



Instituto Politécnico Nacional



Escuela Superior De Cómputo

Ingeniería En Sistemas Computacionales

Redes de computadoras

Profesor:

Puebla Lomas Jaime Hugo

Alumno:

Olguín Martínez José Arturo

Proyecto

Analizador de protocolos de red con Scapy

# Índice

Estado del arte.....	4
Detection and Prevention of Poisoning Targets with ARP Cache using Scapy .....	4
SentinelScan: Advanced Network Scanner and Packet Detection Suite .....	5
Visualization of DDoS Attack using Python Libraries .....	6
Scapy Scripting to Automate Testing of Networking Middleboxes.....	7
SCAPY- A powerfull interactive packet manipulation program.....	8
Introducción .....	9
Scapy como librería de Python.....	9
Características principales de Scapy.....	9
Objetivo .....	10
Desarrollo .....	11
Requisitos para Scapy.....	11
Importación de librerías .....	12
Definición de la clase Sniffer .....	13
Definiciones.....	14
Inicialización.....	14
Inicio de la captura .....	14
Lectura de los paquetes.....	15
Filtro por protocolo .....	15
Impresión de los paquetes .....	16
Exportación del archivo pcap .....	17
Función principal main.....	17
Gráficos .....	20
Diagrama de casos de uso UML .....	20
Diagrama de secuencias .....	21
Pruebas de funcionamiento .....	22
Ejecución del programa con permisos del administrador. ....	22
Pruebas de detección.....	23
Impresión de detalles .....	24
Recopilación.....	24
Prueba de exportación .....	25

Prueba de captura en Wireshark .....	25
Conclusiones .....	26
Bibliografía.....	27

## Estado del arte

### Detection and Prevention of Poisoning Targets with ARP Cache using Scapy

Debido a sus vulnerabilidades, ARP se convierte en un objetivo para atacantes que buscan ejecutar falsificación o envenenamiento de ARP, lo cual permite manipular la caché ARP de los dispositivos en red, generando problemas de integridad y confidencialidad de datos.

La amenaza principal son los ataques de envenenamiento ARP explotan la falta de autenticación en el protocolo, permitiendo a un atacante interceptar, modificar, o redirigir tráfico de red mediante paquetes ARP falsificados. Estos ataques pueden llevar a consecuencias serias, como ataques de hombre en el medio (MITM) y denegación de servicio (DoS), que afectan a dispositivos específicos o a toda la red.

### Propuesta de solución planteada

1. Simulación del Ataque: Utilizando la biblioteca de Python Scapy, se elabora un "spoofing" de ARP que fabrica y envía paquetes ARP falsificados, simulando ataques de envenenamiento a la caché ARP.
2. Algoritmo de Detección de Ataques ARP: Un sistema implementado en Python y Scapy que monitorea constantemente los paquetes ARP, verificando discrepancias en las direcciones MAC y los patrones de tráfico anómalos. Este proceso clasifica diferentes tipos de ataque, incluyendo DoS y MITM.
3. Medidas Preventivas: Además del sistema de detección, se incluye una estrategia de protección preventiva mediante la incorporación de entradas ARP estáticas en los dispositivos de red para mitigar la posibilidad de envenenamiento.

## SentinelScan: Advanced Network Scanner and Packet Detection Suite

La ciberseguridad de redes destaca la importancia de herramientas avanzadas de detección y defensa proactiva, como SentinelScan, en la protección contra una creciente gama de amenazas cibernéticas. La integración de técnicas como olfateo de paquetes (packet sniffing) y manipulación de paquetes se ha vuelto crucial para la detección temprana de vulnerabilidades y el bloqueo de ataques.

SentinelScan, se basan en frameworks y bibliotecas que permiten el análisis y manipulación de tráfico en capas profundas de la red. La biblioteca Scapy, se ha consolidado como una opción robusta para el manejo de paquetes en Python, permitiendo la creación, el envío y la captura de paquetes personalizados para analizar patrones de tráfico. Este enfoque se combina con técnicas de criptografía y autenticación de red para asegurar la integridad y autenticidad de los datos.

SentinelScan también se construye sobre protocolos fundamentales, abordando desde los niveles de transporte, con TCP y UDP que permiten la transmisión fiable de datos, hasta los protocolos de aplicación, como los protocolos de correo y web, que exigen una mayor precisión en la detección de amenazas.

SentinelScan, se propone un modelo de múltiples capas que abarca diferentes protocolos y verifica de forma continua la legitimidad de los patrones de tráfico en la red. Los algoritmos criptográficos y mecanismos de autenticación integrados facilitan el manejo seguro de datos sensibles y la verificación de identidad, asegurando que cada conexión sea autenticada y confiable.

## Visualization of DDoS Attack using Python Libraries

El crecimiento de los servicios en línea ha expuesto a las organizaciones a mayores riesgos de ataques de Denegación de Servicio Distribuido (DDoS), los cuales pueden saturar la infraestructura de red, provocando la interrupción de servicios, pérdidas económicas y daño reputacional.

Los ataques DDoS han evolucionado tanto en volumen como en complejidad, usando múltiples métodos de inundación como ACK, SYN, PUSH y RESET para maximizar el tráfico de red y obstruir las defensas tradicionales, como cortafuegos y sistemas de detección de intrusiones.

Se han implementado técnicas de análisis y visualización utilizando Python y bibliotecas como Scapy, que permite la captura y manipulación de paquetes. Además, el uso de Wireshark para recopilar archivos “pcap” permite analizar patrones de tráfico en busca de actividad anómala. Estas herramientas, junto con la visualización de datos en HTML y gráficos personalizados, permiten al usuario monitorear múltiples parámetros de tráfico en tiempo real, logrando una visión integral del comportamiento de la red.

La metodología de este enfoque se centra en la detección visual de patrones irregulares, como picos repentinos en el tráfico o el uso anómalo de determinados protocolos, que son indicativos de un ataque en curso.

La capacidad de predecir y visualizar ataques DDoS mejora la seguridad, permitiendo a las organizaciones adaptar sus estrategias de defensa en tiempo real. Al ofrecer una visualización clara y accesible, los analistas pueden tomar decisiones informadas, lo que reduce la dependencia de sistemas automatizados propensos a errores y facilita una respuesta rápida y eficiente.

Este enfoque contribuye significativamente a la ciberseguridad al ofrecer una herramienta visual en tiempo real, basada en Python y Scapy, que permite identificar patrones sospechosos en el tráfico de red.

## Scapy Scripting to Automate Testing of Networking Middleboxes

La adopción de middleboxes como los balanceadores de carga en las corporaciones responde a la necesidad de gestionar y respaldar sus infraestructuras. Estos dispositivos procesan diariamente grandes volúmenes de ancho de banda, abarcando desde protocolos de red hasta protocolos de capa de aplicación. Sin embargo, este tráfico puede incluir datos dañinos o maliciosos, lo cual podría llevar a que estos dispositivos fallen o se vuelvan vulnerables a explotaciones

Para abordar la seguridad y la fiabilidad de estos dispositivos, se utiliza Scapy, una herramienta poderosa e interactiva de manipulación de paquetes desarrollada como un módulo en Python. Python, por su versatilidad como lenguaje de scripting, facilita el uso de Scapy en múltiples aplicaciones. En este caso, se estudia el funcionamiento de Citrix NetScaler ADC mediante la creación de un modelo cliente-servidor para protocolos como FTP, HTTPS y TFTP. Estos scripts, desarrollados con Scapy y compatibles con IPv4 e IPv6, han sido publicados como software de código abierto para facilitar su aplicación en distintas pruebas de red.

