

Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Simulador de puertos y servicios de red en un sistema embebido
Proyecto Tecnológico

Trimestre 2018 Primavera

Alfredo Quiroz Rodríguez
2133001968
al2133001968@azc.uam.mx

Ing. Mario Alberto Lagos Acosta
Profesor Asociado Tipo D
Departamento de Electrónica
mlagos@correo.azc.uam.mx

26 de julio del 2018

Declaratoria

Yo, Mario Alberto Lagos Acosta, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Firma del Asesor

Yo, Alfredo Quiroz Rodríguez, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Firma del Alumno

Resumen

En el presente reporte se detalla la realización de un programa para la apertura simultanea de hasta diez puertos de red TCP o UDP, los cuales son capaces de ser identificados a través de un *fingerprint* como si se tratase de un servicio de red previamente instalado en un dispositivo. La identificación se lleva a cabo con una herramienta de escaneo de puertos, los puertos/servicios a ser identificados ya están previamente designados, sin embargo, el programa puede generar múltiples escenarios utilizando diferentes cantidades de puertos.

En el presente reporte se simularon los servicios HTTP, SMTP, POP3, FTP, DNS, IRC, IMAP, Syslog, SNMP y DHCP, usando un sistema embebido para realizar un escaneo de puertos controlado e identificar los servicios antes mencionados.

Tabla de contenido

1. Introducción.....	1
2. Antecedentes.....	2
2.1. Proyectos de Integración	2
2.2. Tesis	2
2.3. Software	2
3. Justificación.....	3
4. Objetivos.....	4
4.1. Objetivo General	4
4.2. Objetivos Específicos	4
5. Marco Teórico	5
5.1 Raspberry Pi 3	5
5.2 Kali Linux	5
5.3 Puerto de Sockets	5
5.4 Escaneo de puertos	5
6. Desarrollo del Proyecto.....	6
6.1. Instalación y configuración del Sistema Operativo en la Raspberry Pi 3.....	6
6.2. Estructura del script.....	6
6.3. Módulo de servicio HTTP.....	7
6.4. Módulo de servicio SMTP.....	7
6.5. Módulo de servicio POP3	8
6.6. Módulo de servicio FTP.....	8
6.7. Módulo de servicio DNS	9
6.8. Módulo de servicio IRC.....	9
6.9. Módulo de servicio IMAP	10
6.10. Módulo de servicio Syslog.....	10
6.11. Módulo de servicio SNMP	11
6.12. Módulo de servicio DHCP.....	11
6.13. Módulo de control de servicios	12
6.14. Obtención de <i>fingerprints</i>	12
7. Resultados	16
8. Conclusiones	19

9. Bibliografía	20
10. Apéndices	20

Tabla de Figuras

Figura 1.1. Sistema Embebido Raspberry Pi 3	5
Figura 2.1. Imagen ISO para Raspberry Pi 3	6
Figura 3.1. Módulos del script	7
Figura 3.2. Código del módulo del servicio HTTP	7
Figura 3.3. Código del módulo del servicio SMTP	8
Figura 3.4. Código del módulo del servicio POP3	8
Figura 3.5. Código del módulo del servicio FTP	9
Figura 3.6. Código del módulo del servicio DNS	9
Figura 3.7. Código del módulo del servicio IRC	10
Figura 3.8. Código del módulo del servicio IMAP	10
Figura 3.9. Código del módulo del servicio Syslog	11
Figura 3.10. Código del módulo del servicio SNMP	11
Figura 3.11. Código del módulo del servicio POP3	11
Figura 4.1. Configuración del archivo puertos.txt	12
Figura 5.1. Fingerprint servicio HTTP	13
Figura 5.2. Fingerprint servicio SMTP	13
Figura 5.3. Fingerprint servicio POP3	14
Figura 5.4. Fingerprint servicio FTP	14
Figura 5.5. Fingerprint servicio DNS	15
Figura 5.6. Fingerprint servicio IRC	15
Figura 5.7. Fingerprint servicio IMAP	16
Figura 6.1. Ejecución del script	17
Figura 6.2. Comando para verificar la apertura de puertos	17
Figura 6.3. Ejecución del escaneo con Nmap	18
Figura 6.4. Salida de la terminal	18

1. Introducción

La mayoría de las aplicaciones que utilizamos a diario requieren conectarse a internet para brindar sus servicios y funciones, esto conlleva la apertura de puertos de red con el fin de establecer una comunicación haciendo uso de los protocolos TCP o UDP. También implica un riesgo o peligro, debido a que se puede efectuar un ataque usando una vulnerabilidad de algún puerto de red activo en el dispositivo.

Debido a la situación anteriormente mencionada, todo el tiempo nuestros dispositivos están en constante apertura y cierre de puertos, esto pasa de forma transparente, sin poderemos percatar del posible peligro al que se está expuesto. Por ello, es importante realizar un análisis de los puertos para identificar estos peligros y tomar acciones al respecto.

Existen herramientas para descubrir qué puertos de red están abiertos, las cuales realizan preguntas al sistema objetivo y esperan una respuesta acorde. Esto también se conoce como "Sondeo de Puertos", existen diferentes técnicas como sondeo TCP SYN, sondeo TCP connect (), sondeo UDP, entre otros. Se tienen puertos asociados a un determinado servicio, por ejemplo, el puerto 67 con el servicio de DHCP. Dicho de otra forma, todo el tráfico perteneciente a este servicio se recibirá y enviará por el puerto antes mencionado.

2. Antecedentes

2.1. Proyectos de Integración

- Análisis de seguridad informática en redes alámbricas e inalámbricas por medio de un dispositivo Raspberry Pi 2 [4].
En este proyecto el principal objetivo era analizar los puertos desde el sistema embebido y proponer políticas de seguridad, a diferencia del presente proyecto donde se analizarán puertos, previamente simulados, en un dispositivo remoto.
- Virtualización de redes de computadoras para el monitoreo y análisis de información entrante no deseada [5].
La similitud es que se trata de identificar las vulnerabilidades, en dicho proyecto se realiza con el tráfico que entra a la red, mientras que el presente proyecto lo hace con el análisis de los puertos.
- Servidor HTTPS en un FPGA [6].
Este proyecto implementa en su totalidad un servidor HTTPS usando una tarjeta FPGA, es similar en la parte del servidor y el uso de un sistema embebido. Aunque el presente proyecto solo simula el servidor y utiliza una tarjeta Raspberry.

2.2. Tesis

- Administración de Puertos TCP/IP como prevención de ataques [7].
Esta tesis pretendía diseñar una aplicación para administrar puertos TCP/IP, la similitud es la concientización al usuario de un posible ataque. Aunque se diferencian en que se analizan los puertos ya existentes.
- Seguridad en Redes [8].
La principal similitud es identificar las vulnerabilidades, en dicha tesis lo identifica a nivel general y de forma local, a diferencia de la presente propuesta que se enfocará específicamente a los puertos de red con un análisis remoto.

2.3. Software

- Simulador Red [9].
Este software puede simular puertos de red, y servidores de aplicaciones. La similitud es la posibilidad de simular algún puerto proporcionando un servicio, pero este programa también tiene herramientas para simular tramas Ethernet y peticiones ping.

