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

Tabla de Figuras

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 podernos 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 especifico, 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 Debian[1] destinada a pruebas avanzadas de penetración y auditoria 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 esta 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 ISO[2] de Kali Linux en la página oficial[3] y seleccionar la opción adecuada como se puede observar en la *Figura 2.1.*

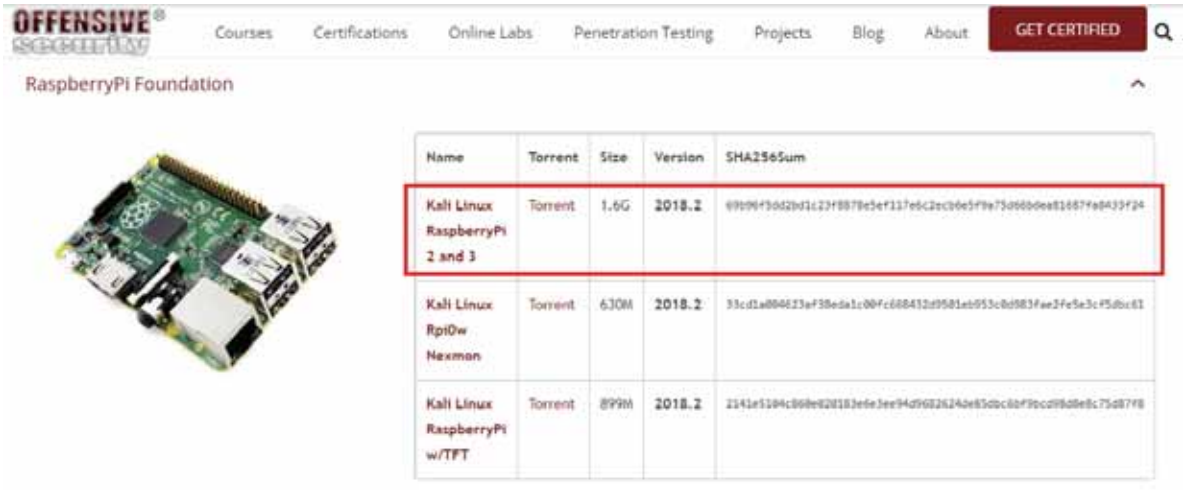

Figura 2.1 Imagen ISO para Raspberry Pi 3

Terminada la descarga de la imagen, se tiene que seguir el tutorial *Installing Operating System Images*, que se encuentra en la página oficial de Raspberry[4], 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

