ACTIVIDADES MODULARES NETWORK

1ª Actividad escaneo de red

El contenido de esta actividad se ha creado utilizando Kali Linux, una distribución de Linux especializada en seguridad informática, pruebas de penetración y auditorías. Las herramientas y comandos descritos a lo largo de este documento están basados en la funcionalidad proporcionada por Kali Linux.

Usando nmap:

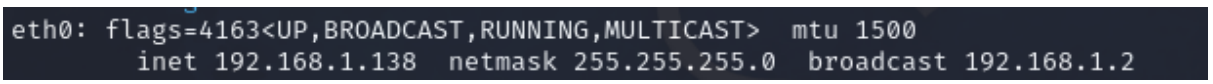
- Explicación:

Aclaraciones:
Durante la realización de este proyecto, utilicé la herramienta Nmap para llevar a cabo un escaneo en mi red local. Este escaneo tuvo como objetivo identificar dispositivos y analizar la configuración de servicios y puertos.
También implementamos Wireshark para realizar análisis de tráfico de red. Wireshark nos permitió examinar en detalle los paquetes de datos que circulan en nuestra red, facilitando la comprensión de patrones de comunicación, protocolos utilizados y cualquier otra información relevante.

Pasos:

Para poder ejecutar este script necesitamos acceder a modo super usuario(sudo su).

Escribiendo en la terminal ifconfig podemos ver la configuración de red que tenemos. En el apartado eth0 podemos distinguir nuestra ip, la máscara de red y el broadcast.



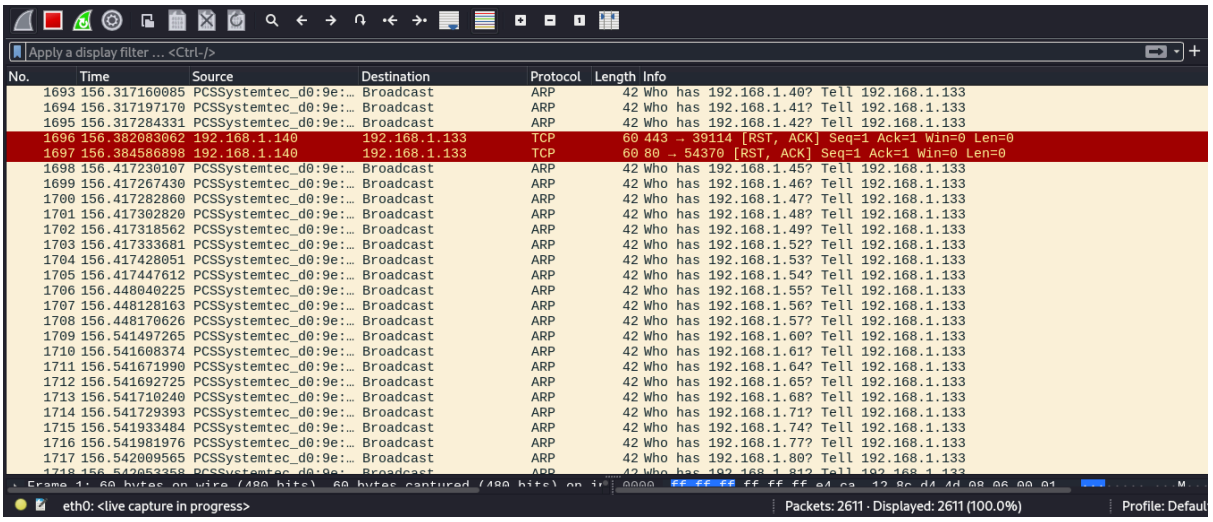
Con esto descubrimos que la dirección de nuestra red es 192.168.1.0 con un máscara /24 = 255.255.255.0, lo que significa que tenemos una ip de clase C.