3. Justificación

Actualmente, existen diversos softwares para levantar servicios de red específicos, por ejemplo, HTTP Apache para el servicio mencionado en su nombre. Estas aplicaciones consumen recursos de hardware para poder ejecutarse. Así que la creación de una herramienta que simule múltiples servicios de red, ayudará a mitigar este consumo ya que evitará la instalación de la aplicación correspondiente a cada servicio. Con la reducción de recursos se podrá hacer uso de un sistema embebido para que esta herramienta sea portable y ligera, evitando la necesidad de utilizar una computadora de grandes dimensiones.

Además, pretende hacer más didáctica la enseñanza de Seguridad en Redes al momento de identificar puertos y servicios.

4. Objetivos

4.1. Objetivo General

Diseñar un programa que implemente módulos con la finalidad de simular puertos y servicios de red usando un sistema embebido.

4.2. Objetivos Específicos

- Diseñar e implementar el módulo para simular el servicio HTTP.
- Diseñar e implementar el módulo para simular el servicio SMTP.
- Diseñar e implementar el módulo para simular el servicio POP3.
- Diseñar e implementar el módulo para simular el servicio FTP.
- Diseñar e implementar el módulo para simular el servicio DNS.
- Diseñar e implementar el módulo para simular el servicio IRC.
- Diseñar e implementar el módulo para simular el servicio IMAP.
- Diseñar e implementar el módulo para simular el servicio Syslog.
- Diseñar e implementar el módulo para simular el servicio SNMP.
- Diseñar e implementar el módulo para simular el servicio DHCP.

5. Marco Teórico

5.1 Raspberry Pi 3

Es una computadora del tamaño de una tarjeta de crédito que se conecta a una pantalla, teclado y mouse, se puede utilizar para proyectos de electrónica o para cumplir ciertas funciones que realiza una computadora de escritorio [1]. Está construida con un hardware limitado debido a que su fin son proyectos de propósito específico, sus características completas se pueden consultar en línea [2].



Figura 1.1 Sistema Embebido Raspberry Pi 3

5.2 Kali Linux

Es una distribución de Linux basada en Debian¹ destinada a pruebas avanzadas de penetración y auditoría de seguridad. Kali contiene varios cientos de herramientas destinadas a diversas tareas de seguridad de la información, como pruebas de penetración, análisis forense e ingeniería inversa. Kali Linux está diseñado, financiado y mantenido por *Offensive Security*, una empresa líder en capacitación en seguridad de la información [3].

5.3 Puerto de Sockets

Un puerto es la interfaz utilizada para establecer una comunicación a través de la red. Existen puertos denominados “bien conocidos”, que son reservados y utilizados por el sistema operativo, son los puertos inferiores al 1024. También se encuentran los puertos registrados, los cuales pueden ser usados por cualquier aplicación, están en el rango del 1024 al 49151. Además de los anteriores, hay puertos dinámicos o privados, estos no tienen ningún propósito específico, van del 49152 hasta el 65535.

5.4 Escaneo de puertos

Es la acción en la cual se analiza el estado de los puertos de un dispositivo conectado a la red, esto se realiza a través de herramientas diseñadas para este propósito. Su

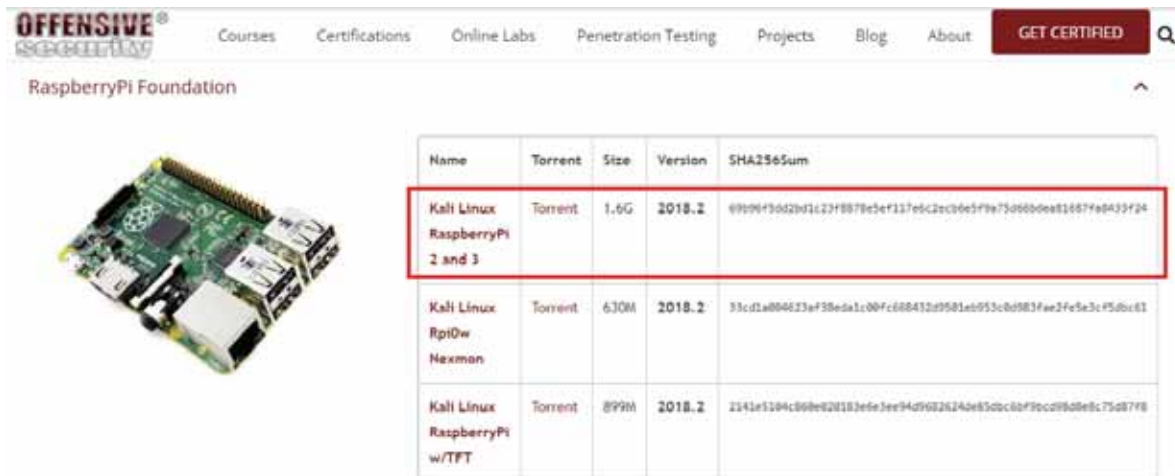
¹ Debian, es un sistema operativo libre que usa Linux como núcleo.

propósito es captar qué servicios “bien conocidos” está ejecutando el dispositivo, además de las posibles vulnerabilidades de acuerdo con los puertos identificados.

6. Desarrollo del Proyecto

6.1. Instalación y configuración del Sistema Operativo en la Raspberry Pi 3.

Primero se tiene que descargar la imagen ISO² de Kali Linux en la página oficial³ y seleccionar la opción adecuada como se puede observar en la *Figura 2.1*.



Name	Torrent	Size	Version	SHA256Sum
Kali Linux RaspberryPi 2 and 3	Torrent	1.6G	2018.2	69109f30d2bd1c22f8878e5e1137e6c2ec36e5f9e75066deat1667fe843f24
Kali Linux Rpi0w Nexman	Torrent	630M	2018.2	33cd1a804e23ef38eda1c00fc688432d9581e6053c8d883fae2fe5e3c75d6c81
Kali Linux RaspberryPi w/TFT	Torrent	899M	2018.2	2141e5184c868e92f183e6e3ee94d962624aeb50c6b0f9cc09b0e8c75d87f8

Figura 2.1 Imagen ISO para Raspberry Pi 3

Terminada la descarga de la imagen, se tiene que seguir el tutorial *Installing Operating System Images*, que se encuentra en la página oficial de Raspberry⁴, para la instalación de Kali Linux en una memoria SD.

Una vez que se tiene lo anterior listo, basta iniciar sesión con las credenciales predeterminadas, *User: root, Password: toor*, para poder trabajar. Cabe mencionar que el sistema operativo ya tiene instalada una versión 2.7.x de Python.

6.2. Estructura del script

El script está escrito en el lenguaje de programación Python en su versión 2.7.x, interactúa con diversos módulos y archivos, para llevar a cabo sus funciones, como se puede observar en la *Figura 3.1*.

² Imagen ISO se refiere a un archivo que contiene una copia exacta de un sistema de archivos.