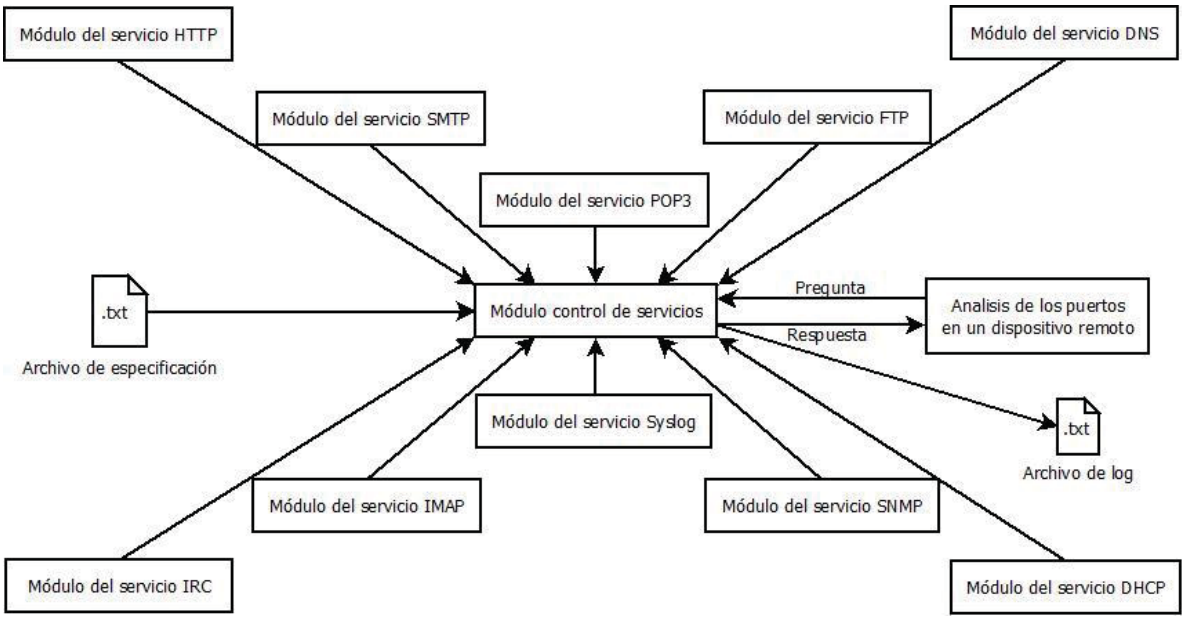

Figura 3.1 Módulos del script.

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, envia 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, envia 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.

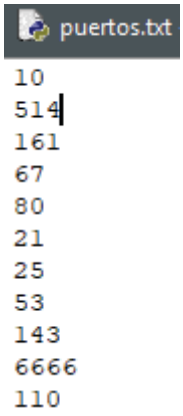

Figura 4.1 Configuración del archivo puertos.txt

- 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 esta 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 trafico 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.*

