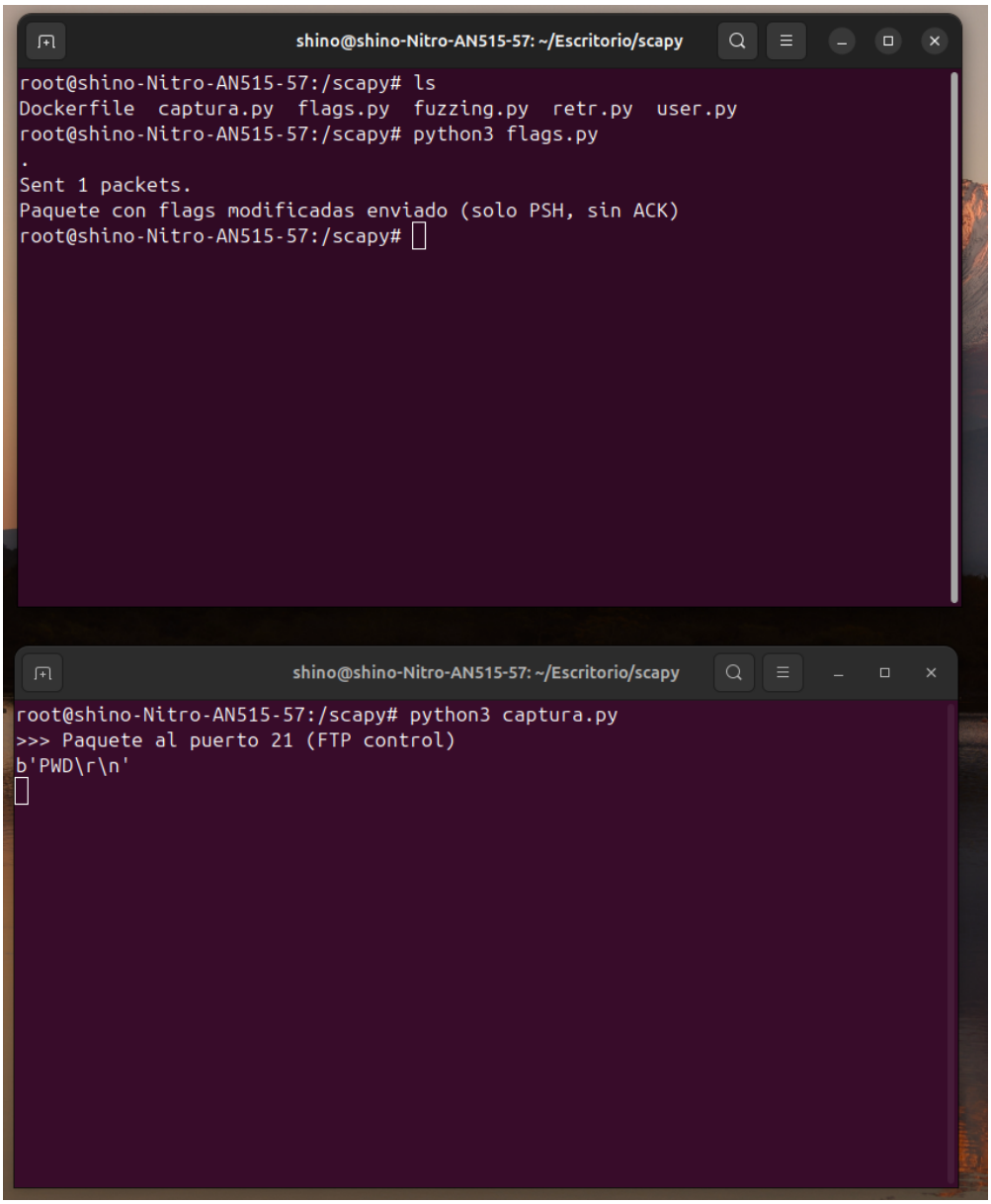


Figura 3.12: Ejecución de script.

Como podemos ver tanto en la ejecución como en la captura, el paquete fue modificado e inyectado, pero el cliente y el servidor continuaron comunicándose sin problemas. El paquete con flags alterados fue ignorado, posiblemente por tener un número de secuencia incorrecto. Por lo que aunque no tuvo un efecto visible, se logro modificar el campo del protocolo correctamente.