³ <https://www.offensive-security.com/kali-linux-arm-images>

⁴ <https://www.raspberrypi.org/documentation/installation/installing-images/README.md>

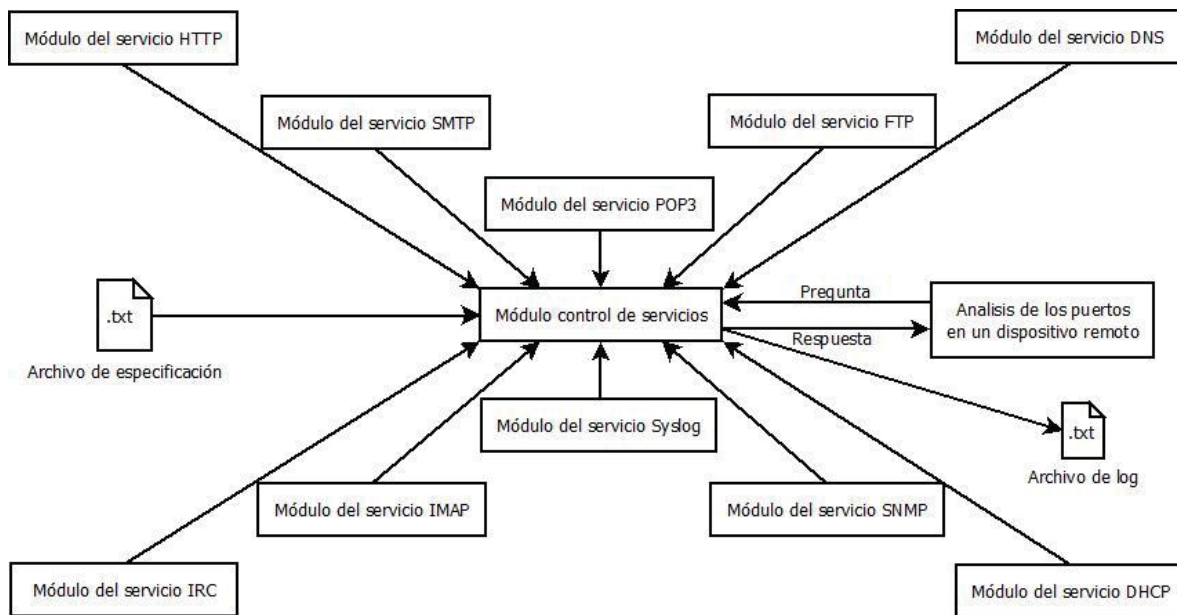


Figura 3.1 Módulos del script.

6.3. Módulo de servicio HTTP

Se encarga de abrir un socket TCP en el puerto 80, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente cierra la conexión. Además, el fingerprint enviado se registrará en un archivo (log.txt). En la *Figura 3.2* se observa parte del código del módulo antes descrito.

```
TCP_IP = ip_address
TCP_PORT = 80
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'HTTP Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data HTTP:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----HTTP fingerprint---- \x0d\x0a" + str(banner))
conn.close()
```

Figura 3.2 Código del módulo del servicio HTTP

6.4. Módulo de servicio SMTP

Se encarga de abrir un socket TCP en el puerto 25, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente

cierra la conexión. Además, el fingerprint enviado se registra en un archivo (log.txt). En la *Figura 3.3* se observa parte del código del módulo antes descrito.

```
TCP_IP = ip_address
TCP_PORT = 25
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'SMTP Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data SMTP:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----SMTP fingerprint---- \x0d\x0a" + str(banner))
conn.close()
```

Figura 3.3 Código del módulo del servicio SMTP

6.5. Módulo de servicio POP3

Se encarga de abrir un socket TCP en el puerto 110, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente cierra la conexión. Además, el fingerprint enviado se registra en un archivo (log.txt). En la *Figura 3.4* se observa parte del código del módulo antes descrito.

```
TCP_IP = ip_address
TCP_PORT = 110
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'POP3 Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data POP3:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----POP3 fingerprint---- \x0d\x0a" + str(banner))
conn.close()
```

Figura 3.4 Código del módulo del servicio POP3

6.6. Módulo de servicio FTP

Se encarga de abrir un socket TCP en el puerto 21, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente cierra la conexión. Además, el fingerprint enviado se registra en un archivo (log.txt). En la *Figura 3.5* se observa parte del código del módulo antes descrito.

```

TCP_IP = ip_address
TCP_PORT = 21
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'FTP Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data FTP:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----FTP fingerprint---- \x0d\x0a" + str(banner))
conn.close()

```

Figura 3.5 Código del módulo del servicio FTP

6.7. Módulo de servicio DNS

Se encarga de abrir un socket TCP en el puerto 53, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente cierra la conexión. Además, el fingerprint enviado se registra en un archivo (log.txt). En la *Figura 3.6* se observa parte del código del módulo antes descrito.

```

TCP_IP = ip_address
TCP_PORT = 53
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'DNS Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data DNS:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----DNS fingerprint---- \x0d\x0a" + str(banner))
conn.close()

```

Figura 3.6 Código del módulo del servicio DNS

6.8. Módulo de servicio IRC

Se encarga de abrir un socket TCP en el puerto 6666, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente cierra la conexión. Además, el fingerprint enviado se registra en un archivo (log.txt). En la *Figura 3.7* se observa parte del código del módulo antes descrito.

```

TCP_IP = ip_address
TCP_PORT = 6666
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'IRC Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data IRC:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----IRC fingerprint---- \x0d\x0a" + str(banner))
conn.close()

```

Figura 3.7 Código del módulo del servicio IRC

6.9. Módulo de servicio IMAP

Se encarga de abrir un socket TCP en el puerto 143, el cual está a la escucha de una petición de conexión, una vez que acepta la petición recibe las preguntas correspondientes, envía un *fingerprint* para poder ser identificado y posteriormente cierra la conexión. Además, el fingerprint enviado se registra en un archivo (log.txt). En la *Figura 3.8* se observa parte del código del módulo antes descrito.

```

TCP_IP = ip_address
TCP_PORT = 143
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'IMAP Connection address: ', addr

while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    #print 'Received data IMAP:', data
    conn.send(banner)
    archivol.write("\x0d\x0a----IMAP fingerprint---- \x0d\x0a" + str(banner))
conn.close()

```

Figura 3.8 Código del módulo del servicio IMAP

6.10. Módulo de servicio Syslog

Se encarga de abrir un socket UDP en el puerto 514, el cual está a la escucha y recibe las preguntas correspondientes para poder ser identificado. En la *Figura 3.9* se observa parte del código del módulo antes descrito.


```

UDP_IP = ip_address
UDP_PORT = 514

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024)
    #if not data: break
    print 'SYSLOG Received data from (' + addr + '): ', data

```

Figura 3.9 Código del módulo del servicio Syslog

6.11. Módulo de servicio SNMP

Se encarga de abrir un socket UDP en el puerto 161, el cual está a la escucha y recibe las preguntas correspondientes para poder ser identificado. En la *Figura 3.10* se observa parte del código del módulo antes descrito.

```

UDP_IP = ip_address
UDP_PORT = 161

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024)
    #if not data: break
    print 'SNMP Received data from (' + addr + '): ', data