Scapy se basa en dos funciones fundamentales: enviar paquetes y recibir respuestas, lo que permite construir paquetes personalizados que abarcan detalles desde la capa de transporte TCP/UDP hasta configuraciones de bajo nivel como el tamaño de ventana o el número de acuse de recibo. Una ventaja significativa de Scapy frente a herramientas como Nmap o hping es que no se limita a paquetes conocidos, sino que permite al usuario escuchar, filtrar, modificar y redirigir paquetes en la red.

### SCAPY- A powerfull interactive packet manipulation program

Scapy admite múltiples protocolos de red, desde las capas bajas (como IP y ARP) hasta protocolos de aplicación. Además, permite manipular de manera precisa las características de los paquetes, lo que es útil en diversas tareas de seguridad, como pruebas de penetración y detección de vulnerabilidades.

Scapy también es compatible con herramientas de monitoreo de red como Wireshark, permitiendo a los investigadores rastrear en tiempo real los paquetes enviados y recibidos en un entorno visual y detallado.

Entre sus puntos fuertes, Scapy destaca por su capacidad para manejar tareas complejas de redes que suelen requerir herramientas dedicadas, como filtrado avanzado, trazado de rutas y pruebas de conectividad. Sin embargo, Scapy tiene un soporte limitado para protocolos más nuevos y enfrenta restricciones de rendimiento cuando se utiliza en redes de gran escala.



## Introducción

Un Sniffer de red, también conocido como analizador de paquetes, es una herramienta de software o hardware diseñada para monitorizar y capturar el tráfico de datos que circula por una red. Su función principal es permitir a los administradores y profesionales de TI observar y analizar la información que se envía y recibe a través de la red.

### Scapy como librería de Python

Scapy destaca debido a su enfoque en el control preciso de paquetes. Mientras que otras librerías como socket y requests permiten establecer conexiones y manejar protocolos a nivel de aplicación, Scapy opera en un nivel más bajo, ofreciendo capacidades de construcción y manipulación de paquetes a nivel de red.

### Características principales de Scapy

Scapy permite realizar múltiples tareas en redes, como:

1. Construcción y manipulación de paquetes: Scapy facilita la creación de paquetes personalizados, especificando cada campo de los diferentes protocolos (TCP, UDP, IP, ARP, ICMP, etc.).
2. Envío y recepción de paquetes: Scapy permite enviar y recibir paquetes directamente desde Python, lo que es ideal para realizar pruebas de conexión y análisis de red en tiempo real.
3. Sniffing de red: Con Scapy, es posible ver o escuchar el tráfico de red, filtrando y capturando paquetes específicos, lo cual es muy valioso para el análisis de tráfico en ciberseguridad y la detección de intrusiones.
4. Análisis y manipulación: Ofrece funciones para modificar y analizar los paquetes recibidos, permitiendo realizar transformaciones, reenviar tráfico, o investigar patrones específicos.

## Objetivo

El objetivo de este proyecto es crear un Sniffer implementado con Python y la librería Scapy, aplicando los conocimientos previos de programación y el conocimiento sobre los protocolos de transmisión en específico los protocolos UDP y TCP que están montados sobre IP, además de esto, el Sniffer deberá de ser capaz de poder filtrar por protocolos cambiando una ejecución en el main del programa y este podrá desplegar en terminal la información básica de la transmisión de datos y exportara los resultados en un archivo pcap que podrá ejecutarse con Wireshark para una mejor visualización.

## Desarrollo

Para comenzar el proyecto se reviso la documentación de Scapy, en donde se decidió que el sistema operativo en donde se ejecutaría el Sniffer seria Kali Linux, pues en sistemas Linux Scapy viene precargado, sin embargo, para ejecutarlo en un sistema Windows se debe instalar la librería.

```

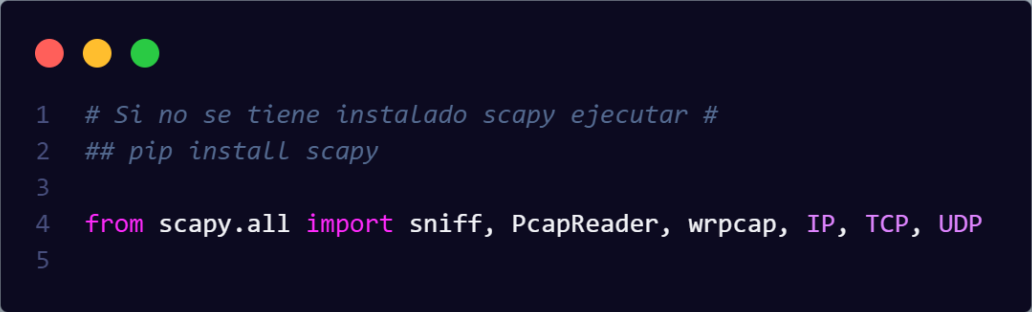
aSPY//YASa
apyyyyCY/////////YCa
sY////YSpcs  scpCY//Pp
ayp ayyyyyySCP//Pp      syY//C
AYAsAYYYYYY//Ps      cY//S
pCCCY//p      cSSps y//Y
SPPPP//a      pP//AC//Y
A//A      cyP///C
p//Ac      sC//a
P///YCpc      A//A
sccccp//pSP//p      p//Y
sY/////////y  caa      S//P
cayCyayP//Ya      pY/Ya
sY/PsY///YCc      aC//Yp
sc  sccaCY//PCypaapyCP//YSs
spCPY/////////YPSps
ccaacs
|
| Welcome to Scapy
| Version 2.6.1
|
| https://github.com/secdev/scapy
|
| Have fun!
|
| Wanna support scapy? Star us on GitHub!
|                                     -- Satoshi Nakamoto
|
```

## Requisitos para Scapy

```

1  # Proyecto_Redex_Sniffer
2
3  # scapy opera de forma normal en linux
4
5  # Si no es una maquina linux se debe instalar Scapy
6
7  # Instruccion de instalación "pip install scapy"
8
9  # Es necesario tener instalado Python a partir de la version 2.5.0+
```

## Importación de librerías



```
1  # Si no se tiene instalado scapy ejecutar #
2  ## pip install scapy
3
4  from scapy.all import sniff, PcapReader, wrpcap, IP, TCP, UDP
5
```

Como se observa de todo lo que es Scapy únicamente se utilizan 5 librerías:

Sniff: Esta librería permite capturar paquetes de en una interfaz específica.

PcaReader: Nos permite poder leer archivos con la extensión pcap, que son los mismos con los que trabaja Wireshark.

Wrpcap: Nos permite hacer la exportación de un archivo con extensión pcap, lo que lo hace ejecutable en Wireshark.

IP, TCP, UDP: Son los protocolos que se van a examinar.