3.3. Análisis y conclusiones

Durante esta actividad se logró interceptar, modificar e inyectar tráfico en tiempo real entre un cliente y servidor FTP utilizando Scapy, cumpliendo con los objetivos centrales de la tarea. Se llevaron a cabo dos tipos principales de pruebas: fuzzing e inyecciones con modificaciones específicas a campos del protocolo FTP y del protocolo de transporte TCP. En el caso del fuzzing, se enviaron comandos aleatorios no válidos al servidor, los cuales fueron correctamente ignorados sin afectar el funcionamiento del servicio, evidenciando que el servidor es tolerante a entradas inesperadas. En cuanto a las modificaciones, se alteraron comandos clave como USER y RETR, así como los flags del encabezado TCP, logrando capturar y modificar los paquetes, aunque en la mayoría de los casos el comportamiento del servidor no cambió porque los paquetes originales seguían siendo procesados. Esto demuestra que, si bien la manipulación técnica fue exitosa, generar un impacto visible en la sesión requiere bloquear o interceptar completamente los paquetes legítimos. En resumen, la experiencia permitió comprender a fondo cómo funciona el protocolo FTP y cómo se puede manipular tráfico de red con herramientas de bajo nivel, lo que refuerza la importancia de implementar mecanismos de seguridad y validación frente a posibles ataques o alteraciones de red.

Conclusión

La realización de esta tarea permitió comprender en profundidad el funcionamiento del protocolo FTP desde una perspectiva tanto teórica como práctica. A través de la implementación de un entorno distribuido mediante contenedores Docker, se logró instalar, configurar y ejecutar exitosamente un servidor FTP utilizando ProFTPD, y un cliente mediante LFTP.

La conexión entre cliente y servidor fue establecida correctamente, permitiendo la autenticación, el listado de archivos y la transferencia de datos, todo ello verificado mediante el análisis del tráfico de red con Wireshark. Esta herramienta permitió observar el comportamiento detallado del protocolo, incluyendo el establecimiento de la conexión TCP (Three-Way Handshake), la autenticación con usuario y contraseña, y las transferencias mediante comandos como `LIST`, `RETR` y `STOR`.

Durante la implementación práctica, una de las principales interferencias encontradas fue la necesidad de liberar manualmente los puertos en el router donde se ejecutó el servidor FTP. Por defecto, los puertos utilizados por el protocolo (21 para control y el rango 30000–30010 para modo pasivo) no estaban habilitados, lo que impedía la conexión desde redes externas. Esta situación evidenció la importancia de considerar la configuración del entorno de red al desplegar servicios en contenedores, especialmente en escenarios que involucran NAT o firewalls intermedios.

Además, se identificaron las vulnerabilidades relacionadas al protocolo FTP, como el envío de credenciales en texto plano, lo que refuerza la importancia de emplear variantes seguras (como FTPS o SFTP) en entornos donde la seguridad es crítica.

En general, esta experiencia facilitó el desarrollo de habilidades prácticas en redes y servicios, fomentando el uso de herramientas modernas como Docker para simular infraestructuras reales. También permitió comprende conceptos fundamentales sobre protocolos de aplicación, captura y análisis de tráfico, así como sobre la configuración de servicios en sistemas basados en Linux(Ubuntu).

Bibliografía

[1] Canonical Ltd. (2019–2025). *lftp — Sophisticated file transfer program*. Ubuntu Manpage Repository.
https://manpages.ubuntu.com/manpages/bionic/en/man1/lftp.1.html

[2] Ubuntu Community. (s.f.). *FTP server installation and configuration*. Ubuntu Community Help Wiki.
https://help.ubuntu.com/community/vsftpd

[3] FileZilla Project. (s.f.). *FileZilla – The free FTP solution*.
https://filezilla-project.org

[4] IBM Knowledge Center. (s.f.). *File Transfer Protocol - FTP*. Documentación oficial de IBM AIX 7.1.
https://www.ibm.com/docs/es/aix/7.1.0?topic=protocols-file-transfer-protocol

[5] Encyclopædia Britannica. (s.f.). *FTP – File Transfer Protocol*.
https://www.britannica.com/technology/FTP

[6] WebAsha Technologies. (2023). *Which Protocols Are Used for File Sharing? Guide to FTP, SFTP, SMB, and More*.
https://www.webasha.com/blog/which-protocols-are-used-for-file-sharing-guide-to-ftp-sftp-smb-and-more