```

Figura 3.10 Código del módulo del servicio SNMP

6.12. Módulo de servicio DHCP

Se encarga de abrir un socket UDP en el puerto 67, el cual está a la escucha y recibe las preguntas correspondientes para poder ser identificado. En la *Figura 3.11* se observa parte del código del módulo antes descrito.

```

UDP_IP = ip_address
UDP_PORT = 67

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024)
    #if not data: break
    print 'DHCP Received data from (' + addr + '): ', data

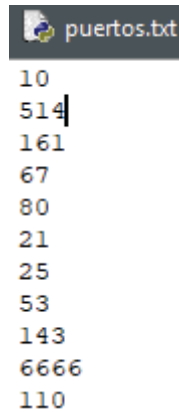
```

Figura 3.11 Código del módulo del servicio DHCP

6.13. Módulo de control de servicios

Este módulo es el encargado de gestionar la apertura de los puertos de la siguiente forma:

- Se realiza la lectura del archivo “puertos.txt”, para obtener la cantidad de puertos a simular y el número de puerto correspondiente. En la *Figura 4.1* se muestra una configuración del archivo antes mencionado.



```
puertos.txt
10
514
161
67
80
21
25
53
143
6666
110
```

Figura 4.1 Configuración del archivo puertos.txt

- Terminada la lectura, se procede a la validación de cada uno de los puertos y a la posterior ejecución del módulo correspondiente.

Es importante mencionar que el archivo de configuración está diseñado para leer primero los posibles tres puertos UDP y después los puertos TCP, es decir, en el archivo se deben especificar primero los puertos UDP.

6.14. Obtención de *fingerprints*

Como ya se mencionó, los módulos que simulan puertos TCP envían *fingerprints* para ser reconocidos al momento de que se realiza el escaneo. Para obtenerlos, se debe instalar cada uno de los servicios TCP en un dispositivo externo con sistema Linux, después se realiza el escaneo con Nmap y con Wireshark se captura el tráfico de la red el tiempo que se tarde en ejecutarse.

Ya que se tiene la captura del tráfico, se buscan los *fingerprints* para cada puerto. Esto se observa en las *Figuras 5.1 – 5.7*.

No.	Time	Source	Destination	Protocol	Length	Info
1898	23.372000	192.168.9.37	192.168.9.92	HTTP	246	HTTP/1.1 200 OK
1888	23.220267	192.168.9.37	192.168.9.92	HTTP	246	HTTP/1.1 200 OK
1875	23.068592	192.168.9.37	192.168.9.92	HTTP	522	HTTP/1.1 404 Not Found (text/html)
1873	23.068383	192.168.9.37	192.168.9.92	HTTP	246	HTTP/1.1 200 OK
1852	22.878419	192.168.9.37	192.168.9.92	HTTP	246	HTTP/1.1 200 OK
1781	22.799340	192.168.9.37	192.168.9.92	HTTP	520	HTTP/1.1 404 Not Found (text/html)
1779	22.799115	192.168.9.37	192.168.9.92	HTTP	520	HTTP/1.1 404 Not Found (text/html)

> Frame 1873: 246 bytes on wire (1968 bits), 246 bytes captured (1968 bits) on interface 0
 > Ethernet II, Src: PcsCompu_11:15:60 (08:00:27:11:15:60), Dst: Apple_64:39:5b (3c:07:54:64:39:5b)
 > Internet Protocol Version 4, Src: 192.168.9.37, Dst: 192.168.9.92
 > Transmission Control Protocol, Src Port: 80, Dst Port: 20367, Seq: 1, Ack: 212, Len: 191
 > Hypertext Transfer Protocol
 > VSS-Monitoring ethernet trailer, Source Port: 0

```

0000 3c 07 54 64 39 5b 08 00 27 11 15 60 08 00 45 00 <-Td9[... '...'E-
0010 00 e7 9c e4 40 00 40 06 09 5b c0 a8 09 25 c0 a8 ...@.@ [...'%.
0020 09 5c 00 50 4f 8f ee 08 85 40 1e 92 41 5b 50 18 \..PO...@..A[P.
0030 00 ed 48 e2 00 00 48 54 54 50 2f 31 2e 31 20 32 ..H...HT TP/1.1 2
0040 30 30 20 4f 4b 0d 0a 44 61 74 65 3a 20 54 75 65 00 OK D ate: Tue
0050 2c 20 31 33 20 4d 61 72 20 32 30 31 38 20 32 30 , 13 Mar 2018 20
0060 3a 35 36 3a 34 31 20 47 4d 54 0d 0a 53 65 72 76 :56:41 G MT Serv
0070 65 72 3a 20 41 70 61 63 68 65 2f 32 2e 34 2e 32 er: Apac he/2.4.2
0080 35 20 28 44 65 62 69 61 6e 29 0d 0a 41 6c 6c 6f 5 (Debia n) Allo
0090 77 3a 20 30 0d 0a 43 6f 6e 6e 65 63 74 69 6f w: OPTIO NS,HEAD,
00a0 48 45 41 44 2c 47 45 54 2c 48 45 41 44 2c 50 4f HEAD,GET ,HEAD,PO
00b0 53 54 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 ST Cont ent-Leng
00c0 74 68 3a 20 30 0d 0a 43 6f 6e 6e 65 63 74 69 6f th: 0 C connectio
00d0 6e 3a 20 63 6c 6f 73 65 0d 0a 43 6f 6e 74 65 6e n: close ..Conten
00e0 74 2d 54 79 70 65 3a 20 74 65 78 74 2f 68 74 6d t-Type: text/htm
00f0 6c 0d 0a 0d 0a 00 l...
    
```

Figura 5.1 Fingerprint servicio HTTP

No.	Time	Source	Destination	Protocol	Length	Info
1859	22.881338	192.168.9.37	192.168.9.92	SMTP	96	S: 502 5.5.2 Error: command not recognized
1858	22.880647	192.168.9.37	192.168.9.92	SMTP	84	S: 220 2.0.0 Ready to start TLS
1854	22.878581	192.168.9.37	192.168.9.92	SMTP	84	S: 220 2.0.0 Ready to start TLS
1811	22.823760	192.168.9.37	192.168.9.92	SMTP	202	S: 250 Fredo.CADI 250 PIPELINING 250 SIZE
1808	22.823561	192.168.9.37	192.168.9.92	SMTP	202	S: 250 Fredo.CADI 250 PIPELINING 250 SIZE
1803	22.822829	192.168.9.37	192.168.9.92	SMTP	202	S: 250 Fredo.CADI 250 PIPELINING 250 SIZE
1798	22.820928	192.168.9.37	192.168.9.92	SMTP	98	S: 220 Fredo.CADI ESMTP Postfix (Debian/GNU)