Para hacer el escaneo de nuestra red local con nmap utilizaremos este comando: nmap -sn ‘dirección ip’, Nmap realiza el envío de solicitudes de eco ICMP (ping) a todas las direcciones IP dentro del rango especificado. A través de este proceso, identifica las direcciones IP que responden positivamente a las solicitudes de ping. El resultado final de este escaneo se presenta como una lista que refleja las

direcciones IP activas en la red local, proporcionando así una visión clara de la conectividad y disponibilidad de dispositivos en la red.

```
rodrix@maquinon ~/Documents/Informáticos nmap -sn 192.168.1.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-12 17:25 CET
Nmap scan report for 192.168.1.1
Host is up (0.0011s latency).
Nmap scan report for 192.168.1.128
Host is up (0.0036s latency).
Nmap scan report for 192.168.1.131
Host is up (0.015s latency).
Nmap scan report for 192.168.1.132
Host is up (0.0064s latency).
Nmap scan report for 192.168.1.133
Host is up (0.0067s latency).
Nmap scan report for 192.168.1.134
Host is up (0.0066s latency).
Nmap scan report for 192.168.1.135
Host is up (0.041s latency).
Nmap scan report for 192.168.1.139
Host is up (0.0073s latency).
Nmap scan report for 192.168.1.140
Host is up (0.071s latency).
Nmap done: 256 IP addresses (9 hosts up) scanned in 3.11 seconds
```

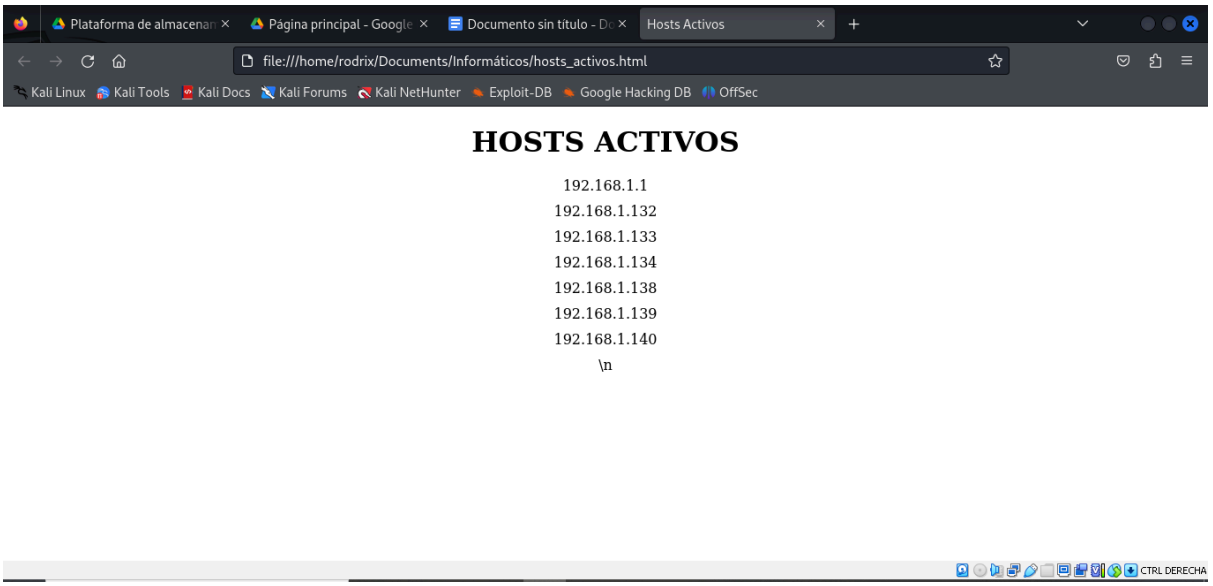
Con wireshark podemos analizar el tráfico de red que estamos generando cuando utilizamos este comando



