SISTEMAS INFORMÁTICOS

–

ACTIVIDAD INTEGRAL EVALUABLE SEGUNDA EVALUACIÓN 1º DAM

Alex Álvarez de Sotomayor Sugimoto

1. **Metodología:**
   Todo el software para realizar el trabajo de cada módulo se ha descargado con el gestor de paquetes de debian apt-get. El formato del comando para instalar es:
   - sudo apt-get update
   - sudo apt-get upgrade
   - sudo apt-get nombre_paquete o sudo apt install nombre_aplicación.

2. **Resolución de incidencias:**
   Las incidencias se han resuelto con los apuntes tomados en clase, stack overflow, preguntas a profesores, revisión extensa de las documentaciones oficiales y chatGPT se ha usado para depurar código en en módulo 2.

3. **Evidencias: Introducción**
   El objetivo general de este documento es proyectar el proceso de aprendizaje a través de diversas documentaciones, la instalación de software requerida para cada módulo, plasmar el proceso de diseño y el proceso mental para la escritura de los diversos scripts, así como la bibliografía con todas las fuentes consultadas para la realización de este proyecto. Cada módulo estará compuesto en varias secciones ya descritas en el documento instructivo subido en el campus virtual de iFP en el apartado del curso de sistemas informáticos. A continuación, se describen las secciones para mantener una estructura general de la documentación.
   - Introducción técnica de la tecnología, adaptada al temario de la asignatura.
   - Descripción técnica de los pasos a realizar para descargar el software necesario, instalarlo, configurarlo, ejecutarlo y conseguir los objetivos descritos.
   - Descripción de las problemáticas encontradas y las formas que se han elegido para resolverlas, indicando una taxonomía de:
     - Los pasos que se han realizado de forma manual, de memoria con los conocimientos previos o con la información vista en clase, en los apuntes, en el material de la plataforma, o en el libro de referencia.
     - Listado de URLs de la bibliografía consultada, estándares, guías de buenas prácticas, etc.
     - Los pasos que se han aprendido gracias a buscadores de internet, indicando las cadenas de búsqueda y las URLs de referencia utilizadas, mediante técnicas OSINT y Google Hacking.
     - Los pasos aprendidos gracias a la inteligencia artificial, indicando la plataforma utilizada y las consultas realizadas mediante técnicas de “Prompt Engineering”.

   La intención en este sentido es orientar al alumno hacia las nuevas técnicas de aprendizaje, basado en consultas en buscadores y cadenas de

interacciones con las plataformas de inteligencia artificial como apoyo y ayuda a la rápida resolución de situaciones de atasco y bloqueo, para que, con su posterior procesamiento y depuración, aplicando el sentido común y su propio conocimiento, se logra una aceleración de los resultados en poco tiempo.

4. **Network 1: Script de Hosts Activos**
   Introducción del módulo:

   El objetivo del módulo es crear un script que realice un escaneo de la red para encontrar las IPs activas introduciendo los datos de la red que se quiere escanear y crear una salida para poder posteriormente imprimirlo en pantalla a través de una web basada en HTML, CSS y JavaScript.

   Explicación del script:

   El script se compone de varias funciones:

   - *Comprobar_estado_ip(ip)*
   - *Analizar_red (inicio_i, fin_i, inicio_j, fin_j)*
   - *escribir_json_activas(resultados, directory="../NetInsightWeb", filename="ips_activas.json"):*
   - *imprimir_logo()*
   - *main()*

   Antes de definir las funciones se han importado las siguientes librerías:

```
import os, subprocess, json
```

   - **OS**: provee una manera versátil de usar funcionalidades dependientes del sistema operativo.
   - **Subprocess**: permite lanzar nuevos procesos, conectarse a sus pipes de entrada/salida/error y obtener sus códigos de resultado.
   - **Json**: se trata de un codificador y decodificador de archivos en formato *JavaScript Object Notation (JSON)*.

   A continuación, se explican las funciones que se han creado:

   Comprobar_estado_ip(ip):

```
def comprobar_estado_ip(ip):
    try:
        # Run the ping command with a timeout of 1 second
        subprocess.run( args: ["ping", "-c", "1", "-W", "1", ip], check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        # -c: Le dice a ping que envía solo un paquete ICMP
        # -W: Le pone un límite de tiempo de un segundo a ping. Si no se recibe nada se considera que se ha perdido el paquete
        return True  # Si el comando es exitoso, la IP está activa
    except subprocess.CalledProcessError:
        return False  # Si el comando no es exitoso, la IP no está activa
```

   Esta función realiza lo siguiente:

   - Ejecuta un subproceso con los siguientes argumentos:
     - ["ping", "-c", "1", "-W", "1", ip]:

- ***Ping***: realiza un ping o envío de paquetes ICMP y espera una respuesta del mismo tipo.
- ***-c***: es el número de paquetes que se quieren enviar. En este caso se envía un solo paquete y se espera a una respuesta.
- ***-W***: indica el tiempo de espera en segundos. Si no recibe nada se considera que el paquete se ha perdido.
- ***Ip***: Es la dirección de la IP que se está analizando. Es una variable que se reemplaza de forma dinámica por la dirección IP adecuada en cada iteración del bucle for visto más adelante.
- ***Check=True***: Devuelve subprocess.CalledProcessError si el comando devuelve algo diferente a 0 (código de salida por defecto que indica éxito). Nos permite atrapar el error si el comando ping falla.
- ***Stdout=subprocess.PIPE, stderr=subprocess.PIPE***: Redirigen la salida estándar y el error estándar a "tuberías" o "pipes". Esto permite que el script capture la salida si es necesario. En este caso la salida no se usa en el script, pero se podría capturar en el caso de querer analizarlo.

Analizar_red(inicio_i, fin_i, inicio_j, fin_j):

```
1 usage
def analizar_red(inicio_i, fin_i, inicio_j, fin_j):
    resultados = {}
    for i in range(inicio_i, fin_i + 1):
        for j in range(inicio_j, fin_j + 1):
            ip = f"192.168.{i}.{j}"
            estado = comprobar_estado_ip(ip)
            if estado:
                print(f"IP activa encontrada: {ip}")
            resultados[ip] = estado;
    return resultados
```

La función contiene las siguientes instrucciones:

- Primero le pasamos los siguientes parámetros:
  - ***Inicio_i***: A partir de qué número queremos que empiece el primer octeto.
  - ***Fin_i***: En qué número queremos que termine el primer octeto.
  - ***Inicio_j***: A partir de qué número queremos que empiece el segundo octeto.
  - ***Fin_j***: En qué número queremos que termine el segundo octeto.
- Creamos una lista vacía llamada resultados.