> Frame 1798: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
 > Ethernet II, Src: PcsCompu_11:15:60 (08:00:27:11:15:60), Dst: Apple_64:39:5b (3c:07:54:64:39:5b)
 > Internet Protocol Version 4, Src: 192.168.9.37, Dst: 192.168.9.92
 > Transmission Control Protocol, Src Port: 25, Dst Port: 20351, Seq: 1, Ack: 1, Len: 43
 > Simple Mail Transfer Protocol
 > VSS-Monitoring ethernet trailer, Source Port: 0

```

0000 3c 07 54 64 39 5b 08 00 27 11 15 60 08 00 45 00 <-Td9[... '...'E-
0010 00 53 bb e4 40 00 40 06 ea ee c0 a8 09 25 c0 a8 ..S..@.@ .....%.
0020 09 5c 00 19 4f 7f c9 5d a2 ba 68 99 8e 86 50 18 \..O..] ..h...P.
0030 00 e5 54 a4 00 00 32 32 30 20 46 72 65 64 6f 2e ..T...22 0 Fredo.
0040 43 41 44 49 20 45 53 4d 54 50 20 50 6f 73 74 66 CADI ESM TP Postf
0050 69 78 20 28 44 65 62 69 61 6e 2f 47 4e 55 29 0d ix (Debi an/GNU)
0060 0a 00 .
    
```

Figura 5.2 Fingerprint servicio SMTP

No.	Time	Source	Destination	Protocol	Length	Info
2084	17.359597	10.0.1.67	10.0.1.69	POP	74	S: +OK Dovecot ready.
2159	19.962992	10.0.1.67	10.0.1.69	POP	74	S: +OK Dovecot ready.
2165	19.978271	10.0.1.67	10.0.1.69	POP	74	S: +OK Dovecot ready.
2166	19.978561	10.0.1.67	10.0.1.69	POP	74	S: +OK Dovecot ready.
2171	19.996820	10.0.1.69	10.0.1.67	POP	60	C: STLS
2172	19.996952	10.0.1.69	10.0.1.67	POP	60	C: CAPA
2173	19.997072	10.0.1.69	10.0.1.67	POP	60	C: STLS

> Frame 2084: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 > Ethernet II, Src: Tp-LinkT_17:6c:0c (c4:e9:84:17:6c:0c), Dst: HonHaiPr_f5:e3:41 (9c:d2:1e:f5:e3:41)
 > Internet Protocol Version 4, Src: 10.0.1.67, Dst: 10.0.1.69
 > Transmission Control Protocol, Src Port: 110, Dst Port: 11534, Seq: 1, Ack: 1, Len: 20
 > Post Office Protocol

```

0000  9c d2 1e f5 e3 41 c4 e9 84 17 6c 0c 08 00 45 00  . . . . A . . . . 1 . . . . E .
0010  00 3c 13 81 40 00 40 06 10 b4 0a 00 01 43 0a 00  < . . @ . @ . . . . . C . .
0020  01 45 00 0e 2d 0e 99 9a 7a c6 0e 25 48 4e 50 18  . E . n . . . . . z . . % H N P .
0030  00 e5 9d 25 00 00 2b 4f 4b 20 4a 6f 76 65 63 6f  . . % . . + O K Doveco
0040  74 20 72 65 61 64 79 2e 0d 0a                      t ready. .
    
```

Figura 5.3 Fingerprint servicio POP3

No.	Time	Source	Destination	Protocol	Length	Info
1900	23.372396	192.168.9.37	192.168.9.92	FTP	68	Response: 221 Goodbye.
1891	23.270910	192.168.9.37	192.168.9.92	FTP	80	Response: 500 AUTH not understood
1886	23.176080	192.168.9.37	192.168.9.92	FTP	112	Response: 220 ProFTPD 1.3.5b Server (Debian) [::ffff:192.168.9.37]
1857	22.880572	192.168.9.37	192.168.9.92	FTP	76	Response: 530 Login incorrect.
1855	22.879248	192.168.9.37	192.168.9.92	FTP	68	Response: 221 Goodbye.
1850	22.877797	192.168.9.37	192.168.9.92	FTP	76	Response: 530 Login incorrect.
1810	22.823688	192.168.9.37	192.168.9.92	FTP	92	Response: 331 Password required for anonymous

> Frame 1886: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
 > Ethernet II, Src: PcsCompu_11:15:60 (08:00:27:11:15:60), Dst: Apple_64:39:5b (3c:07:54:64:39:5b)
 > Internet Protocol Version 4, Src: 192.168.9.37, Dst: 192.168.9.92
 > Transmission Control Protocol, Src Port: 21, Dst Port: 20370, Seq: 1, Ack: 1, Len: 58
 > File Transfer Protocol (FTP)
 [Current working directory:]

```

0000  3c 07 54 64 39 5b 08 00 27 11 15 60 08 00 45 00  < . T d 9 [ . . . . . E .
0010  00 62 b9 53 40 00 40 06 ed 70 c0 a8 09 25 c0 a8  . b . 5 @ . @ . p . . % .
0020  09 5c 00 15 4f 92 7f 8d a0 ef 26 11 f7 4e 50 18  . \ . 0 . . . . . 8 . N P .
0030  00 e5 c5 9d 00 00 32 32 30 20 50 72 6f 46 54 50  . . . . . 3 2 0 P r o F T P
0040  44 20 31 2e 33 2e 35 62 20 33 65 72 76 65 72 20  0 1 . 3 . 5 b S e r v e r
0050  20 44 65 62 69 61 6e 29 20 5b 3a 3a 66 66 66 66  ( D e b i a n ) [ : : f f f f
0060  3a 31 39 32 2e 31 36 30 2e 39 2e 33 37 5d 0d 0a  : 1 9 2 . 1 6 8 . 9 . 3 7 ]
    
```

Figura 5.4 Fingerprint servicio FTP

No.	Time	Source	Destination	Protocol	Length	Info
1539	10.937468	192.168.9.37	192.168.9.92	DNS	130	Standard query response 0x0006 TXT version.bind TXT NS version.bind
46	4.860871	192.168.9.37	192.168.9.92	TCP	60	53 → 34923 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
1033	4.936551	192.168.9.37	192.168.9.92	TCP	66	53 → 20228 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM
1507	5.071464	192.168.9.37	192.168.9.92	TCP	60	[TCP Retransmission] 53 → 34923 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
1515	7.807419	192.168.9.37	192.168.9.92	TCP	60	[TCP Retransmission] 53 → 34923 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
1537	10.936925	192.168.9.37	192.168.9.92	TCP	60	53 → 20228 [ACK] Seq=1 Ack=33 Win=29312 Len=0
1540	10.938531	192.168.9.37	192.168.9.92	TCP	60	53 → 20228 [FIN, ACK] Seq=76 Ack=34 Win=29312 Len=0