## Definición de la clase Sniffer

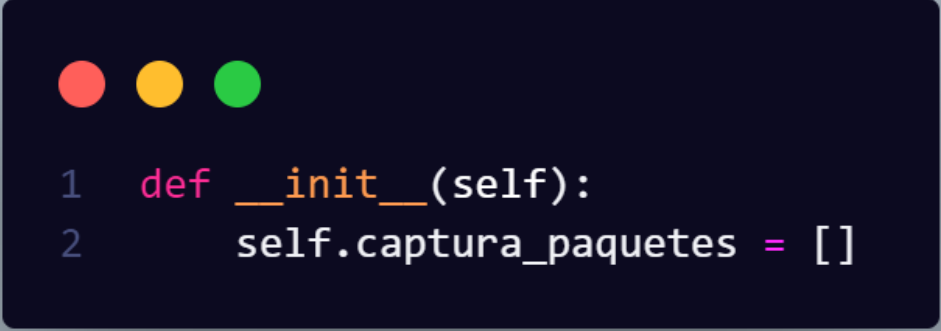
```
1 class Sniffer:
2
3     def __init__(self):
4         self.captura_paquetes = []
5
6     def iniciar_captura (self, interface="eth0", filter=""):
7
8         print ("Captura iniciada. Presionar CTRL+C para detener la captura")
9
10        try:
11            self.captura_paquetes = sniff(iface=interface, filter=filter, prn=lambda x:x.summary(),store=True)
12
13        except KeyboardInterrupt:
14            print(f"Captura finalizada. Se capturaron {len(self.captura_paquetes)}")
15
16    def leer_paquetes (self, pcapfile):
17        try:
18            self.captura_paquetes = [pkt for pkt in PcapReader(pcapfile)]
19            print(f"Lectura del archivo {pcapfile} correcta")
20
21        except Exception as e:
22            print(f"Error al leer el archivo {pcapfile}: {e}")
23
24
25    def filtro_por_protocolo (self, protocol):
26        filtrado_de_paquetes = [pkt for pkt in self.captura_paquetes if pkt.haslayer(protocol)]
27        return filtrado_de_paquetes
28
29    def imprimir_paquetes(self, packets=None):
30        if packets is None:
31            packets = self.captura_paquetes
32
33        print("Resumen de paquetes:")
34        for i, packet in enumerate(packets, 1):
35            # Verificar si el paquete tiene capa IP
36            if IP in packet:
37                protocol = "TCP" if TCP in packet else "UDP" if UDP in packet else "Otro"
38                print(f"{i}: Time: {packet.time:.6f} | Source: {packet[IP].src} | "
39                    f"Destination: {packet[IP].dst} | Protocol: {protocol} | Length: {len(packet)}")
40            else:
41                print(f"{i}: Paquete sin capa IP.")
42        print("Fin del resumen.")
43
44    def exportar (self, packets, filename = "captura.pcap"):
45        wrpcap(filename, packets)
46        print("Paquetes guardados con éxito")
```

Esta clase tendrá todas las funciones que se ejecutan para poder capturar los paquetes.

## Definiciones

### Inicialización

El programa inicia con la declaración de una lista vacía en donde se guardarán todos los paquetes capturados.



```
1 def __init__(self):  
2     self.captura_paquetes = []
```

### Inicio de la captura

Posteriormente se inicia la captura, en donde se reciben como parámetros la lista vacía, la interfaz en donde se van a capturar los paquetes de información (Por defecto eth0) y por último el filtro (en este caso vacío porque se declara el protocolo buscado en la función main).

Además, la forma de truncar el analizador y detener la captura hay que presionar la combinación Ctrl + C.

Cabe aclarar que la función para capturar los paquetes busca por interfaz, filtro y con lambda hace que se muestren los detalles de cada paquete.

```

1 def iniciar_captura (self, interface="eth0", filter=""):
2
3     print ("Captura iniciada. Presionar CTRL+C para detener la captura")
4
5     try:
6         self.captura_paquetes = sniff(iface=interface, filter=filter, prn=lambda x:x.summary(),store=True)
7
8     except KeyboardInterrupt:
9         print(f"Captura finalizada. Se capturaron {len(self.captura_paquetes)}")
10

```

### Lectura de los paquetes

Para leer los paquetes se pasan como argumentos la lista que ahora ya no está vacía porque ya tiene paquetes capturados y se le pasa el documento pcap que ya fue exportado.

```

1 def leer_paquetes (self, pcapfile):
2     try:
3         self.captura_paquetes = [pkt for pkt in PcapReader(pcapfile)]
4         print(f"Lectura del archivo {pcapfile} correcta")
5
6     except Exception as e:
7         print(f"Error al leer el archivo {pcapfile}: {e}")
8

```

### Filtro por protocolo

La función más importante al menos para este proyecto es justamente el filtrado por protocolos justamente porque solo se busca capturar paquetes TCP y UDP que están montados sobre IP.

Aunque el código es relativamente corto la función es muy importante, pues nuevamente recibe la lista que ya tiene capturados los paquetes y se indica cual es el protocolo que se quiere extraer, para esto nos apoyamos de la función `haslayer`

que contiene Scapy, pues esta función nos permite hacer un filtrado específico, en este caso por protocolo.

```
1 def filtro_por_protocolo (self, protocol):
2     filtrado_de_paquetes = [pkt for pkt in self.captura_paquetes if pkt.haslayer(protocol)]
3     return filtrado_de_paquetes
4
```

### Impresión de los paquetes

Para la impresión únicamente se pasa la lista y mientras se cumpla la condición de filtrado regresara los paquetes.

```
1 def imprimir_paquetes(self, packets=None):
2     if packets is None:
3         packets = self.captura_paquetes
```

Sin embargo, para tener un resumen de los paquetes modifique la función para que la lectura que se muestra en terminal fuera de una forma más legible.



```

1 print("Resumen de paquetes:")
2 for i, packet in enumerate(packets, 1):
3     # Verificar si el paquete tiene capa IP
4     if IP in packet:
5         protocol = "TCP" if TCP in packet else "UDP" if UDP in packet else "Otro"
6         print(f"{i}: Time: {packet.time:.6f} | Source: {packet[IP].src} | "
7               f"Destination: {packet[IP].dst} | Protocol: {protocol} | Length: {len(packet)}")
8     else:
9         print(f"{i}: Paquete sin capa IP.")
10 print("Fin del resumen.")
11

```

### Exportación del archivo pcap

Para un mejor análisis de los paquetes capturados nos apoyamos de la librería wrpcap, para poder escribir los resultados en un archivo compatible con Wireshark, pues el Sniffer que se crea solo muestra datos básicos, pero intención es que se vean de una forma más gráfica que solo en terminal.


```

1 def exportar (self, packets, filename = "captura.pcap"):
2     wrpcap(filename, packets)
3     print("Paquetes guardados con éxito")

```


### Función principal main

Para la función principal main se debe importar el documento de todo el Sniffer, así como importar la clase e iniciar la instancia del Sniffer para poder capturar.



```
1  from Sniffer_Scapy import Sniffer
2
3  def main():
4
5      sniffer = Sniffer()
```

La parte importante para el proceso de main es la forma en que se utiliza el filtrado por protocolo, al momento de iniciar la captura se debe poner la interfaz utilizada con normalidad en el equipo en donde se ejecuta el Sniffer, además de lo más importante que es el filtro, en este caso únicamente TCP y UDP.



```
1  sniffer.iniciar_captura(interface = "Wi-Fi", filter="tcp or udp")
2
3  paquetes_tcp = sniffer.filtro_por_protocolo ("TCP")
4  paquetes_upd = sniffer.filtro_por_protocolo("UDP")
5
6  total_de_paquetes = paquetes_tcp + paquetes_upd
```

Así como asignar de forma independiente los protocolos en una variable específica para cada uno y el total de estos.