- Creamos dos bucles for anidados. Cada uno va a iterar en los 2 octetos correspondientes.
- Se sustituyen los octetos de forma dinámica por los valores de i y j dentro de los límites que hemos establecido anteriormente.
- A una variable estado le asignamos un valor booleano haciendo una llamada a la función comprobar_estado_ip(ip).
- Asignamos una condición: si la variable estado es verdadera, imprimimos en consola “IP activa encontrada: “ + el valor de la variable ip, siendo este la IP activa que se ha encontrado.
- A la lista resultados le metemos la IP activa que se ha encontrado.
- Finalmente se retorna la lista.

```
def escribir_json_activas(resultados, directory="../NetInsightWeb", filename="ips_activas.json"):
    ips_activas = [{"activa": ip} for ip, status in resultados.items() if status]
    filepath = os.path.join(directory, filename)

    with open(filepath, "w") as json_file:
        json.dump( obj: {"activas": ips_activas}, json_file, indent=2)
```

escribir_json_activas(resultados, directory="../NetInsightWeb", filename="ips_activas.json"):

- Le pasamos los siguientes parámetros:
  - La lista de resultados
  - El directorio donde queremos guardar el archivo JSON.
  - Y el nombre del archivo que se va a crear.
- Guarda cada IP en formato JSON: {“activa”: ip}.
- Abre un archivo JSON en modo escritura en la ruta especificada en los parámetros de la función.
- Vuelca todos los datos de la lista ips_activas.

```
def imprimir_logo():
    logo = """
NNNNNNNN        NNNNNNNNEEEEEEEEEEEEEEEEEEEEEET
N:::::::N       N::::::NE::::::::::::::::::::ET
N::::::::N      N::::::NE::::::::::::::::::::ET
N:::::::::N     N::::::NEE::::::EEEEEEEEE::::ET
N::::::::::N    N::::::N  E:::::E       EEEEEET
N:::::::::::N   N::::::N  E:::::E
N:::::::N::::N  N::::::N  E::::::EEEEEEEEEE
N::::::N N::::N N::::::N  E:::::::::::::::E
N::::::N  N::::N:::::::N  E:::::::::::::::E
N::::::N   N:::::::::::N  E::::::EEEEEEEEEE
N::::::N    N::::::::::N  E:::::E
N::::::N     N:::::::::N  E:::::E       EEEEEE
N::::::N      N::::::::NEE::::::EEEEEEEE:::::E
N::::::N       N:::::::NE::::::::::::::::::::E
N::::::N        N::::::NE::::::::::::::::::::E
NNNNNNNN         NNNNNNNEEEEEEEEEEEEEEEEEEEEEE
    """
    print(logo)
```

Imprimir_logo( ): Imprime el logo en ASCII Art.

```python
def main():
    imprimir_logo()
    print("Bienvenido al Analizador de Red")

    inicio_i = int(input("Ingrese el valor de inicio para el primer octeto: "))
    fin_i = int(input("Ingrese el valor de final para el primer octeto: "))
    inicio_j = int(input("Ingrese el valor de inicio para el segundo octeto: "))
    fin_j = int(input("Ingrese el valor de final para el segundo octeto: "))

    print("\nAnalizando la red...\n")

    resultados = analizar_red(inicio_i, fin_i, inicio_j, fin_j)

    escribir_json_activas(resultados)
    print("\nIPs activas guardadas en ips_activas.json.")

    print("\nAnálisis completo.")
```

Main():

- Llamamos a la función que imprime el logo y damos la bienvenida.
- Pedimos como entrada el valor decimal inicial y final de los octetos i e j.
- Imprimimos “Analizando la red”.
- Guardamos en la variable resultados el retorno de la función *analizar_red*.
- Llamamos a la función *escribir_json_activas* para guardar los resultados.
- Imprimimos los últimos mensajes.

5. **Network 2: Calculadora de direcciones IP en binario:**

   Introducción del módulo:

   Este módulo tiene como objetivo crear un script que tenga como entrada una dirección IP y una máscara para poder calcular todas la IPs posibles dentro de la red. La salida la exporta a un archivo JSON con todas las direcciones IP posibles convertidas a binario.

   Explicación del script:

```python
import json

1 usage
def calcular_ips(ip, mascara):
    # Convertir la dirección IP y la máscara de red a una lista de enteros
    ip_split = [int(num) for num in ip.split('.')]
    mascara_split = [int(num) for num in mascara.split('.')]

    # Calcular la dirección de red
    direccion_red = [ip_split[i] & mascara_split[i] for i in range(4)]

    # Calcular la cantidad de bits de host
    bits_host = sum([bin(mascara_split[i] ^ 255).count('1') for i in range(4)])

    # Calcular la cantidad de direcciones disponibles
    direcciones_disponibles = 2**(32 - bits_host)

    # Calcular las direcciones IP disponibles
    direcciones_binarias = []
    for i in range(1, direcciones_disponibles - 1):
        direccion = direccion_red.copy()
        direccion[3] += i
        direcciones_binarias.append('.'.join([bin(num)[2:].zfill(8) for num in direccion]))

    return direcciones_binarias

ip = "192.168.8.23"
mascara = "255.0.250.255"
ips_binarias = calcular_ips(ip, mascara)

# Format IPs into the desired structure
formatted_ips = [{"ip_binaria": ip} for ip in ips_binarias]

# Specify the path where you want to save the JSON file
output_path = "/home/alexsugimoto/Documentos/DAM/Sistemas_Informáticos/2EV/ASAlex_Actividad_Evaluabl

# Write IPs to JSON file
with open(output_path, 'w') as json_file:
    json.dump( obj: {"ip_binaria": formatted_ips}, json_file, indent=2)
```

Calcular_ips(ip, mascara) realiza lo siguiente:

- Primero convierte la dirección IP y la máscara de red a una lista de enteros haciendo un split cada vez que haya un punto y haciendo type casting a un int.
- Después calcula la dirección de la red haciendo una operación AND entre los octetos de la IP y a mascara. Todo esto lo hace a través de un bucle for.
- En el tercer paso del algoritmo, se calcula la cantidad de bits que tiene el host.
- Para calcular la cantidad de direcciones posibles se realiza la siguiente operación: $2x10^{32\text{-bits\_host}}$
- Finalmente se calcula la cantidad de direcciones disponibles en binario y se retornan las direcciones.
- Finalmente, se le da el formato requerido para guardarlo en formato JSON y se guarda en un archivo en la ruta especificada del proyecto.