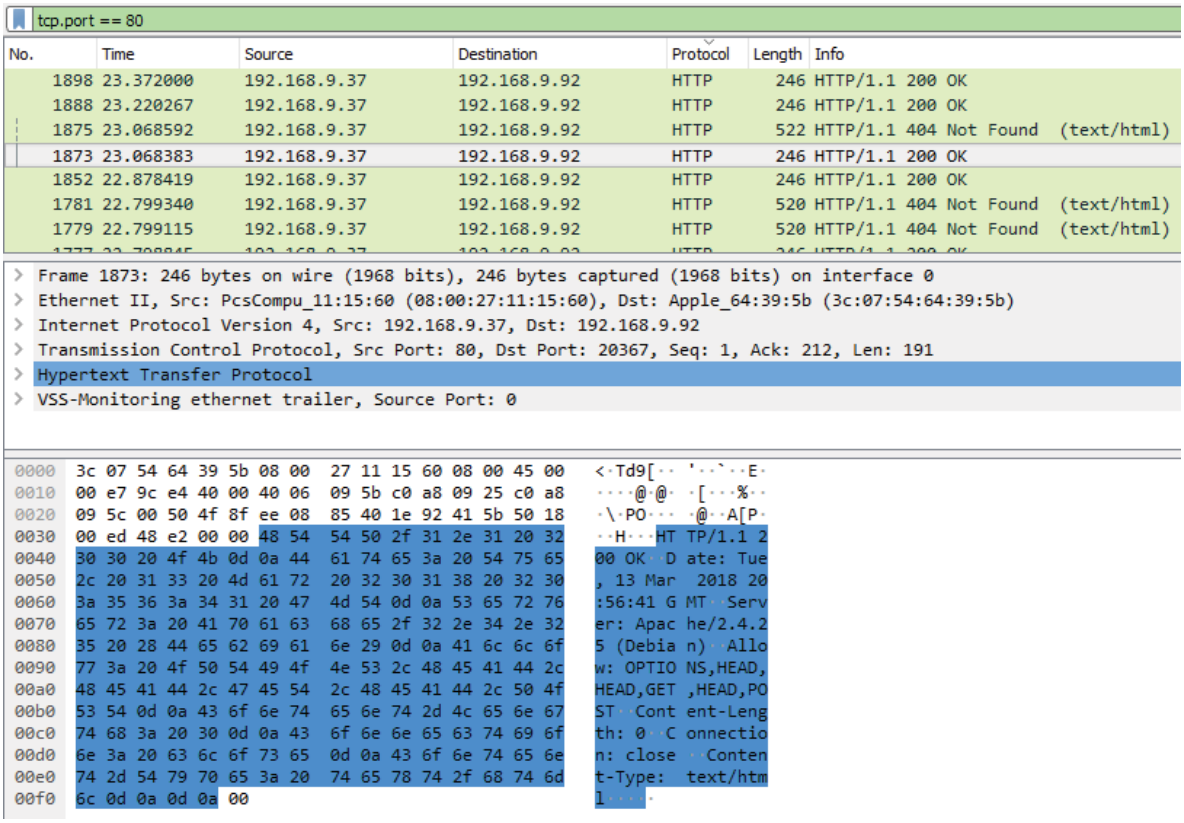

Figura 5.1 Fingerprint servicio HTTP

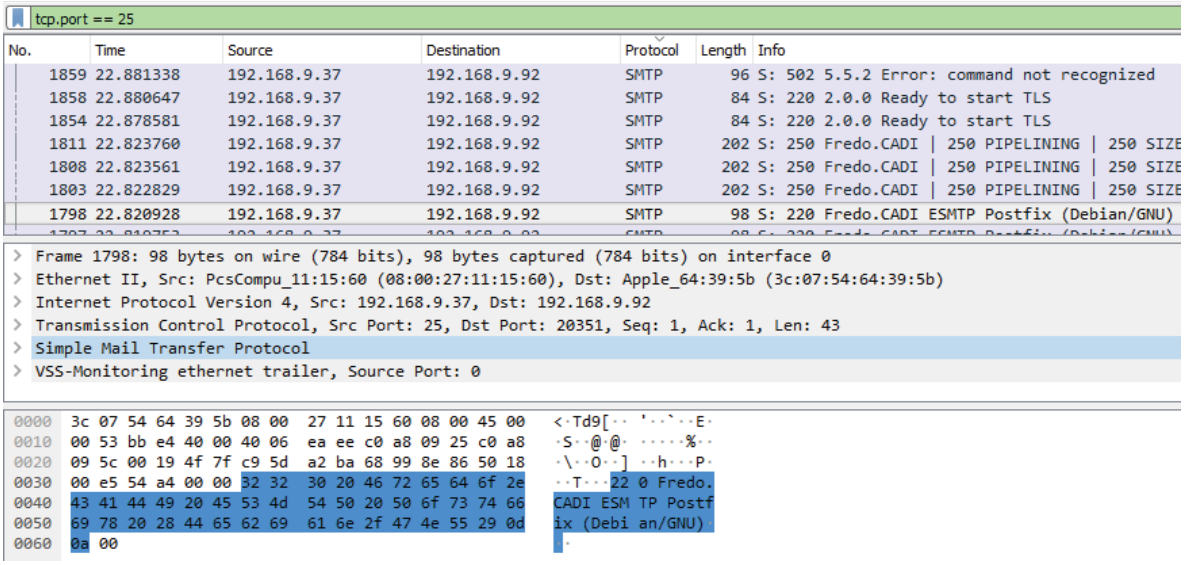

Figura 5.2 Fingerprint servicio SMTP