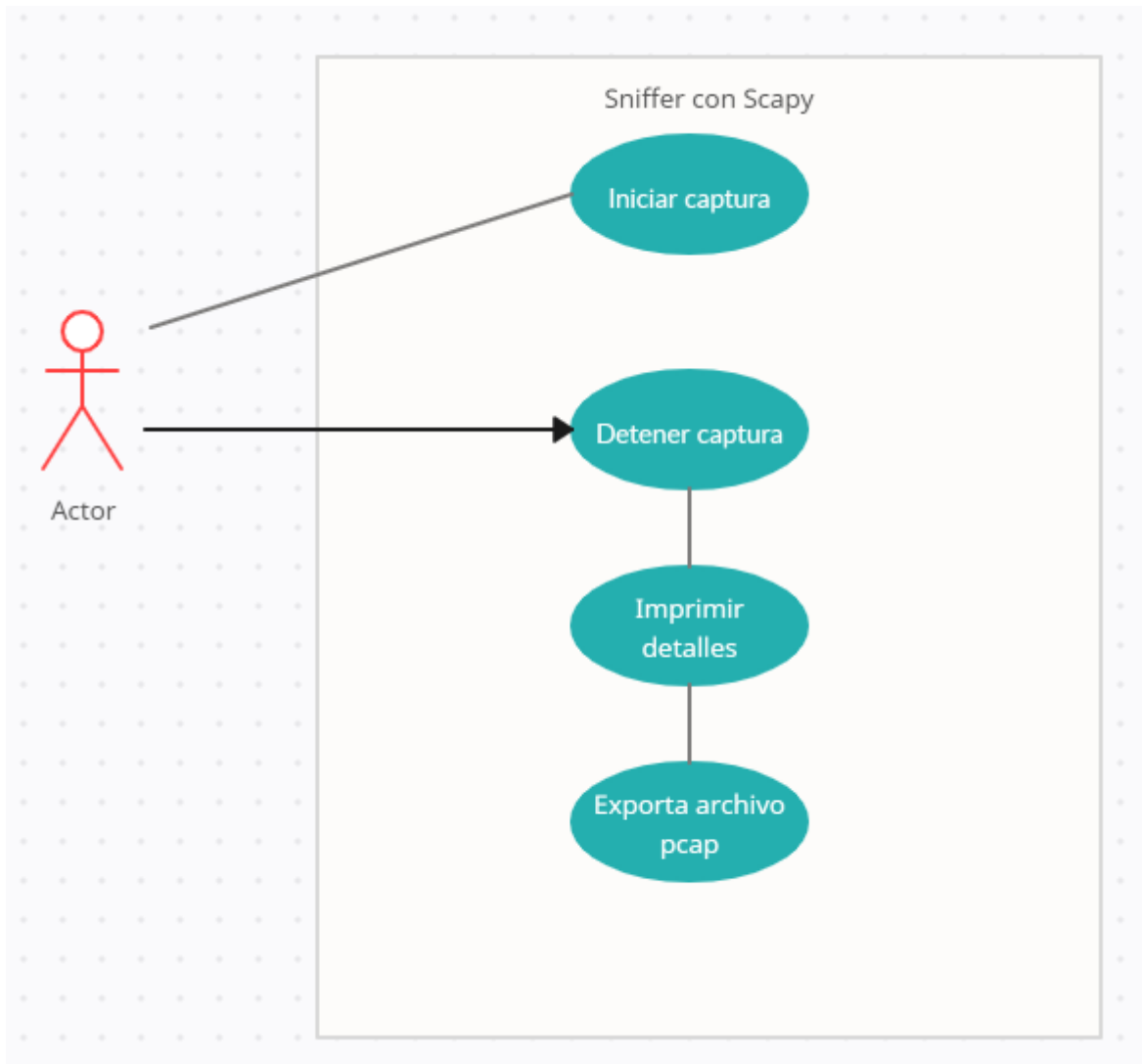
Por último, para imprimir los paquetes se manda a llamar a la función imprimir, la cual ya recibe los parámetros por el otro archivo, en caso de no detectar protocolos

de los que se filtran, únicamente imprime que no se encontró ninguno, para finalmente mostrar cuantos protocolos se encontraron.

```
1     if total_de_paquetes:
2         print("Detalles de los paquetes:")
3         sniffer.imprimir_paquetes(total_de_paquetes)
4     else:
5         print("No se capturaron paquetes UDP ni TCP")
6
7     sniffer.exportar (total_de_paquetes, filename="Captura_TCP_&_UDP.pcap")
8     print("Paquetes exportados a 'Captura_TCP_&_UDP.pcap'.")
9
10    print(f"Total paquetes: {len(total_de_paquetes)}")
11    print(f"Total paquetes TCP: {len(paquetes_tcp)}")
12    print(f"Total paquetes UDP: {len(paquetes_upd)}")
13
14    if __name__ == "__main__":
15        main()
16
```