6. Network 3: Captura de tráfico de red con Wireshark

Introducción al módulo:

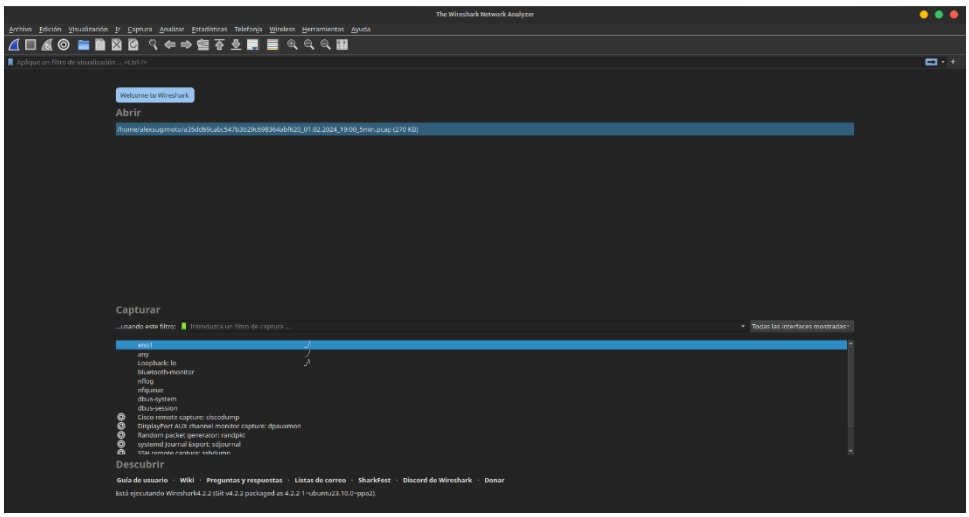
Este módulo consta de un simple objetivo: capturar todo el tráfico de red entre la IP de la máquina y la IP de la página www.confianza23.es pasándolo por un filtro para detectar únicamente el tráfico http. Luego se

quiere exportar la captura a un archivo .pcap/.pcapng para importarlo a la herramienta GrassMarlin de la NSA y poder obtener una topología lógica de la red.

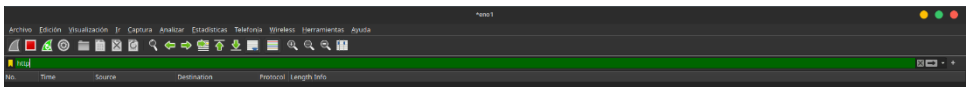
Explicación del módulo:

- Primero, abro WireShark desde un terminal ejecutando sudo wireshark para poder tener todos los permisos para analizar el tráfico de red. En mi caso, quiero escanear el tráfico que pasa por el cable Ethernet eno1.
- Empiezo la captura y establezco un filtro en la parte superior de la interfaz gráfica para capturar únicamente el tráfico http.
- Con el filtro puedo observar 2 paquetes al cargar la página de www.cofianza23.es en mi navegador con todas sus cabeceras y la información que transportan.
- Como nota, he podido observar una posible vulnerabilidad en la seguridad de la red ya que los paquetes traen el código fuente de la red sin encriptar. Esto podría suponer una posible suplantación de identidad de la web para realizar ataques phishing.

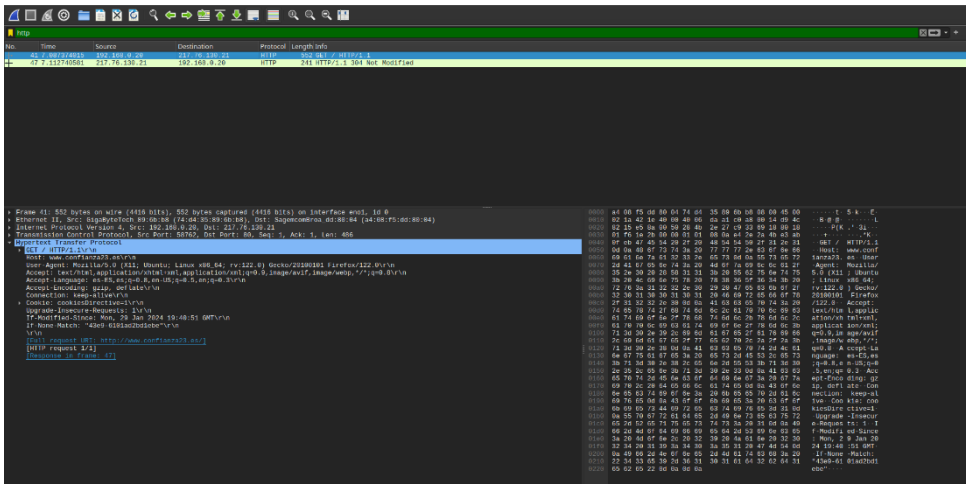
A continuación, las capturas del trabajo con WireShark:



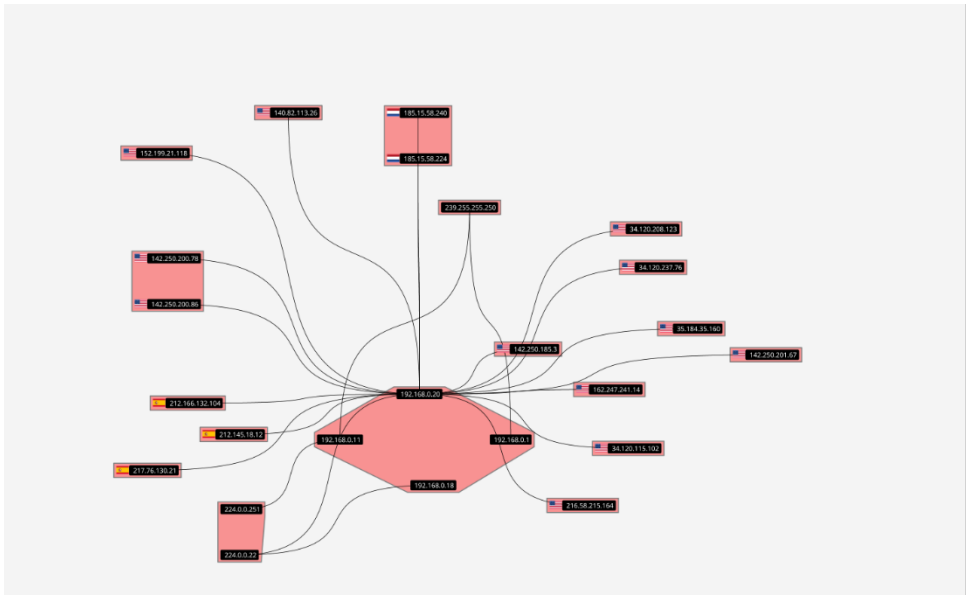
Página inicial de WireShark 1



Estableciendo el filtro http 1



Paquetes de tráficos cargados



Topología de red en GrassMarlin 1

7. Network 4: Escaneador de red:

Introducción al módulo:

- El objetivo consiste realizar un escaneo exhaustivo de la red con la potente herramienta en línea de comandos de Linux nmap.
- Se introducirá como entrada un rango de direcciones, que en este caso y como se viene trabajando en este proyecto es el de mi red local.
- En un archivo de texto se exportarán las salidas de los comandos ejecutados.

Explicación del módulo:

- Para detectar las IPs vivas:
  - nmap -sP -oN IPsvivas.txt 192.168.0.1-255
    - Nmap es el comando por defecto para ejecutar la herramienta.

- -sP realiza un escaneo ping para ver qué IPs están vivas.
- -oN es el argumento de salida normal o por defecto de nmap, siendo un archivo txt, llamado en este caso IPsvivas.txt.
- 192.168.0.1-255 es el rango de IPs que se quieren escanear. Como mi IP de mi máquina es 192.168.0.20/24 el rango de direcciones disponibles es el último octeto.

```
# Nmap 7.94SVN scan initiated Tue Feb  6 16:30:21 2024 as: nmap -sP -oN IPsvivas.txt 192.168.0.1-255
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.0058s latency).
Nmap scan report for 192.168.0.11
Host is up (0.0088s latency).
Nmap scan report for 192.168.0.12
Host is up (0.11s latency).
Nmap scan report for 192.168.0.15
Host is up (0.064s latency).
Nmap scan report for 192.168.0.17
Host is up (0.10s latency).
Nmap scan report for 192.168.0.19
Host is up (0.058s latency).
Nmap scan report for alexsugimoto (192.168.0.20)
Host is up (0.000033s latency).
# Nmap done at Tue Feb  6 16:30:41 2024 -- 255 IP addresses (7 hosts up) scanned in 19.78 seconds
```

- Para realizar un escaneo de los puertos TCP:
  - nmap -p 1-65535 -oN 192.168.0.20.txt 192.168.0.20
    - Nmap es el comando de ejecución de
    - -p nos permite especificar qué puertos queremos escanear. En este caso le damos un rango que cubre todos los puertos posibles desde el 1 al 65535.
    - -oN la salida por defecto de la herramienta, exportándolo como antes a un .txt.
    - 192.168.0.20 la IP objetivo a escanear.

```
# Nmap 7.94SVN scan initiated Tue Feb  6 16:52:23 2024 as: nmap -p 1-65535 -oN 192.168.0.20.txt 192.168.0.20
Nmap scan report for alexsugimoto (192.168.0.20)
Host is up (0.0000030s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE
3389/tcp  open  ms-wbt-server
3390/tcp  open  dsc
57621/tcp open  unknown
59277/tcp open  unknown

# Nmap done at Tue Feb  6 16:52:29 2024 -- 1 IP address (1 host up) scanned in 5.90 seconds
```

- Para realizar un escaneo de los puertos UDP:
  - Es un comando muy similar al del escaneo de los puertos TCP. Únicamente se le añade el argumento –sU para escanear los puertos UDP. nmap -sU -p 1-65535 -oN 192.168.0.20-UDP.txt 192.168.0.20.

```
# Nmap 7.94SVN scan initiated Tue Feb  6 16:42:01 2024 as: nmap -sU -p 1-65535 -oN 192.168.0.20-UDP.txt 192.168.0.20
Nmap scan report for alexsugimoto (192.168.0.20)
Host is up (0.0000030s latency).
Not shown: 65530 closed udp ports (port-unreach)
PORT      STATE         SERVICE
1900/udp  open|filtered upnp
5353/udp  open|filtered zeroconf
43004/udp open|filtered unknown
57621/udp open|filtered unknown
58083/udp open|filtered unknown

# Nmap done at Tue Feb  6 16:42:13 2024 -- 1 IP address (1 host up) scanned in 12.83 seconds
```

Podemos observar que las salidas de los escaneos de los puertos nos arrojan información sobre el estado del puerto (si está abierto, filtrado o cerrado) y los servicios que se estén ejecutando en el puerto.

8. System 1

Introducción al módulo:

El objetivo de este módulo es crear un script que genere ciertos datos relacionados con la máquina. Sacamos la siguiente información: datos generales de la máquina, datos generales del hardware, datos generales de la CPU, datos generales de los dispositivos de bloque, datos generales estáticos y dinámicos de la red e información de los servicios internos.

Explicación del script:

- Importamos los módulos de os y platform y realizamos llamadas a funciones que nos permiten sacar información.
    - OS: Nos permite interactuar con el sistema operativo. En este caso lo usamos para abrir y escribir archivos:
        - Os.popen:
            - Abra una tubería hacia o desde el comando cmd. El valor de retorno es un objeto de archivo abierto conectado a la tubería, que puede leerse o escribirse.
    - Platform nos permite sacar información de la máquina:
        - Platform.machine(): Da el tipo de máquina.
        - Platform.node(): Da el nombre de la red de la máquina.
        - Platform.platform(): Identifica el sistema operativo.
        - Platform.processor(): Da el nombre
        - Platform.release(): Devuelve la versión del sistema.
- Realizamos una serie de comandos que nos permiten sacar información sobre varias partes de la máquina. En mi caso al tener una máquina personal con Linux, explico los comandos de Linux:
    - Uname -v: Nos da la versión del kernel.
    - Lshw -short: Datos del hardware de la máquina.
    - Lscpu: datos generales de la CPU.
    - Lsblk -a: datos generales de los dispositivos de bloque (discos duros).
    - Ifconfig: datos estáticos de la red