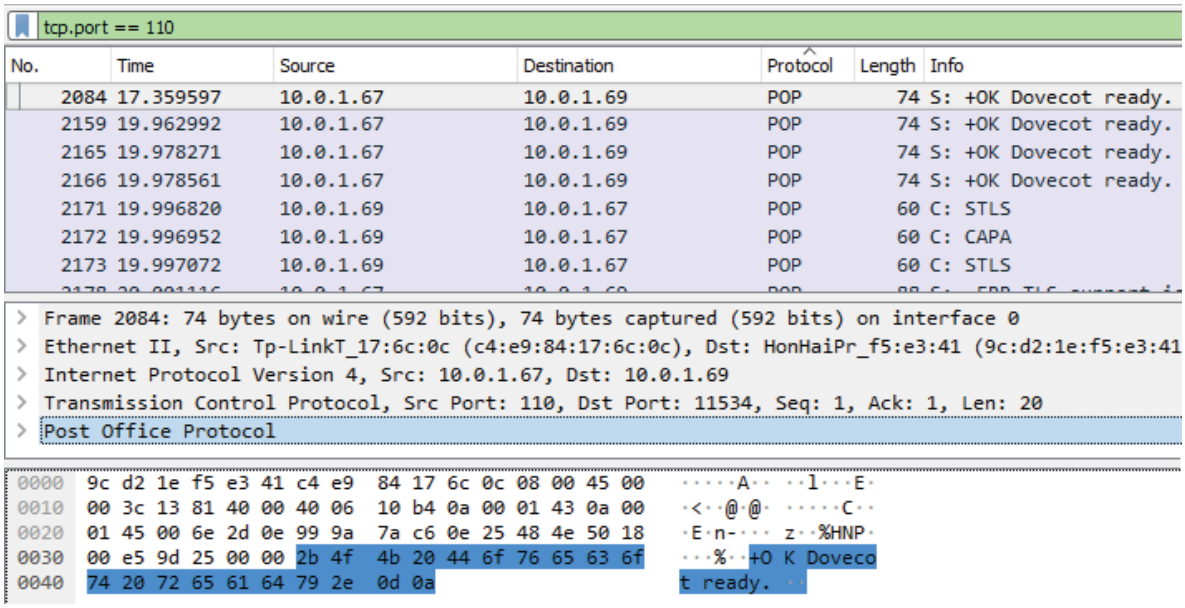

Figura 5.3 Fingerprint servicio POP3

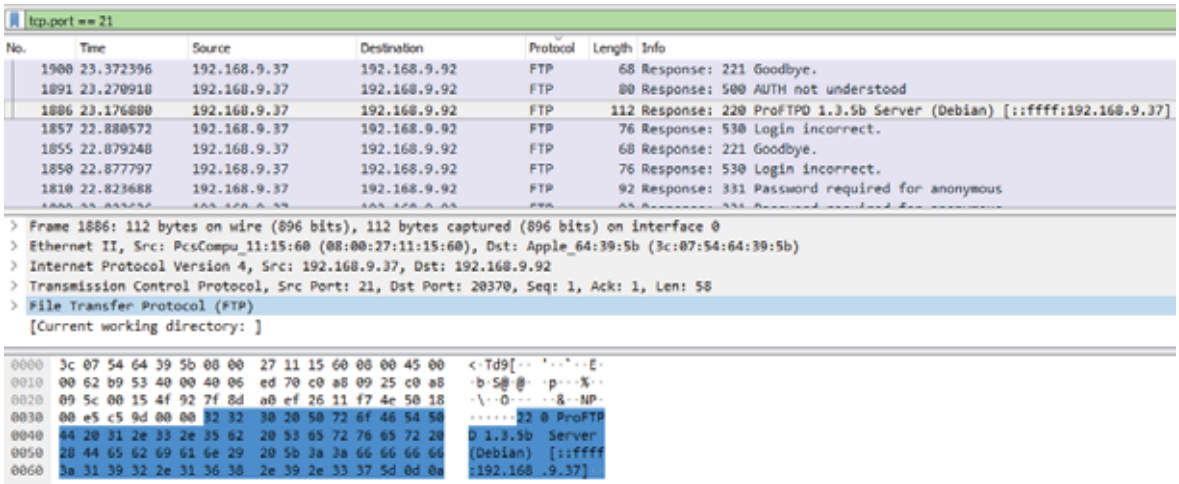

Figura 5.4 Fingerprint servicio FTP