> Frame 1539: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
 > Ethernet II, Src: PcsCompu_11:15:60 (08:00:27:11:15:60), Dst: Apple_64:39:5b (3c:07:54:64:39:5b)
 > Internet Protocol Version 4, Src: 192.168.9.37, Dst: 192.168.9.92
 > Transmission Control Protocol, Src Port: 53, Dst Port: 20228, Seq: 1, Ack: 33, Len: 75
 > Domain Name System (response)
 > VSS-Monitoring ethernet trailer, Source Port: 0

```

0000 3c 07 54 64 39 5b 08 00 27 11 15 60 08 00 45 10 <-Td9[... '...'E-
0010 00 73 68 c0 40 00 40 06 3d f3 c0 a8 09 25 c0 a8 sh-@ @ T...%--
0020 09 5c 00 35 4f 04 76 a3 6f 51 c3 6a 6c 2e 50 18 \-50 v: cQ }l.P-
0030 00 e5 d2 59 00 00 00 49 00 06 85 00 00 01 00 01 ...Y: I
0040 00 01 00 00 07 76 65 72 73 69 6f 6e 04 62 69 6e .....sh ybrid.d
0050 64 00 00 10 00 03 c0 0c 00 10 00 03 00 00 00 00 ebian.lo cal NOTI
0060 00 11 10 39 2e 31 30 2e 33 2d 50 34 2d 44 65 62 CE * : ** * Lookin
0070 69 61 6e c0 0c 00 02 00 03 00 00 00 00 02 c0 g up you r hostna
0080 0e 00 me
    
```

Figura 5.5 Fingerprint servicio DNS

No.	Time	Source	Destination	Protocol	Length	Info
2249	71.498985	192.168.9.37	192.168.9.92	TCP	60	6666 → 20386 [ACK] Seq=1 Ack=61 Win=29312 Len=0
2250	71.499187	192.168.9.37	192.168.9.92	TCP	116	6666 → 20386 [PSH, ACK] Seq=1 Ack=61 Win=29312 Len=62
2251	71.499256	192.168.9.37	192.168.9.92	TCP	106	6666 → 20386 [PSH, ACK] Seq=63 Ack=61 Win=29312 Len=52
2254	71.539703	192.168.9.37	192.168.9.92	TCP	60	6666 → 20385 [ACK] Seq=183 Ack=25 Win=29312 Len=0
2257	72.000898	192.168.9.37	192.168.9.92	TCP	110	6666 → 20383 [PSH, ACK] Seq=183 Ack=13 Win=29312 Len=55
2258	72.001067	192.168.9.37	192.168.9.92	TCP	126	6666 → 20383 [PSH, ACK] Seq=238 Ack=13 Win=29312 Len=71
2259	72.001110	192.168.9.37	192.168.9.92	TCP	60	6666 → 20383 [FIN, ACK] Seq=309 Ack=13 Win=29312 Len=0

> Frame 2250: 116 bytes on wire (928 bits), 116 bytes captured (928 bits) on interface 0
 > Ethernet II, Src: PcsCompu_11:15:60 (08:00:27:11:15:60), Dst: Apple_64:39:5b (3c:07:54:64:39:5b)
 > Internet Protocol Version 4, Src: 192.168.9.37, Dst: 192.168.9.92
 > Transmission Control Protocol, Src Port: 6666, Dst Port: 20386, Seq: 1, Ack: 61, Len: 62
 > Data (62 bytes)

```

0000 3c 07 54 64 39 5b 08 00 27 11 15 60 08 00 45 10 <-Td9[... '...'E-
0010 00 66 51 fb 40 00 40 06 54 b5 c0 a8 09 25 c0 a8 -fQ @ @ T...%--
0020 09 5c 1a 0a 4f a2 67 3b 19 15 90 f9 33 2f 50 18 \-0 g: ...3/P-
0030 00 e5 ef d3 00 00 3a 08 79 62 72 69 64 38 2e 64 .....sh ybrid.d
0040 85 62 69 61 6e 2e 6c 6f 63 61 6c 20 4e 4f 54 49 ebian.lo cal NOTI
0050 43 45 20 2a 20 3a 2a 2a 2a 20 4c 6f 6f 6b 69 6e CE * : ** * Lookin
0060 67 20 75 70 20 79 6f 75 72 20 68 6f 73 74 6e 61 g up you r hostna
0070 6d 65 0d 0a me
    
```