    - Arp -a: tabla ARP.
    - Ifconfig | grep ether: Dirección MAC de la máquina.
    - Ip route: tabla de enrutamiento IP.
    - Netstat -rn: Nos da estadísticas de la red con una tabla de enrutamiento.
    - Hwinfo: Datos del hardware de red.
    - Systemctl –type=service: nos da el estado de los servicios.
    - Os.popen("top"): Nos da los datos internos de los procesos en memoria.
- Finalmente imprimimos en pantalla la ruta donde se ha guardado el archivo de salida.

```python
import platform
import os

output_file_path = "infoMaquina.txt"

# Redirigir la salida estándar a un archivo txt
with open(output_file_path, 'w') as f:
    # Información general de la máquina
    f.write("---------------------------------\n")
    f.write("INICIO DE LOS DATOS GENERALES DE LA MÁQUINA\n")
    f.write("Arquitectura de la máquina: " + platform.machine() + "\n")
    f.write("Nombre del host: " + platform.node() + "\n")
    f.write("Sistema Operativo: " + platform.platform() + "\n")
    f.write("Nombre del procesador: " + platform.processor() + "\n")
    f.write("Versión del sistema: " + platform.release() + "\n")
    f.write("Versión del kernel: ")
    f.write(os.popen("uname -v").read() + "\n")

    f.write("FIN DE LOS DATOS GENERALES DE LA MÁQUINA\n")
    f.write("---------------------------------\n")

    # Información del hardware
    f.write("INICIO DE LOS DATOS DE HARDWARE DE LA MÁQUINA\n")
    f.write(os.popen("lshw -short").read() + "\n")

    f.write("FIN DE LOS DATOS DE HARDWARE DE LA MÁQUINA\n")
    f.write("---------------------------------\n")

    # Información de la CPU
    f.write("INICIO DE LOS DATOS GENERALES DE LA CPU\n")
    f.write(os.popen("lscpu").read() + "\n")
    f.write("FIN DE LOS DATOS GENERALES DE LA CPU\n")
    f.write("---------------------------------\n")

    # Información de los dispositivos de bloque
    f.write("INICIO DE LOS DATOS GENERALES DE LOS DISPOSITIVOS DE BLOQUE\n")
    f.write(os.popen("lsblk -a").read() + "\n")
    f.write("FIN DE LOS DATOS GENERALES DE LOS DISPOSITIVOS DE BLOQUE\n")
    f.write("---------------------------------\n")

    # Información de la red estática
    f.write("INICIO DE LOS DATOS ESTÁTICOS DE RED\n")
```

```
f.write("Tabla de rutas: \n")
f.write(os.popen("ip route").read() + "\n")
f.write("Estadísticas de red: \n")
f.write(os.popen("netstat -rn").read() + "\n")
f.write("Adaptadores de red: \n")
f.write(os.popen("hwinfo").read() + "\n")
f.write("FIN DE LOS DATOS DINÁMICOS DE RED\n")
f.write("---------------------------------\n")

# Información de los servicios internos
f.write("INICIO DE LOS DATOS INTERNOS DE PROCESOS EN MEMORIA\n")
f.write("Estado de los servicios: \n")
f.write(os.popen("systemctl --type=service").read() + "\n")
f.write("Servicios corriendo actualmente: \n")
f.write(os.popen("top").read() + "\n")
f.write("FIN DE LOS DATOS INTERNOS DE PROCESOS EN MEMORIA\n")
f.write("---------------------------------\n")

print(f"Output written to {os.getcwd()}")
```

9. **Interfaz web y consolidación del proyecto**

Introducción:

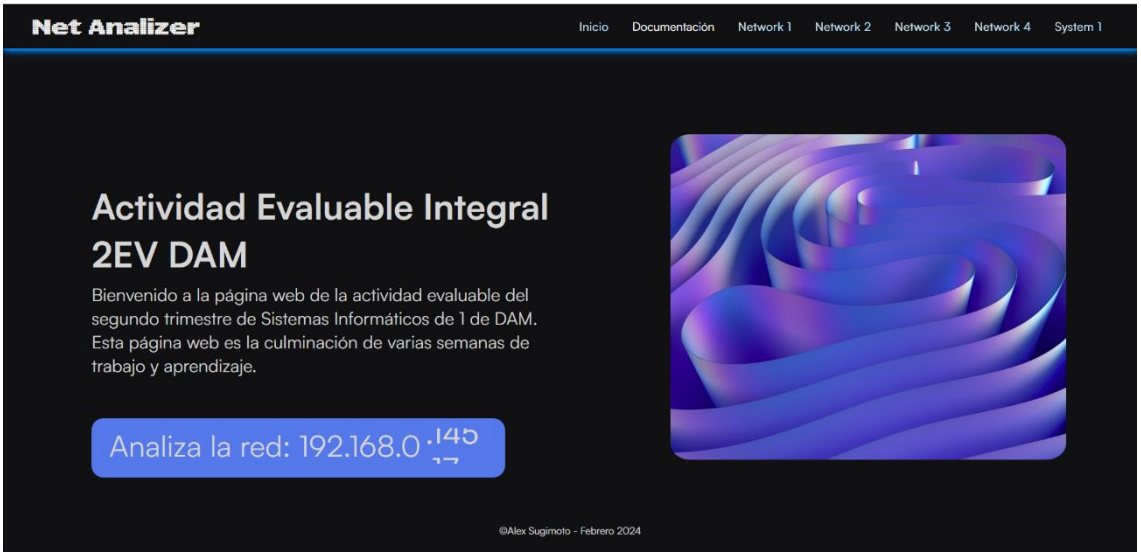
Para consolidar todo el trabajo realizado se ha creado una interfaz web con una página inicial y varias páginas que explican brevemente cada módulo y da un ejemplo muy gráfico. Además, tiene una página de documentación en la que está este documento.

NOTA IMPORTANTE: La página web no es responsive, se debe mirar con el navegador maximizado, pero sin la pantalla completa. Debería ser vista en Firefox ya que no se ha trabajado la compatibilidad entre navegadores y se ha desarrollado solo para Firefox.

Explicación de la interfaz gráfica:

La interfaz tiene varias partes: inicio, documentación, Network 1, Network 2, Network 3, Network 4 y System 1. Todas las páginas tienen el elemento del navegador y el footer con el copyright en común y además, vienen con una imagen para estilizar la primera parte de cada página.