Si agregamos un par de modificaciones al script podemos conseguir que nos genere un archivo html con estilo en donde esten todas las ips activas de nuestra red local:

```
#!/bin/bash

# Especifica el rango de IP de tu red (ajusta según tu red)
red="192.168.128.0/24"
```

```
# Realiza un escaneo de todos los hosts activos en la red y guarda los
resultados en un archivo
nmap -sn $red -oG resultados.txt

# Crea un archivo HTML con la estructura básica y un título
echo -e "<!DOCTYPE html>\n<html lang=\"en\">\n<head>\n<meta
charset=\"UTF-8\">\n<meta name=\"viewport\"
content=\"width=device-width, initial-scale=1.0\">\n<title>Hosts
Activos</title>\n<style>\nbody { text-align: center; }\np { margin:
10px; }\nh1 { margin-top: 20px; }\n</style>\n</head>\n<body>\n<h1>HOSTS
ACTIVOS</h1>" > script_host_activos.html

# Extrae las direcciones IP activas del archivo de resultados y las
agrega al archivo HTML
awk '/Up$/{print "<p>" $2 "</p>"}' resultados.txt >>
script_host_activos.html

# Cierra la estructura del archivo HTML
echo "</body>\n</html>" >> script_host_activos.html

# Abre el archivo HTML en el navegador predeterminado
xdg-open script_host_activos.html
```

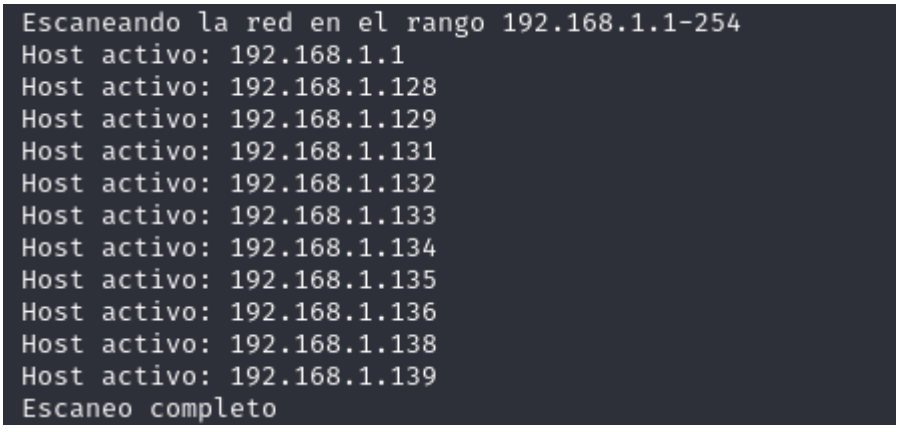


Usando bash

Además de utilizar herramientas especializadas como nmap para realizar escaneos de red, también podemos aprovechar el poder del lenguaje de scripting Bash para crear scripts personalizados que lleven a cabo escaneos de red básicos.

```
#!/bin/bash

# Escaneo de red utilizando ping en un rango de direcciones IP

if [ "$#" -ne 1 ]; then
  echo "Uso: $0 <rango_de_IP>"
  echo "Ejemplo: $0 192.168.1.1-254"
  exit 1
fi

range="$1"

echo "Escaneando la red en el rango $range"

for ip in $(seq -f "192.168.1.%g" 1 254); do
  ping -c 1 -w 1 $ip > /dev/null 2>&1
  if [ $? -eq 0 ]; then
    echo "Host activo: $ip"
  fi
done

echo "Escaneo completo"
```

Este script utiliza un bucle for para iterar a través de direcciones IP en el rango especificado y utiliza el comando ping para determinar si cada host está activo. La salida del script mostrará los hosts activos en la red.



El método de escaneo de red con ping en un script Bash tiende a ser más lento que herramientas especializadas como nmap por varias razones. El enfoque secuencial del script, la espera entre solicitudes, y la limitación de respuestas de algunos dispositivos contribuyen a la lentitud. En contraste, herramientas como nmap

realizan escaneos concurrentes y utilizan técnicas avanzadas para obtener información detallada de la red, lo que las hace más eficientes, especialmente en entornos más grandes o complejos. Aunque el script Bash es útil para tareas básicas, nmap es preferible para escaneos rápidos y detallados.