Figura 5.6 Fingerprint servicio IRC

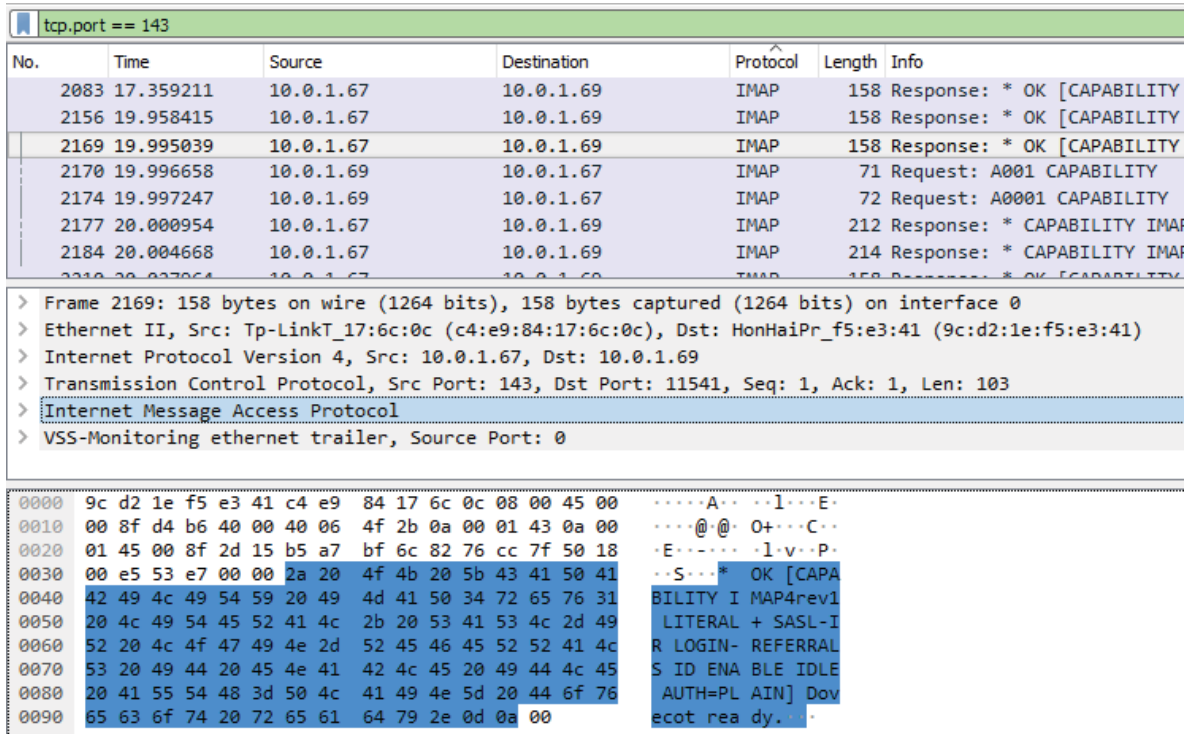


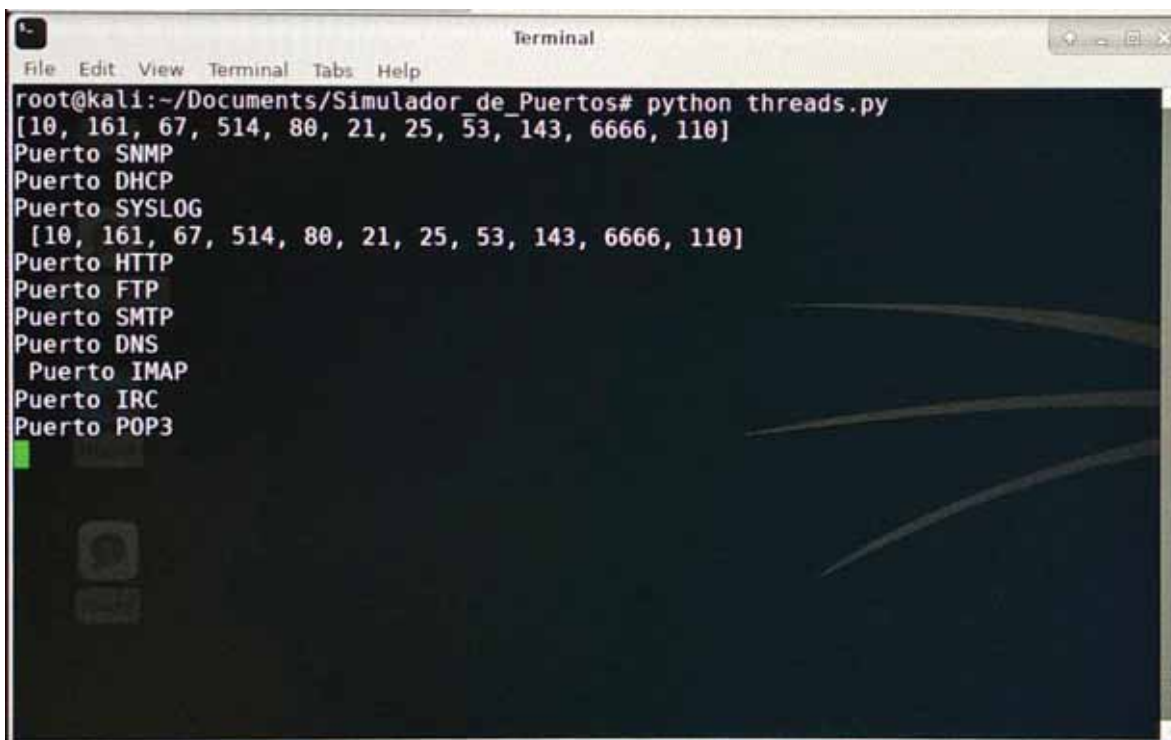
Figura 5.7 Fingerprint servicio IMAP

Cabe destacar que algunos caracteres de los *fingerprints* no fueron enviados como texto plano, si no en formato hexadecimal usando la notación `\x##`.

7. Resultados

Como resultado de lo descrito en el desarrollo del proyecto, se obtiene un script cuya ejecución y funcionamiento se explica a continuación.

Su ejecución en Linux se realiza desde modo superusuario (*root*), ya que, la apertura de los puertos esta restringida a este usuario, en el caso de Windows no es necesario tener permisos de administrador.



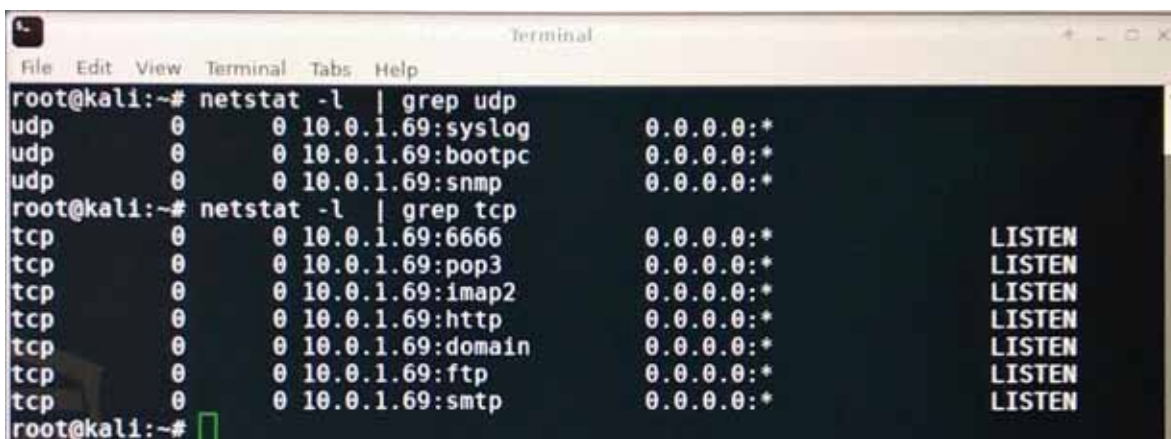
```

root@kali:~/Documents/Simulador de Puertos# python threads.py
[10, 161, 67, 514, 80, 21, 25, 53, 143, 6666, 110]
Puerto SNMP
Puerto DHCP
Puerto SYSLOG
[10, 161, 67, 514, 80, 21, 25, 53, 143, 6666, 110]
Puerto HTTP
Puerto FTP
Puerto SMTP
Puerto DNS
Puerto IMAP
Puerto IRC
Puerto POP3

```

Figura 6.1 Ejecución del script

Como se puede observar en la *Figura 6.1*, el script hace la validación de los números de puerto y ejecuta los módulos correspondientes, en este caso todos los posibles, se encuentra a la espera de una conexión, o bien, la realización de un escaneo por algún dispositivo en la red.



```

root@kali:~# netstat -l | grep udp
udp        0      0 10.0.1.69:syslog      0.0.0.0:*
udp        0      0 10.0.1.69:bootpc     0.0.0.0:*
udp        0      0 10.0.1.69:snmp       0.0.0.0:*
root@kali:~# netstat -l | grep tcp
tcp        0      0 10.0.1.69:6666       0.0.0.0:*   LISTEN
tcp        0      0 10.0.1.69:pop3       0.0.0.0:*   LISTEN
tcp        0      0 10.0.1.69:imap2      0.0.0.0:*   LISTEN
tcp        0      0 10.0.1.69:http       0.0.0.0:*   LISTEN
tcp        0      0 10.0.1.69:domain     0.0.0.0:*   LISTEN
tcp        0      0 10.0.1.69:ftp        0.0.0.0:*   LISTEN
tcp        0      0 10.0.1.69:smtp       0.0.0.0:*   LISTEN
root@kali:~#

```

Figura 6.2 Comando para verificar la apertura de puertos.

Nos cercioramos de que los puertos están abiertos utilizando el comando de Linux *netstat*, como se puede observar en la *Figura 6.2*, los siete puertos TCP se encuentra abiertos y en escucha, al igual que los tres puertos UDP.

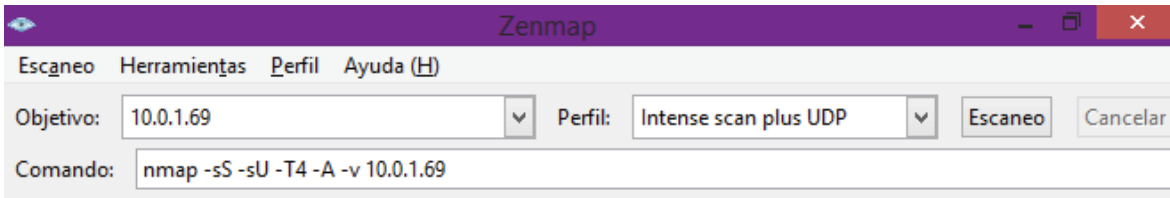


Figura 6.3 Ejecución del escaneo con Nmap

Con la herramienta Nmap, se realiza un escaneo de puertos. Como se puede observar en la *Figura 6.3*, se especifica que se realice un escaneo intenso con adición de puertos UDP, también se indica la dirección IP del dispositivo objetivo.