Inicio:

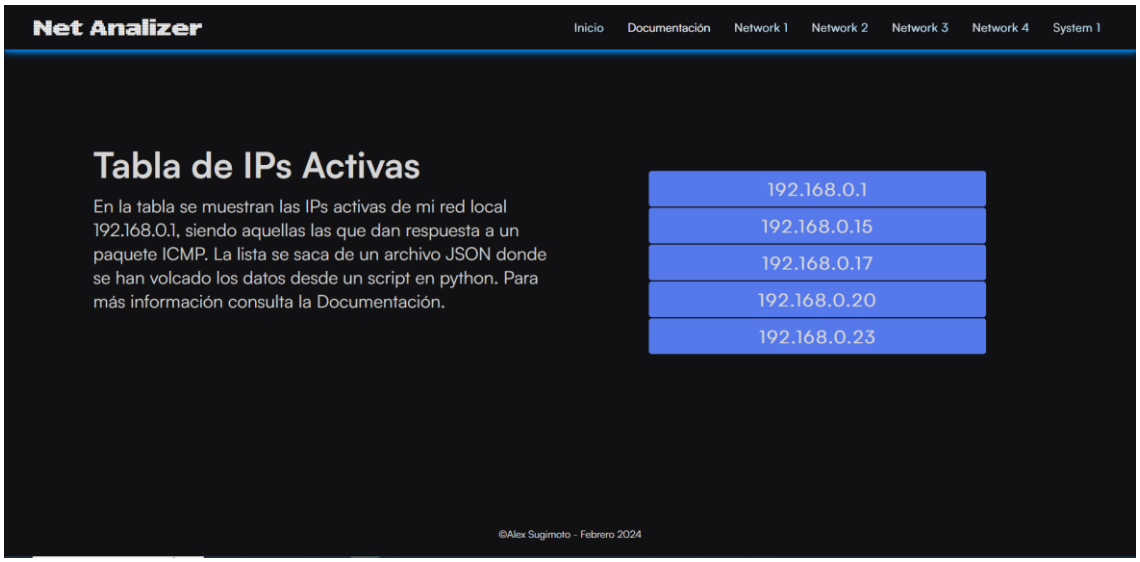


Página de inicio 1

La página de inicio contiene 2 divs de texto y un div con una mágina estilizada con css gracias a la función clip-path. Se ha añadadido un div con un scroller para representar que se puede analizar cualquier red.
El scroller es una simple animación con css que van moviendo la posición del span de forma fluida.

Network 1:

Primera parte de Network 1



Segunda parte de Network 2

Lo interesante de esta página es el div que pone Mostrar todas las IPs vivas:

```
let lista_activas = []
console.log(lista_activas)
let lista_binarias = []

fetch('ips_activas.json')
    .then(response => response.json())
    .then(data => {
        console.log('Fetched active IPs:', data);
        if (data && data.activas) {
            // Populate the lista_activas array
            lista_activas = data.activas.map(ipObj => ipObj.activa);

            // You can use the populated array elsewhere in your script
            console.log('Populated lista_activas array:', lista_activas);
        } else {
            console.error('Invalid or missing data in ips_activas.json');
        }
    })
    .catch(error => console.error('Error fetching active IPs:', error));



window.onbeforeunload = function () {
    window.scrollTo(0, 0);
}

function fActivas() {
    window.scrollTo({
        top: document.body.scrollHeight,
        behavior: "smooth"
    });
    let html = "<table id='tabla1'>";
    for (i = 0; i < lista_activas.length; i++) {
        html += "<tr>";
        html += "<td>" + lista_activas[i] + "</td>";
        html += "</tr>";
    }
    html += "</table>"
    document.querySelector("#activas").innerHTML = html;
}
```

Recordamos que la actividad Network 2 exporta un archivo JSON. La ruta de este archivo es la carpeta donde tenemos alojada la página web. Gracias a

esto podemos sacar la información del archivo JSON metiéndolo en una lista llamada lista_binarias. Lo metemos realizando una función fetch en JavaScript y guardando dentro de una variable data la lista. Con los datos se puebla la variable lista_binarias.

```
window.onbeforeunload = function () {
window.scrollTo(0, 0);
}
```

Esta función nos permite hacer un scroll automático al inicio de la página cada vez que se refresque.

```
function fActivas() {
    window.scrollTo({
        top: document.body.scrollHeight,
        behavior: "smooth"
    });
    let html = "<table id='tabla1'>";
    for (i = 0; i < lista_activas.length; i++) {
        html += "<tr>";
        html += "<td>" + lista_activas[i] + "</td>";
        html += "</tr>";
    }
    html += "</table>"
    document.querySelector("#activas").innerHTML = html;
}
```

Esta función realiza varias cosas:

```
window.scrollTo({
        top: document.body.scrollHeight,
        behavior: "smooth"
    });
```

Baja automáticamente de forma fluida hasta la parte más baja de la página.

```
for (i = 0; i < lista_activas.length; i++) {
        html += "<tr>";
        html += "<td>" + lista_activas[i] + "</td>";
        html += "</tr>";
    }
    html += "</table>"
```

Con un bucle for sacamos los elementos necesarios para enseñar las ips activas.

```
document.querySelector("#activas").innerHTML = html;
```

Añadimos los elementos generados al div con id “activas”.

Network 2:

Primera parte de Network 1

Segunda parte de Network 2

Al igual que en la página anterior lo interesante es el botón que pone mostrar todas la IPs vivas. La función en JavaScript es:

```
function fBinarias() {
    document.querySelector("#loader").style.visibility = "visible";
    document.querySelector("#loader_texto").style.visibility = "visible";
    fetch('ips_binarias.json')
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(data => {
            const items = data.ip_binaria;

            const shuffledItems = items.sort(() => Math.random() - 0.5);
            const randomItems = shuffledItems.slice(0, 5);
            const lista_binarias = randomItems.map(item =>
item.ip_binaria);
            document.querySelector("#loader").style.visibility =
"hidden";
            document.querySelector("#loader_texto").style.visibility =
"hidden";
            window.scrollTo({
                top: document.body.scrollHeight,
                behavior: "smooth"
            });
            // Generate HTML table
            let html = "<table>";
            lista_binarias.forEach(ip => {
                html += "<tr>";
                html += "<td>" + ip + "</td>";
                html += "</tr>";
            });
            html += "</table>";
            document.querySelector("#binarias").innerHTML = html;
        })
        .catch(error => {
            console.error('There was a problem with the fetch
operation:', error);
        });
}
```

```
document.querySelector("#loader").style.visibility = "visible";
    document.querySelector("#loader_texto").style.visibility = "visible";
```

Muestra en pantalla un icono loader mientras cargan los datos.