2ª Actividad calculadora de direcciones ip en binario

Para esta actividad, la cual consiste en crear un script que calcule todos los posibles hosts disponibles de una red con una máscara específica, me he ayudado de chatgpt.

Con una serie de prompts conseguí que me generase un script parecido al resultado que quería sin embargo tenía un par de problemas los cuales tuve que solucionar tocando yo el código.

```
def generate_all_binary_combinations(ip_pattern):
    # Almacenar las direcciones IP en una lista
    ip_addresses = set()  # Utilizar un conjunto para evitar repeticiones

    for bit1 in [0, 1]:
        for bit2 in [0, 1]:
            for bit3 in [0, 1]:
                for bit4 in [0, 1]:
                    for bit5 in [0, 1]:
                        for bit6 in [0, 1]:
                            for bit7 in [0, 1]:
                                for bit8 in [0, 1]:
                                    for bit9 in [0, 1]:
                                        for bit10 in [0, 1]:
                                            for bit11 in [0, 1]:
                                                for bit12 in [0, 1]:
                                                    for bit13 in [0, 1]:
                                                        for bit14 in [0, 1]:
                                                            for bit15 in [0, 1]:
                                                                for bit16 in [0, 1]:
                                                                    ip_address = ip_pattern.format(
                                                                        bit1,
                                                                        bit2,
                                                                        bit3,
                                                                        bit4,
                                                                        bit5,
                                                                        bit6,
                                                                        bit7,
                                                                        bit8,
                                                                        bit9,
                                                                        bit10,
                                                                        bit11,
```

```
                                        bit12,
                                        bit13,
                                        bit14,
                                        bit15,
                                        bit16
                                    )
                                    ip_addresses.add(ip_address)

    return ip_addresses

# Patrón base para la dirección IP en binario con bits variables
ip_pattern = "11000000.{}{}{}{}{}{}{}{}.00001{}0{}.00010111"

# Generar todas las posibles combinaciones de direcciones IP en binario
all_ip_addresses = generate_all_binary_combinations(ip_pattern)

# Imprimir las direcciones IP en binario
for ip_address in all_ip_addresses:
    print(ip_address)

# Nombre del archivo de salida
output_file_name = "IP_address.txt"

# Escribir las direcciones IP en el archivo de salida
with open(output_file_name, "w") as output_file:
    for ip_address in all_ip_addresses:
        output_file.write(ip_address + "\n")
```

- Define una función llamada generate_all_binary_combinations que toma un patrón de dirección IP como argumento.
- Crea un conjunto llamado ip_addresses para almacenar direcciones IP únicas.
- Utiliza bucles anidados para generar todas las combinaciones posibles de bits en la dirección IP, reemplazando los bits variables en el patrón.
- Devuelve el conjunto de direcciones IP generadas por la función.
- Define un patrón de dirección IP en binario con bits variables representados por {}.
- Llama a la función con el patrón de dirección IP y almacena el resultado en all_ip_addresses.
- Define el nombre del archivo de salida como "resultados.txt".