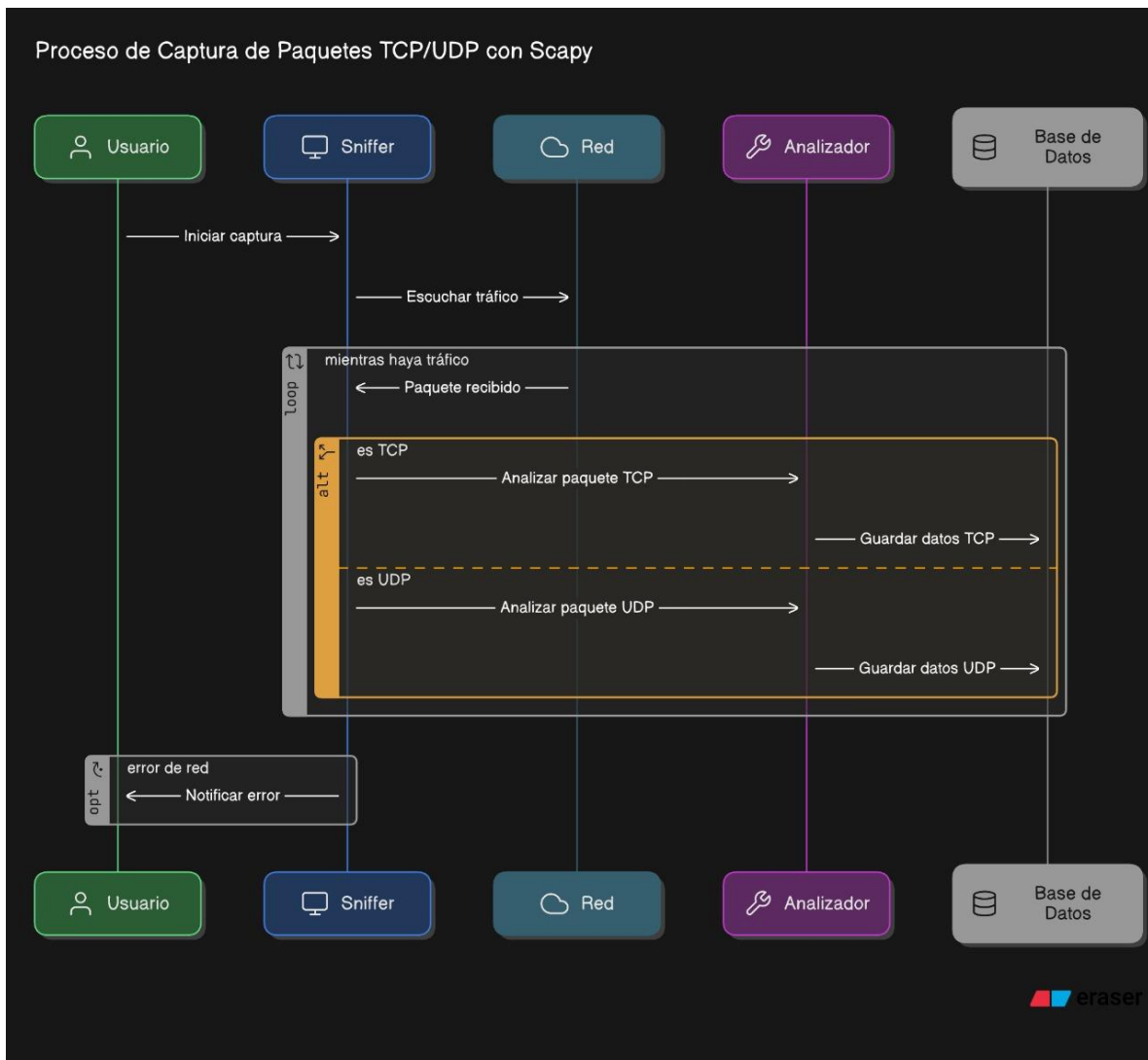
## Gráficos

### Diagrama de casos de uso UML



El usuario puede iniciar captura, así como detenerla, pero únicamente cuando se detiene la captura se pueden imprimir los detalles de los paquetes capturados, así como la exportación del archivo pcap para su análisis posterior

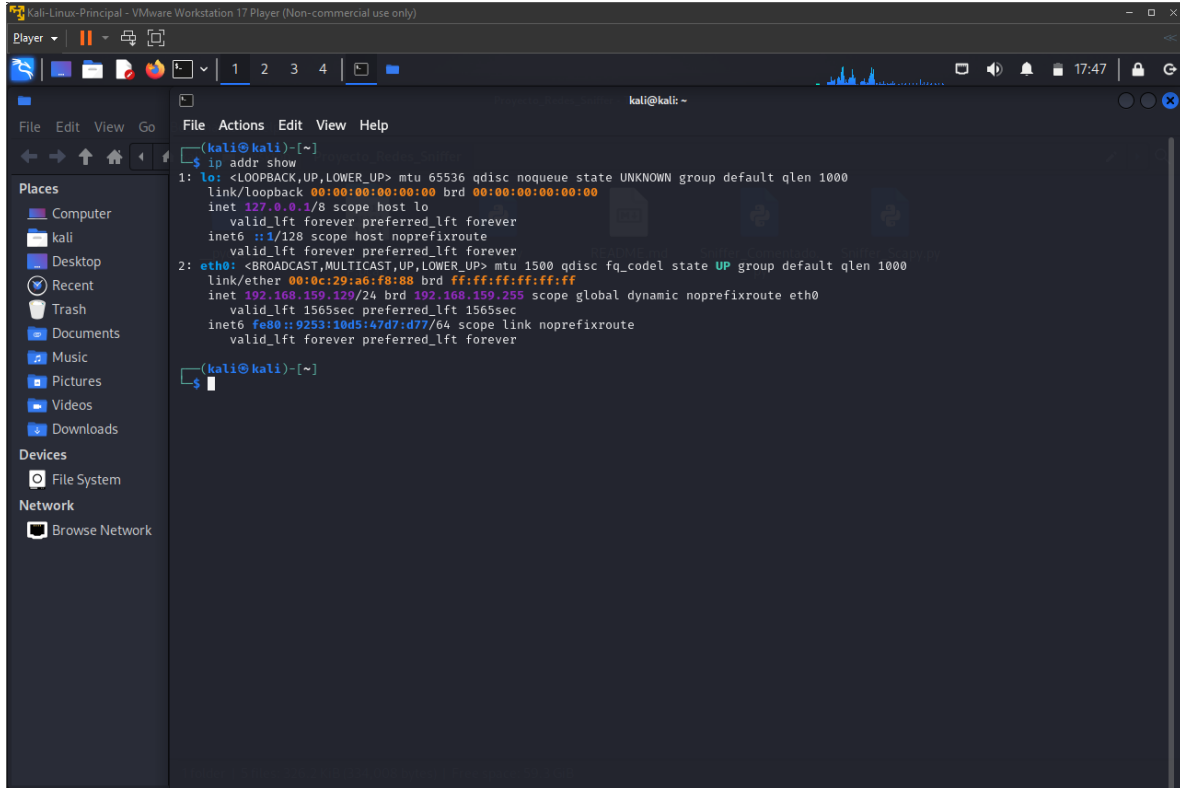
## Diagrama de secuencias



1. Inicio de captura: El usuario solicita iniciar el sniffer a través de la terminal con permisos de administrador.
2. Captura de paquetes: El sniffer escucha en la red y captura los paquetes.
3. Procesamiento: Los paquetes capturados se envían al analizador para su decodificación.
4. Almacenamiento: Los datos procesados se guardan en un archivo pcap (cabe aclarar que en el diagrama principal se mostraba una base de datos que se sustituyó por un archivo pcap y un cache que genera python).

## Pruebas de funcionamiento

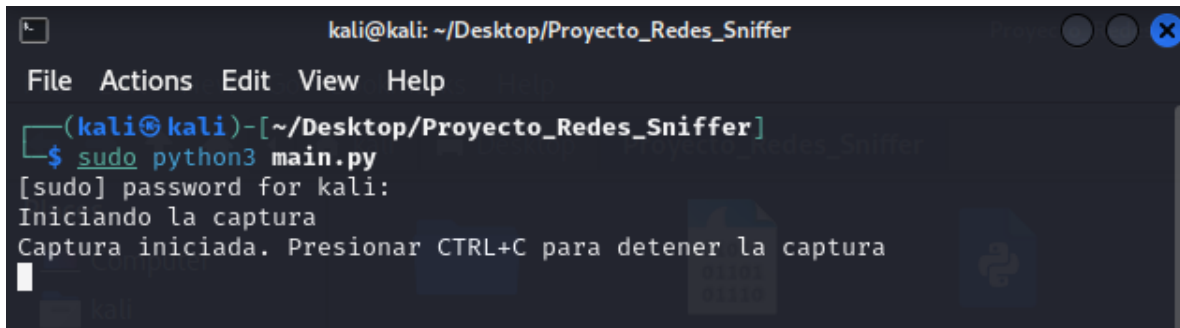
Para probar el funcionamiento del Sniffer, se montó una máquina virtual Kali Linux, por lo tanto, las pruebas están realizadas en este sistema operativo.