Página con el loader 1

```
fetch('ips_binarias.json')
        .then(response => {
            if (!response.ok) {
                throw new Error('ERROR en la respuesta del servidor.');
            }
            return response.json();
        })
        .then(data => {
            const items = data.ip_binaria;

            const shuffledItems = items.sort(() => Math.random() - 0.5);
            const randomItems = shuffledItems.slice(0, 5);
            const lista_binarias = randomItems.map(item =>
item.ip_binaria);
            document.querySelector("#loader").style.visibility =
"hidden";
            document.querySelector("#loader_texto").style.visibility =
"hidden";
            window.scrollTo({
                top: document.body.scrollHeight,
                behavior: "smooth"
            });
            let html = "<table>";
            lista_binarias.forEach(ip => {
                html += "<tr>";
                html += "<td>" + ip + "</td>";
                html += "</tr>";
            });
            html += "</table>";
            document.querySelector("#binarias").innerHTML = html;
        })
```

Luego, con la función fetch sacamos los datos del JSON y los volcamos en data. Hacemos un control de errores por si falla la respuesta del servidor, si no devolvemos la respuesta.

A continuación, los campos ip_binaria de data se vuelcan en la variable constante ítems y se barajean con un random para asignarlo a una variable llamada randomItems. Luego se asignan los valores de randomItems a la variable lista_binarias.

Una vez realizado todo esto ocultamos el loader de la página y hacemos un scroll hasta abajo con:

```
window.scrollTo({
                top: document.body.scrollHeight,
                behavior: "smooth"
            });
```

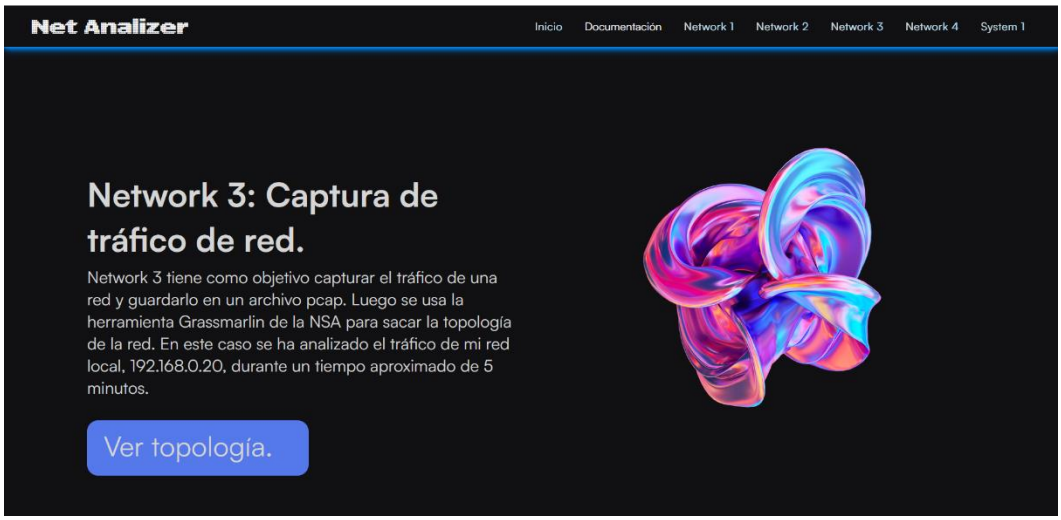
Finalmente, imprime los valores binarios de las IPs seleccionadas de forma aleatoria en pantalla.

```
let html = "<table>";
            lista_binarias.forEach(ip => {
                html += "<tr>";
                html += "<td>" + ip + "</td>";
                html += "</tr>";
            });
            html += "</table>";
            document.querySelector("#binarias").innerHTML = html;
```

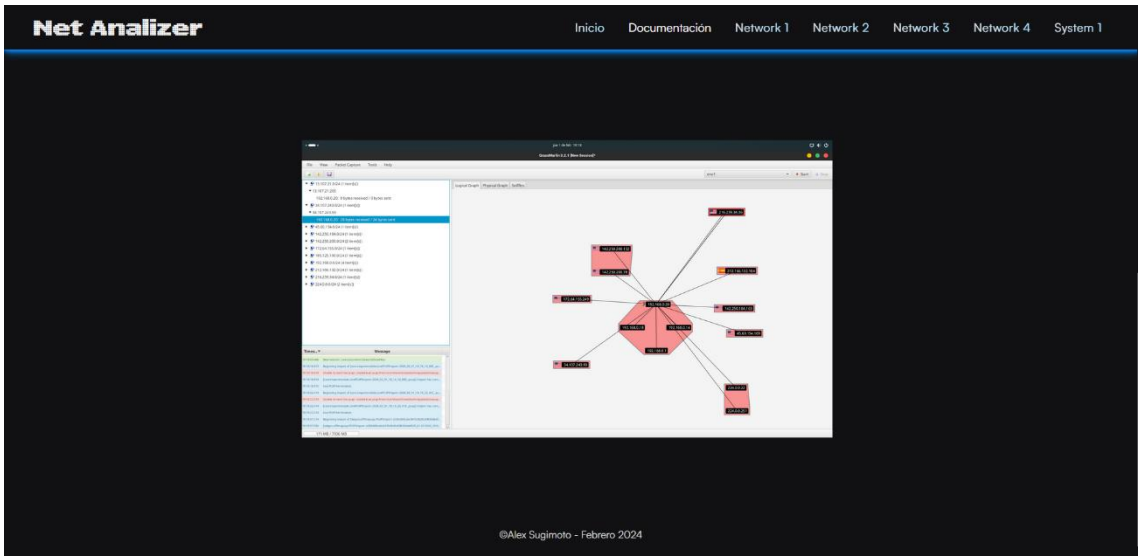
Network 3, 4 y System 1:

Estas páginas son muy simples:

- Network 3 tiene una simple captura de la herramienta GrassMarlin.