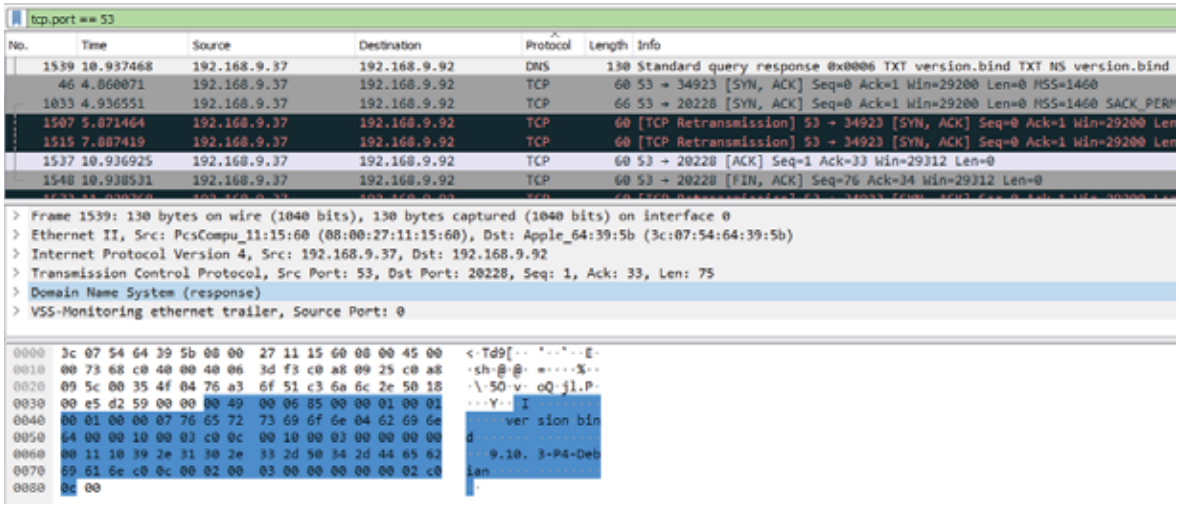

Figura 5.5 Fingerprint servicio DNS

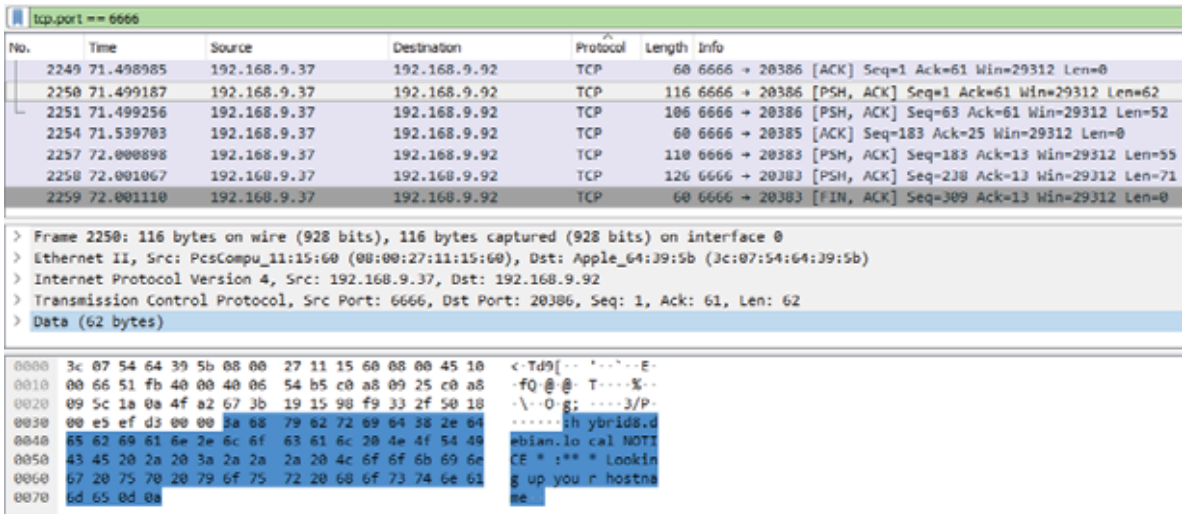

Figura 5.6 Fingerprint servicio IRC