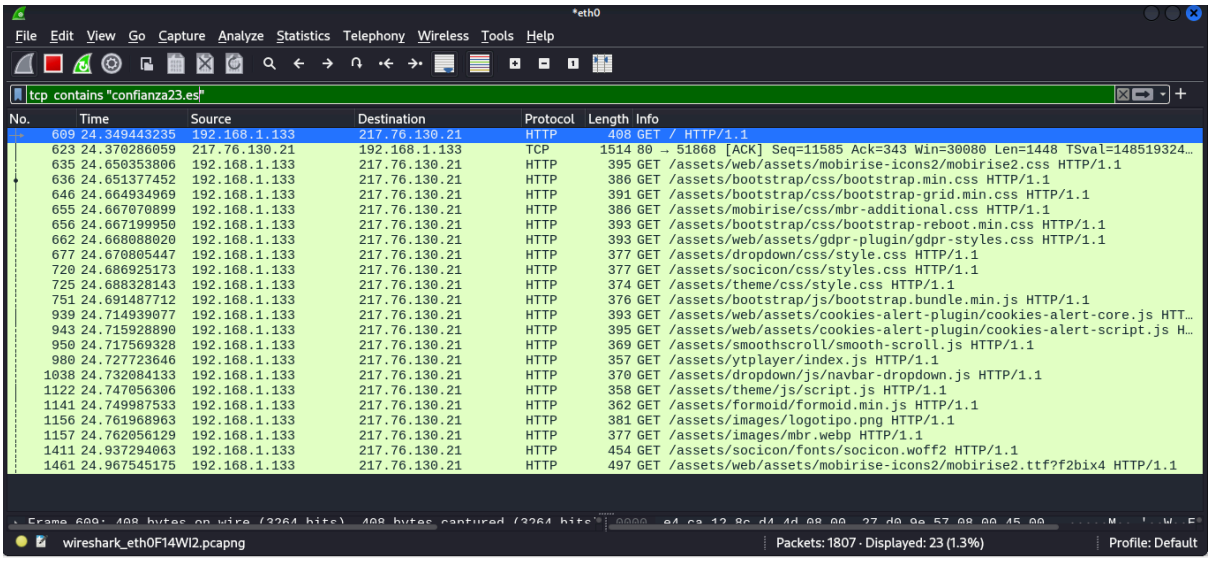
- Abre el archivo en modo de escritura y escribe cada dirección IP única en el archivo, una por línea.

- Imprime un mensaje indicando que las direcciones IP han sido escritas en el archivo.

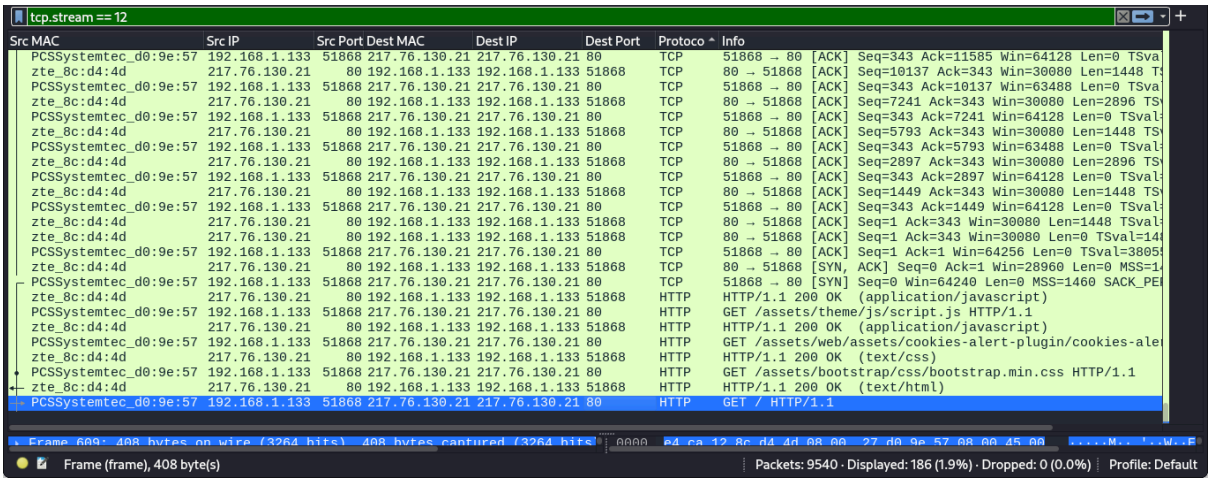
3ª Actividad sniffer captura de tráfico de la red

Para realizar esta actividad he utilizado wireshark. Este módulo tiene como objetivo llevar a cabo la captura de tráfico de una red local en los equipos participantes, simulando el comportamiento normal de un usuario durante un período específico. La finalidad principal de esta actividad es capturar y analizar el patrón de tráfico típicamente asociado con las operaciones habituales de los usuarios en la red. La simulación se centra en reproducir las acciones cotidianas de un usuario convencional dentro de un intervalo de tiempo definido. Esto incluirá actividades como navegación web, envío y recepción de correos electrónicos, acceso a servicios en línea y otras tareas comunes realizadas en el entorno de red. La intención es establecer un "perfil normal" del tráfico de red durante estas operaciones regulares.