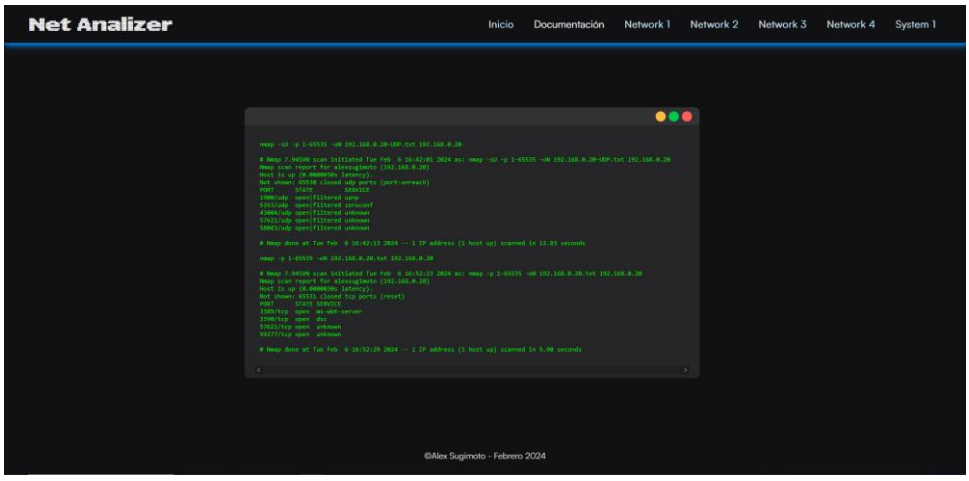
Página principal Network 3



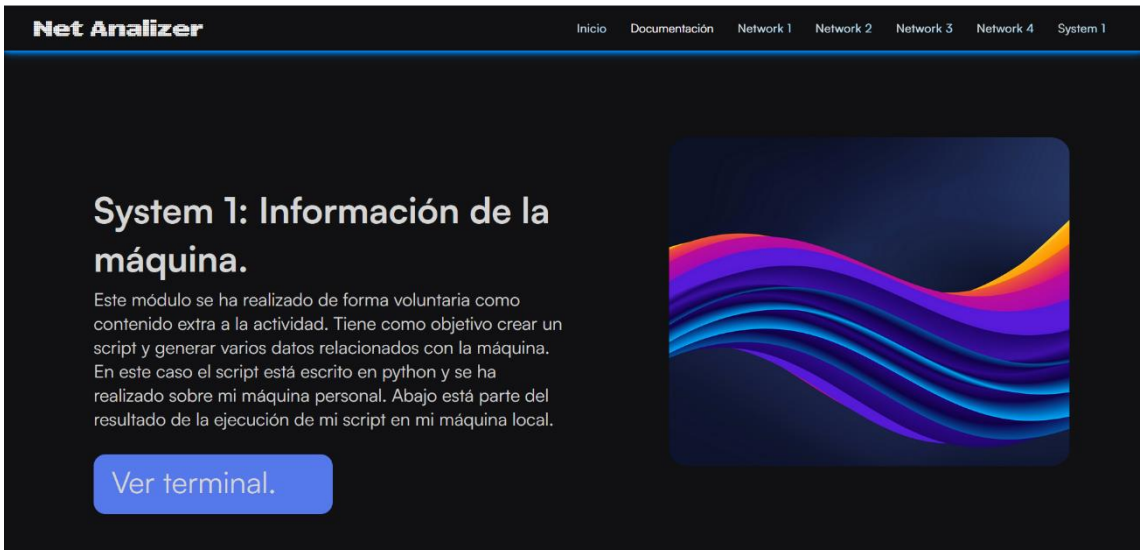
Parte inferior de Network 3

- Network 4 y System 1 tienen una falsa terminal construida con CSS que contiene texto pre-formateado con las salidas de los diversos scripts.

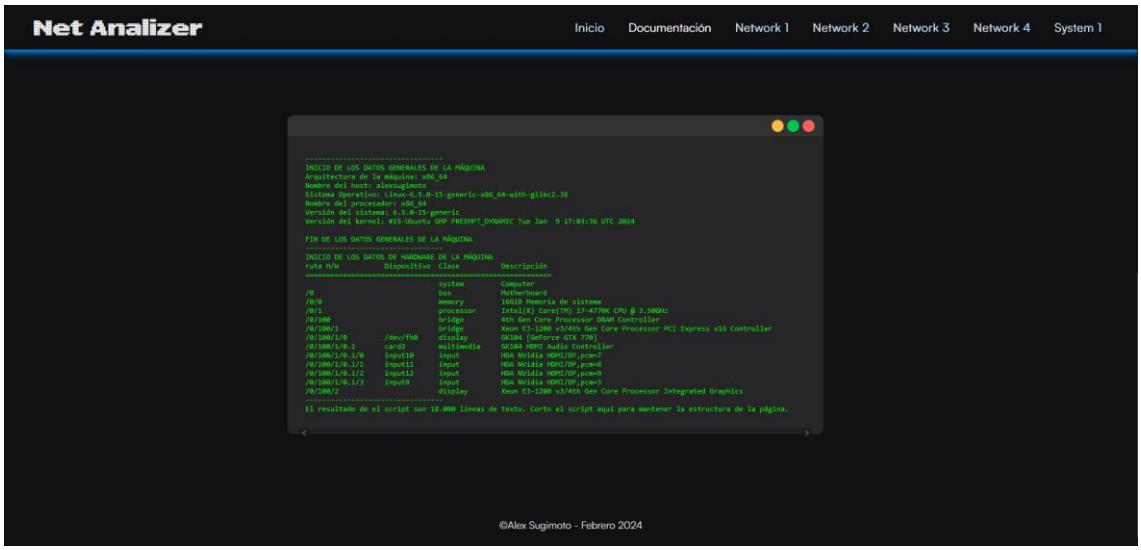
Página principal Network 4



Terminal en CSS de Network 4



Página principal System 1



Terminal en CSS System 1

10. Bibliografía:

platform — Access to underlying platform's identifying data — Python 3.12.2 documentation

os — Miscellaneous operating system interfaces — Python 3.12.2 documentation

How to Use Wireshark to Capture, Filter and Inspect Packets (howtogeek.com)

Nmap: the Network Mapper - Free Security Scanner

Uso de Fetch - Referencia de la API Web | MDN (mozilla.org)

Window: scrollTo() method - Web APIs | MDN (mozilla.org)

Array.prototype.sort() - JavaScript | MDN (mozilla.org)

Get Linux System and Hardware Details on the Command Line - VITUX