The screenshot shows a Kali Linux terminal window with the title "kali@kali: ~". The terminal output displays the command `ip addr show` and its results for two network interfaces: `lo` (loopback) and `eth0` (ethernet). The `lo` interface has an IP of `127.0.0.1` and a MAC of `00:00:00:00:00:00`. The `eth0` interface has an IP of `192.168.159.129` and a MAC of `00:0c:29:a6:f8:88`. The terminal also shows the `valid_lft` and `preferred_lft` values for each interface.

```
(kali@kali)-[~]  
$ ip addr show  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host noprefixroute  
        valid_lft forever preferred_lft forever  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 00:0c:29:a6:f8:88 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.159.129/24 brd 192.168.159.255 scope global dynamic noprefixroute eth0  
        valid_lft 1565sec preferred_lft 1565sec  
    inet6 fe80::9253:10d5:47d7:d77/64 scope link noprefixroute  
        valid_lft forever preferred_lft forever  
(kali@kali)-[~]  
$
```

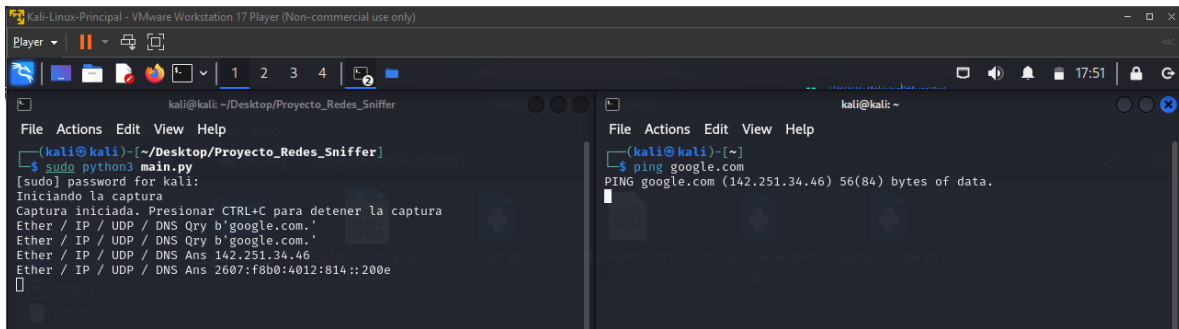
Ejecución del programa con permisos del administrador.



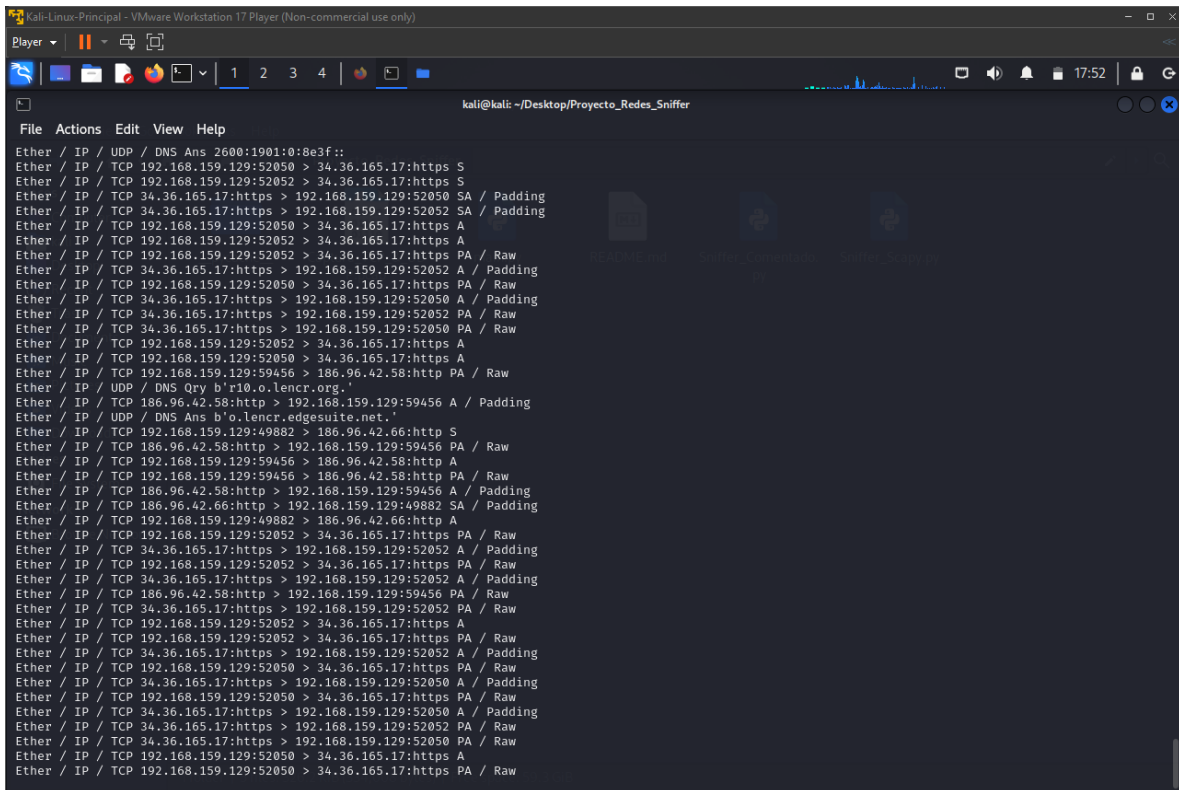
The screenshot shows a Kali Linux terminal window with the title "kali@kali: ~/Desktop/Proyecto\_Redes\_Sniffer". The terminal output displays the command `sudo python3 main.py` and its results. The prompt `[sudo] password for kali:` is shown, followed by the message "Iniciando la captura" (Starting the capture) and "Captura iniciada. Presionar CTRL+C para detener la captura" (Capture started. Press CTRL+C to stop the capture). The terminal also shows the `valid_lft` and `preferred_lft` values for each interface.

```
kali@kali: ~/Desktop/Proyecto_Redes_Sniffer  
File Actions Edit View Help  
(kali@kali)-[~/Desktop/Proyecto_Redes_Sniffer]  
$ sudo python3 main.py  
[sudo] password for kali:  
Iniciando la captura  
Captura iniciada. Presionar CTRL+C para detener la captura  
$
```