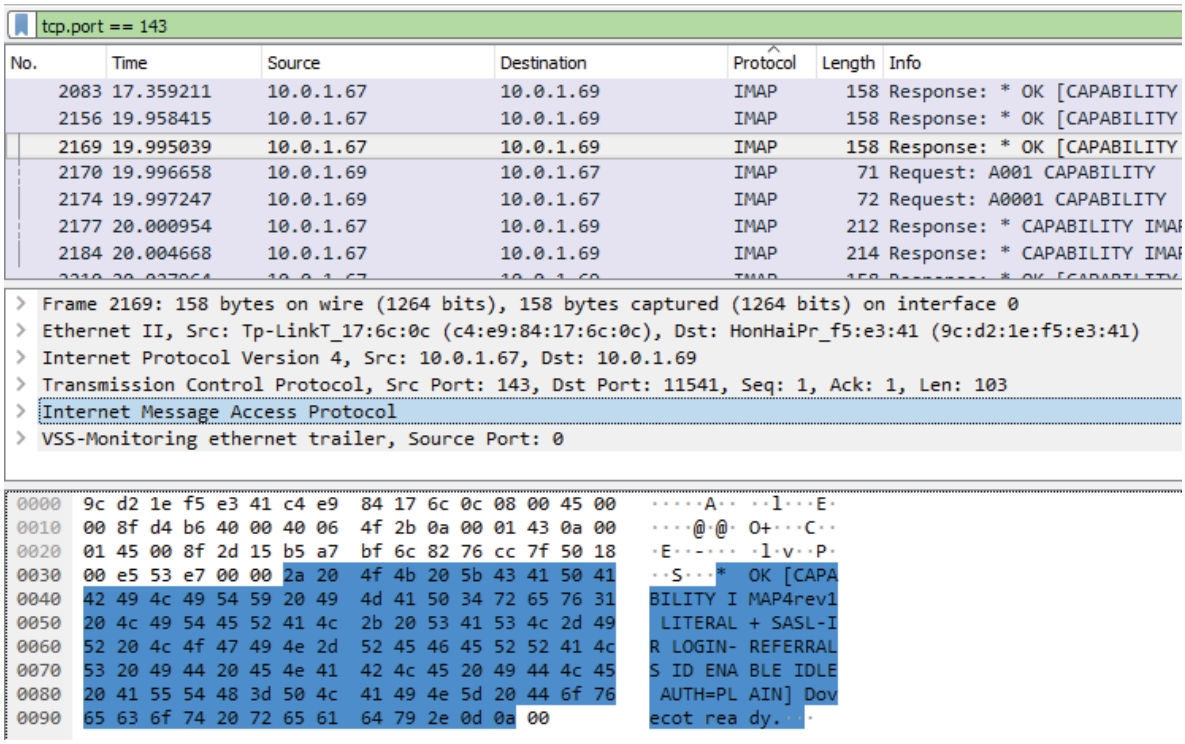

Figura 5.7 Fingerprint servicio IMAP

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.