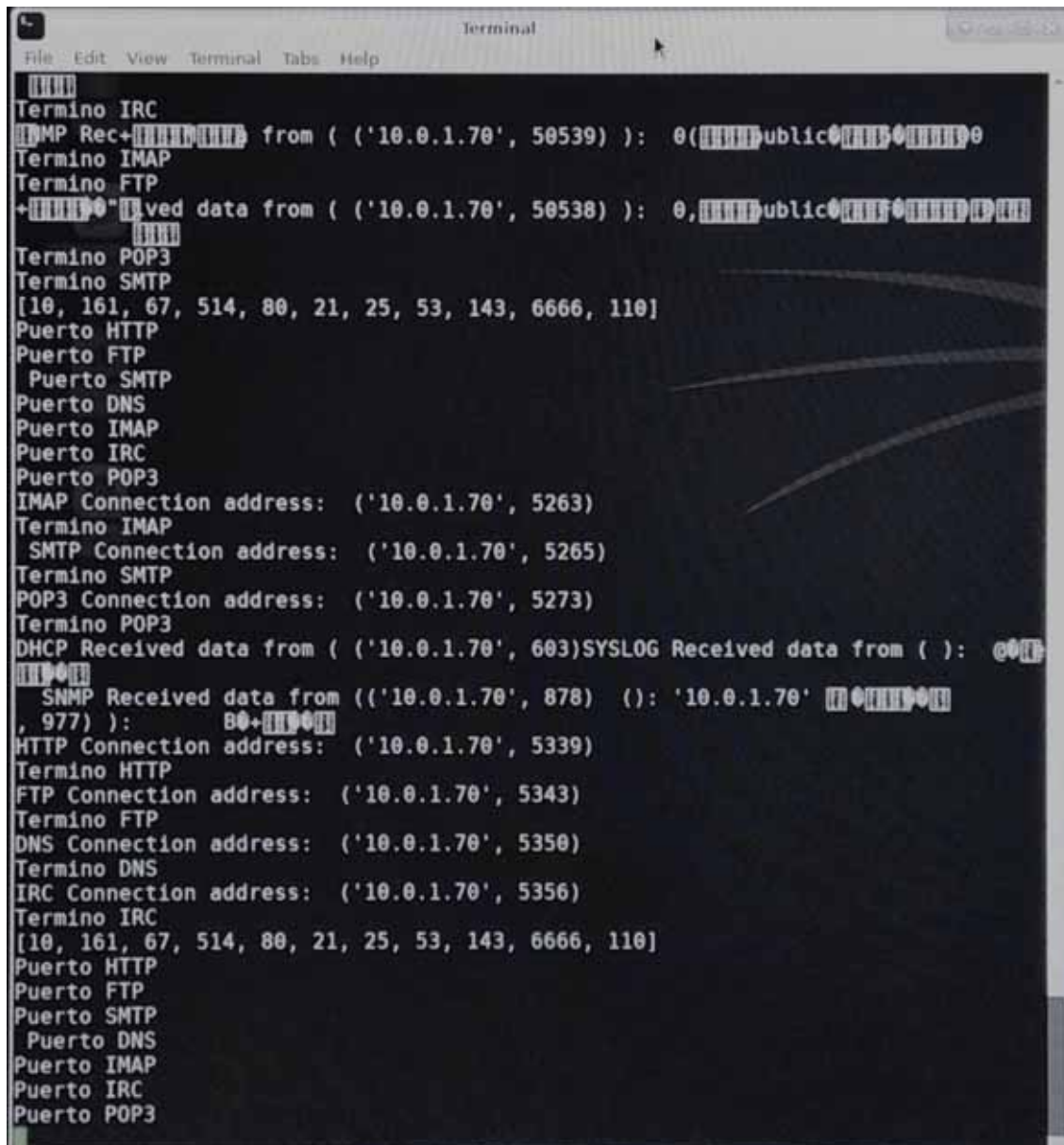


Figura 6.4 Salida de la terminal

En la *Figura 6.4* se observa que al realizar un escaneo el script muestra la información que recibió por los puertos UDP, indica cuando los servicios TCP empiezan y terminan, así como la dirección IP del dispositivo que se comunicó con ellos. Es importante mencionar que los servicios UDP nunca cierran su conexión, mientras que los TCP necesitan cerrar el socket para que otro dispositivo se pueda conectar.

Además, se necesita que terminen todos los servicios TCP para que vuelvan a estar disponibles para una nueva conexión, el script no termina por si solo es requerido que se cierra la ventana o se cierre el proceso.

8. Conclusiones

El escaneo de puertos en dispositivos conectados a la red se ha vuelto más frecuente debido a las vulnerabilidades que han salido a la luz en los últimos años. La importancia de saber realizar este tipo de análisis es muy alta, por lo que una herramienta para practicar es muy útil.

Con el script realizado se pueden diseñar escenarios para la identificación de puertos TCP y UDP, para su posterior análisis e investigación. Con la correcta aplicación de esta herramienta se puede lograr que los alumnos se interesen en la seguridad de redes, ya que se ejecuta en un dispositivo portátil y no hace falta la instalación de herramientas que consuman muchos recursos.

9. Bibliografía

- [1] Raspberry Pi Foundation, “What is a Raspberry Pi?”, <https://www.raspberrypi.org/help/faqs/#introWhatIs>.
- [2] Raspberry Pi Foundation, “Raspberry Pi 3 Model B Specifications,” <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>, 2016.
- [3] Offensive Security, “What is Kali Linux?”, <https://docs.kali.org/introduction/what-is-kali-linux>, 2017.
- [4] J. Sánchez Martínez, “Análisis de seguridad informática en redes alámbricas e inalámbricas por medio de un dispositivo Raspberry Pi 2”, Proyecto Terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2016.
- [5] A. Martínez Escobar y C. Servín García, “Virtualización de redes de computadoras para el monitoreo y análisis de información entrante no deseada”, Proyecto Terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2007.
- [6] A. Cadena Cervantes, “Servidor HTTPS en un FPGA”, Proyecto Terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.
- [7] H. G. Colula Márquez y J. Lama Gervacio, “Administración de Puertos TCP/IP como prevención de ataques”, Tesis de Ingeniería, Instituto Politécnico Nacional, México, 2011.
- [8] R. Bustamante Sánchez, “Seguridad en redes”, Tesis de Ingeniería, Universidad Autónoma del Estado de Hidalgo, Estado de Hidalgo.
- [9] SopiremInfo, “Simulador Red,” <http://www.sopireminfo.com/indexEs.html>.

10. Apéndices

Se anexa código fuente del script y archivos de configuración en un disco CD.