## Pruebas de detección

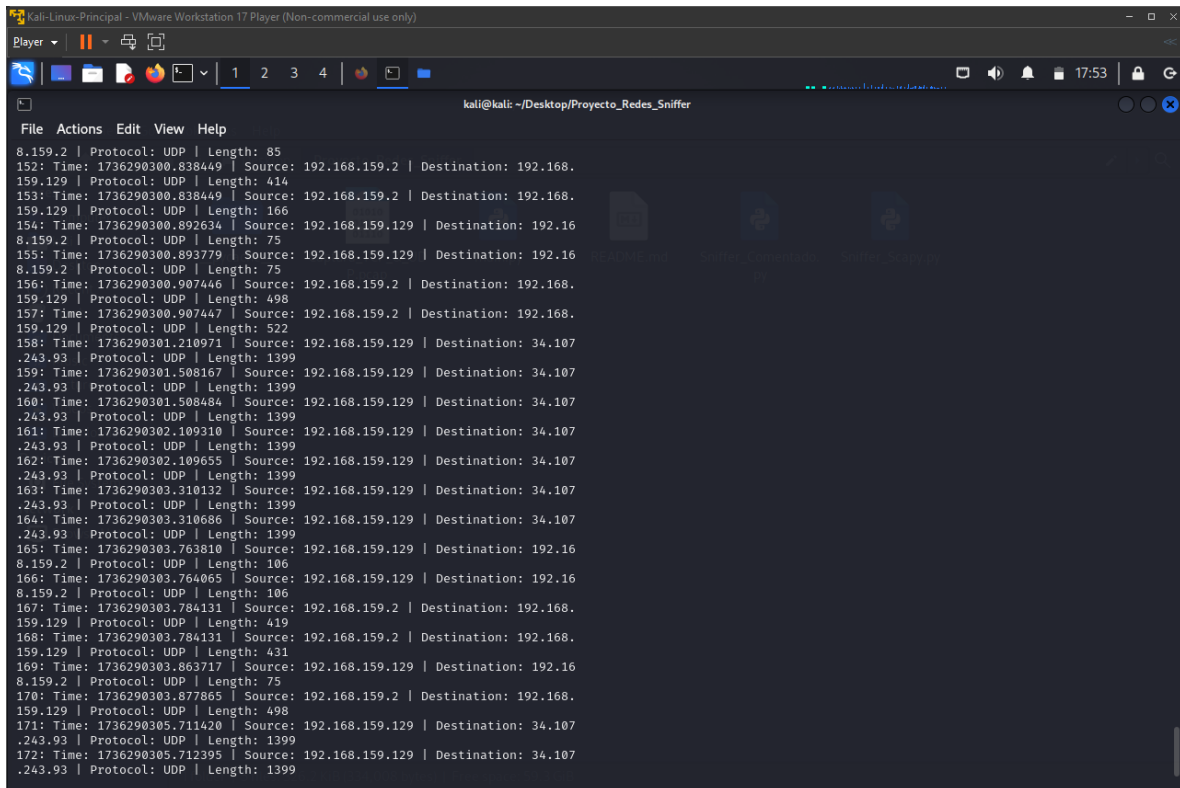


```
kali@kali: ~/Desktop/Proyecto_Redes_Sniffer
File Actions Edit View Help
(kali@kali)-[~/Desktop/Proyecto_Redes_Sniffer]
└─$ sudo python3 main.py
[sudo] password for kali:
Iniciando la captura
Captura iniciada. Presionar CTRL+C para detener la captura
Ether / IP / UDP / DNS Qry b'google.com.'
Ether / IP / UDP / DNS Qry b'google.com.'
Ether / IP / UDP / DNS Ans 142.251.34.46
Ether / IP / UDP / DNS Ans 2607:f8b0:4012:814::200e
└─$ ping google.com
PING google.com (142.251.34.46) 56(84) bytes of data.
```



```
kali@kali: ~/Desktop/Proyecto_Redes_Sniffer
File Actions Edit View Help
Ether / IP / UDP / DNS Ans 2600:1901:0:8e3f::
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https S
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https S
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52050 SA / Padding
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 SA / Padding
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https A
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https A
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 A / Padding
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52050 A / Padding
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52050 PA / Raw
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https A
Ether / IP / TCP 192.168.159.129:52050 > 186.96.42.58:http PA / Raw
Ether / IP / UDP / DNS Qry b'r10.o.lencr.org.'
Ether / IP / TCP 186.96.42.58:http > 192.168.159.129:59456 A / Padding
Ether / IP / UDP / DNS Ans b'o.lencr.edgesuite.net.'
Ether / IP / TCP 192.168.159.129:49882 > 186.96.42.66:http S
Ether / IP / TCP 186.96.42.58:http > 192.168.159.129:59456 PA / Raw
Ether / IP / TCP 192.168.159.129:59456 > 186.96.42.58:http A
Ether / IP / TCP 192.168.159.129:59456 > 186.96.42.58:http PA / Raw
Ether / IP / TCP 186.96.42.58:http > 192.168.159.129:59456 A / Padding
Ether / IP / TCP 186.96.42.66:http > 192.168.159.129:49882 SA / Padding
Ether / IP / TCP 192.168.159.129:49882 > 186.96.42.66:http A
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 A / Padding
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 A / Padding
Ether / IP / TCP 186.96.42.58:http > 192.168.159.129:59456 PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 PA / Raw
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https A
Ether / IP / TCP 192.168.159.129:52052 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 A / Padding
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52050 A / Padding
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52050 A / Padding
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52052 PA / Raw
Ether / IP / TCP 34.36.165.17:https > 192.168.159.129:52050 PA / Raw
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https A
Ether / IP / TCP 192.168.159.129:52050 > 34.36.165.17:https PA / Raw
```

## Impresión de detalles



```
kali@kali: ~/Desktop/Proyecto_Redes_Sniffer
File Actions Edit View Help
81: Time: 1736290300.838449 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 85
152: Time: 1736290300.838449 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 414
153: Time: 1736290300.838449 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 166
154: Time: 1736290300.892634 | Source: 192.168.159.129 | Destination: 192.168.159.2 | Protocol: UDP | Length: 75
155: Time: 1736290300.893779 | Source: 192.168.159.129 | Destination: 192.168.159.2 | Protocol: UDP | Length: 75
156: Time: 1736290300.907446 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 498
157: Time: 1736290300.907447 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 522
158: Time: 1736290301.210971 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
159: Time: 1736290301.508167 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
160: Time: 1736290301.508484 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
161: Time: 1736290302.109310 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
162: Time: 1736290302.109655 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
163: Time: 1736290303.310132 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
164: Time: 1736290303.310686 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
165: Time: 1736290303.763810 | Source: 192.168.159.129 | Destination: 192.168.159.2 | Protocol: UDP | Length: 106
166: Time: 1736290303.764065 | Source: 192.168.159.129 | Destination: 192.168.159.2 | Protocol: UDP | Length: 106
167: Time: 1736290303.784131 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 419
168: Time: 1736290303.784131 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 431
169: Time: 1736290303.863717 | Source: 192.168.159.129 | Destination: 192.168.159.2 | Protocol: UDP | Length: 75
170: Time: 1736290303.877865 | Source: 192.168.159.2 | Destination: 192.168.159.129 | Protocol: UDP | Length: 498
171: Time: 1736290305.711420 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
172: Time: 1736290305.712395 | Source: 192.168.159.129 | Destination: 34.107.243.93 | Protocol: UDP | Length: 1399
```

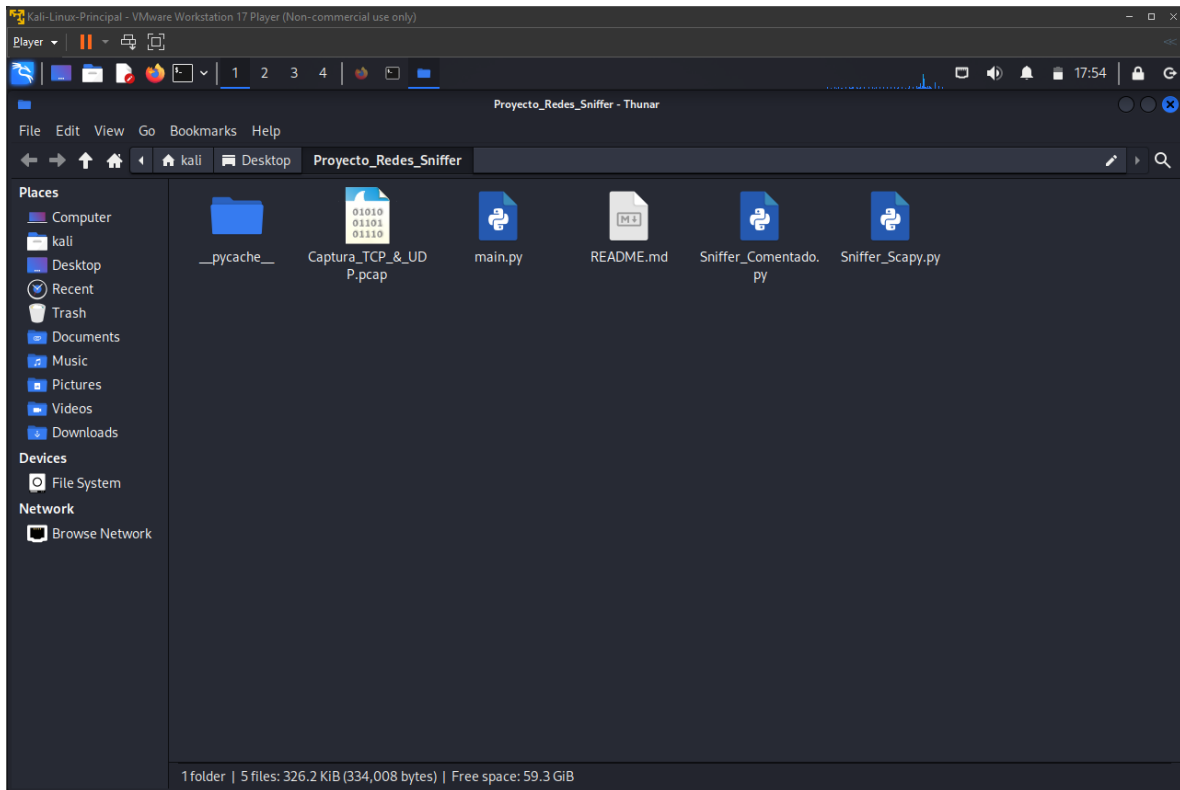
## Recopilación

```
Fin del resumen.
Paquetetes guardados con exito
Paquetes exportados a 'Captura_TCP_&_UDP.pcap'.
Total paquetes: 172
Total paquetes TCP: 121
Total paquetes UDP: 51
```