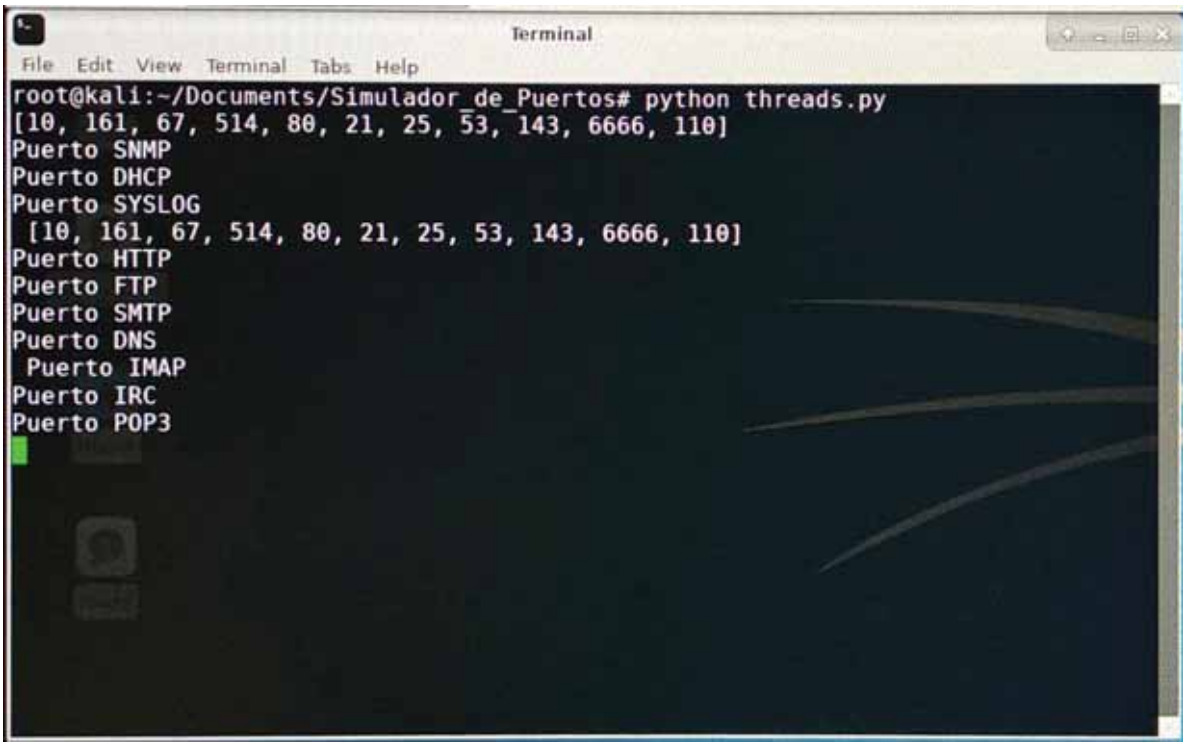

Figura 6.1 Ejecución del script

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.

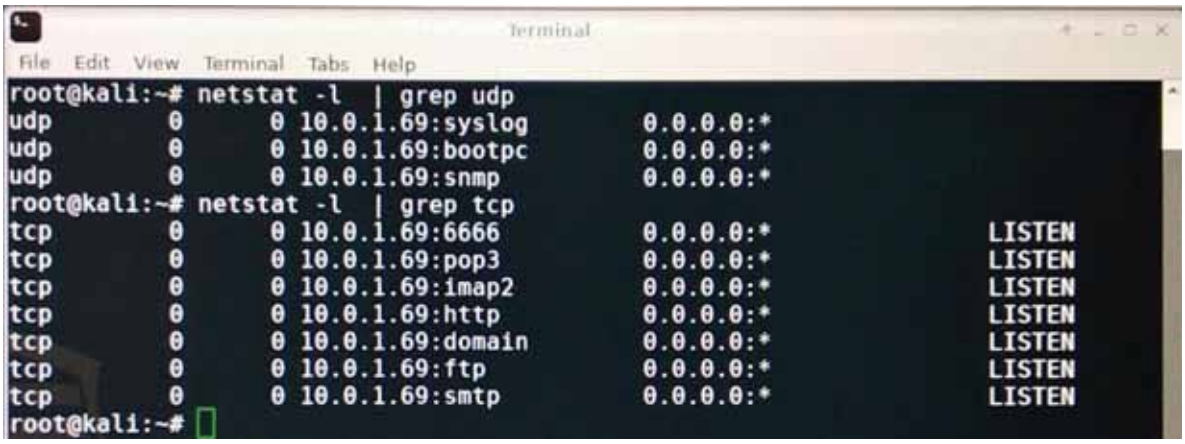

Figura 6.2 Comando para verificar la apertura de puertos.

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.

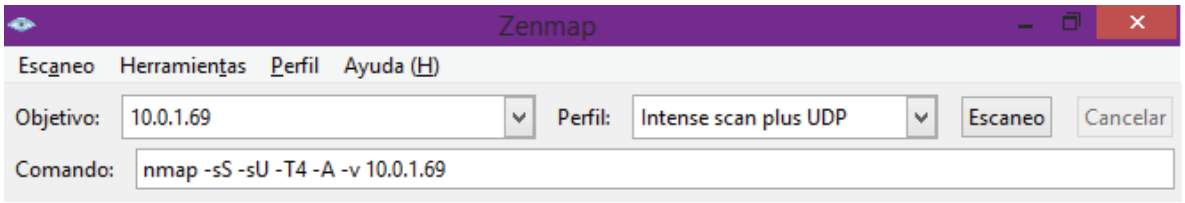

Figura 6.3 Ejecución del escaneo con Nmap

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.