Primero iniciamos en wireshark, luego hacemos una búsqueda en google, en este caso www.confianza 23.es. FIltramos en wireshark tcp contains “conifanza23.es” para ver el tráfico generado por la búsqueda;



Y finalmente hacemos la modificaciones especificadas en la actividad;



La finalidad principal es diferenciar este tráfico normal del tipo de tráfico que podría indicar la realización de operaciones relacionadas con el hacking ético o la telemetría de datos.

4ª Actividad Network Scanner, escaneador de red

El objetivo de la actividad es realizar un escaneo de red de una determinada subred (en este caso, 192.168.1.0/24) y generar un informe que incluya la lista de equipos activos a nivel de IP mediante ping ICMP, así como un listado completo de todos los puertos TCP y UDP en esos equipos con su estado (abierto, cerrado, filtrado).

Enfoque:

Escaneo de Direcciones IP:

- Utilicé la herramienta nmap para realizar un escaneo de direcciones IP de la subred especificada. El resultado incluye una lista de equipos activos que responden al ping ICMP.

Escaneo de Puertos TCP y UDP:

- Nuevamente, empleé nmap para escanear todos los puertos TCP y UDP de los equipos identificados como activos. Esto proporciona un listado completo de los puertos y su estado.

Generación de Resultados y Archivo ASCII:

- He utilizado un script en bash para procesar los resultados y mostrarlos en la terminal.
- El resultado se presenta en la terminal mostrando los equipos activos y la información de los puertos.
- Además, redirigí la salida del script a un archivo ASCII llamado resultado.txt para tener una versión persistente y fácil de compartir del informe.

```
#!/bin/bash

equipos_activos=$(nmap -sn 192.168.1.0/24 | grep "Nmap scan report" | awk '{print $5}')

echo "Equipos Activos:"

echo "$equipos_activos"

resultado_puertos=$(nmap -p- -sS -sU --open 192.168.1.0/24)

echo -e "\nListado de Todos los Puertos TCP y UDP:"

echo "$resultado_puertos" | grep -E '^[0-9]+/' | awk '{print "Puerto "$1": "$2}'

echo -e "\nGuardando la información en el archivo 'resultado.txt'"

{

  echo "Equipos Activos:"

  echo "$equipos_activos"

  echo -e "\nListado de Todos los Puertos TCP y UDP:"

  echo "$resultado_puertos" | grep -E '^[0-9]+/' | awk '{print "Puerto "$1": "$2}'

} > resultado.txt
```

- El script se ejecuta en una terminal de Linux mediante el comando `./nombre_del_script.sh`.
- Muestra información en la terminal y guarda un archivo `resultado.txt` con la misma información.

Material/Apuntes/Anotaciones de la Actividad evaluable Integral (Sistemas informáticos)

Videos para reforzar las lecciones de clase:

Qué es la IP pública. Curso de redes desde 0 | Cap 1 | https://www.youtube.com/watch?v=gVUE2IDwWA0&list=PLSvxAUzJ-XSfY0KpwV8SHBIyLVcrZkENc&index=2

Direcciones IP y mascaras de RED
https://www.youtube.com/watch?v=p9onfNNxGyg

Máscara de red (Qué es y como calcular las redes y hosts posibles)
https://www.youtube.com/watch?v=S7F_7Z3qnzA&t=7s

Direccionamiento IPv4 y Subredes (Explicado)
https://www.youtube.com/watch?v=SHbBso63X38

Subneting

- máscara
- nº de subredes
- nº de hosts por cada subred