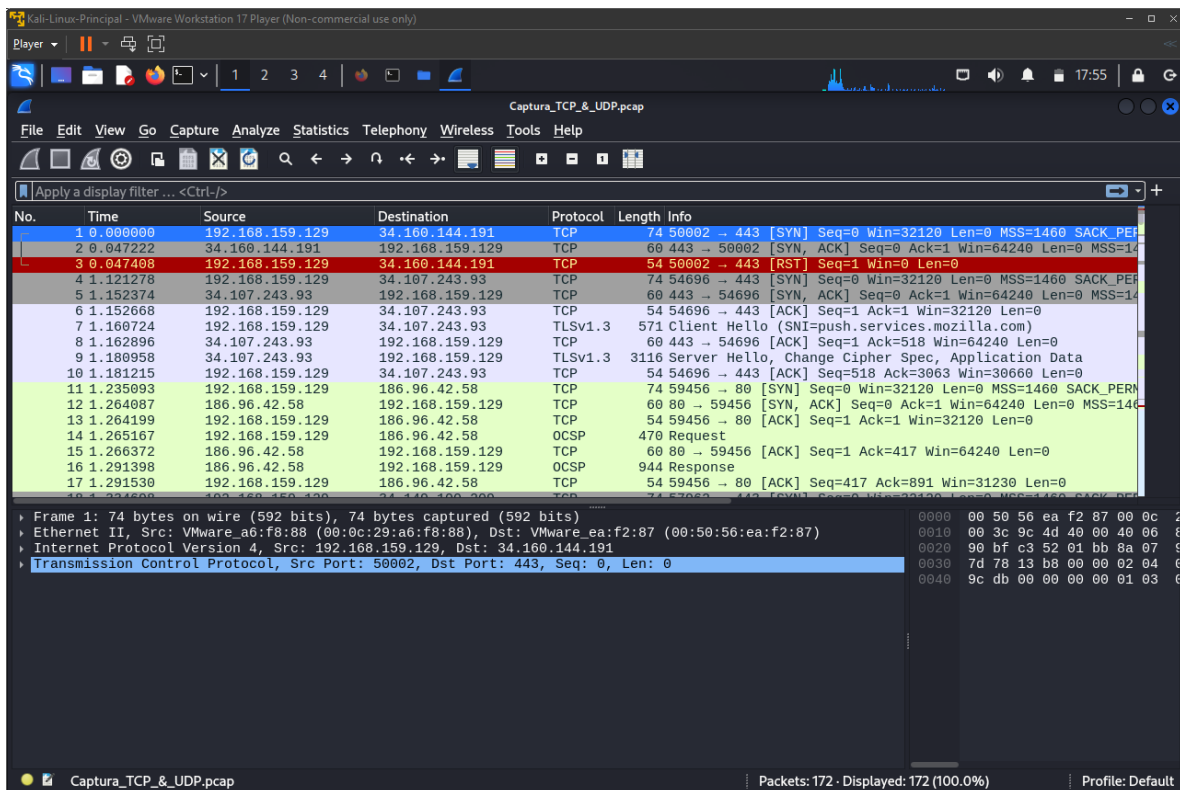
```
(kali@kali)-[~/Desktop/Proyecto_Redes_Sniffer]
$
```



## Prueba de exportación



## Prueba de captura en Wireshark



## Conclusiones

En conclusión, este proyecto me ayudo bastante en diversos aspectos, pues tuve que hacer investigaciones previas para saber en que se usaba Scapy, lo que me pareció muy interesante al saber el potencial que tiene en el ámbito de la ciberseguridad, por parte de la materia de redes fue interesante saber como los protocolos pueden ser infectados para robar información y como es que el encargado de redes tiene que saber utilizar herramientas como lo es Scapy para poder atender esas emergencias. Fue una buena experiencia que me sirve para la formación que estoy buscando en el ámbito de seguridad además de que esto complementa parte de un curso de cisco que estoy tomando.

## Bibliografía

- D. R. Thomas, P. V, W. Nancy, G. Sowmiya, A. T. P and V. Peroumal, "Detection and Prevention of Poisoning Targets with ARP Cache using Scapy," 2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), Bangalore, India, 2024, pp. 1-6, doi: 10.1109/IITCEE59897.2024.10467270. keywords: {Protocols;Machine learning algorithms;Network security;Maintenance engineering;Logic gates;Media Access Protocol;Reliability;ARP;MAC Address;Internet Protocols;ARP spoofing;Linux;MITM},
- T. Seshapriyan, S. M. Dinesh and J. Godwin Ponsam, "SentinelScan: Advanced Network Scanner and Packet Detection Suite," 2024 8th International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2024, pp. 253-258, doi: 10.1109/ICISC62624.2024.00050. keywords: {Protocols;Documentation;Network security;User interfaces;Traffic control;Data transfer;Robustness;Network security;SentinelScan;Cyber threats;Packet manipulation;Real-time packet sniffing;Scapy;Cryptographic algorithms;Authentication mechanisms;Secure communication protocols;Security standards},
- S. T, M. G K y A. Santhosh R A, "Visualization of DDoS Attack using Python Libraries", 2024 Conferencia Internacional sobre Tendencias en Tecnologías de Computación Cuántica y Tecnologías Empresariales Emergentes, Pune, India, 2024, pp. 1-5, doi: 10.1109/TQCEBT59414.2024.10545204. palabras clave: Economía;Computación de Quantum;Visorización de datos;Seguridad de la red;Ataque de la vida de servicio;Investigación de mercados;Bibliotes;Atapas deDDodios;Python librarys;Scapy;Intruyección,
- Moharir, M., Adyathimar, K. B., Shobha, G., & Soni, V. (2020). Scapy scripting to automate testing of networking middleboxes. *Advances in Science, Technology and Engineering Systems Journal*, 5(2), 293-298.
- R. R. S, R. R, M. Moharir and S. G, "SCAPY- A powerful interactive packet manipulation program," 2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS), Bangalore, India, 2018, pp. 1-

5, doi: 10.1109/ICNEWS.2018.8903954. keywords:  
{capture;packet;protocol;python;scapy;tool}.

- *Welcome to Scapy's documentation! — Scapy 2.6.1 documentation.* (s. f.).  
<https://scapy.readthedocs.io/en/latest/>