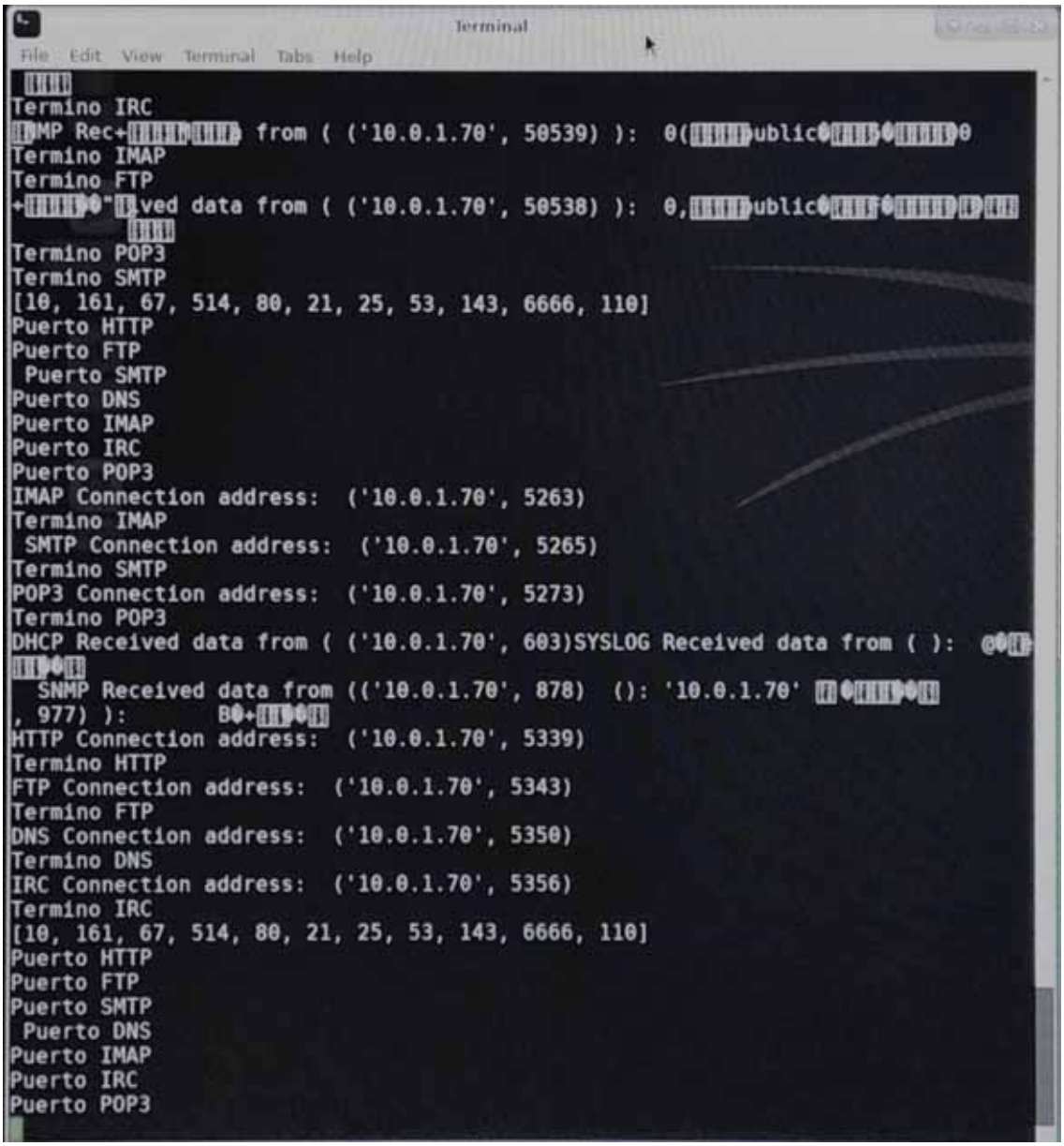

Figura 6.4 Salida de la terminal

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.