Introducción del código

Este proyecto consiste en un escáner de red programado en Python que permite analizar dispositivos conectados a una red. Su objetivo principal es identificar qué equipos están activos, verificar si tienen puertos abiertos (como los usados para servicios web o conexiones remotas) y estimar su sistema operativo.

El programa está diseñado para ser eficiente, ya que utiliza hilos para realizar múltiples verificaciones al mismo tiempo, acelerando el proceso de escaneo. Además, cuenta con una interfaz gráfica sencilla donde el usuario puede ingresar una dirección IP y una máscara de red para iniciar el análisis.

Esta herramienta es útil tanto para administradores de red que necesitan diagnosticar conexiones. El código combina bibliotecas como socket para conexiones de red, scapy para el envío de paquetes y tkinter para la interfaz visual, demostrando un enfoque práctico para el desarrollo de software de red.

Detalles específicos del código

Importación de Bibliotecas

Texto

El contenido generado por IA puede ser incorrecto.

tkinter: Es una biblioteca para crear ventanas, botones y toda la interfaz visual del programa.

scrolledtext: Permite crear un cuadro de texto con barra de desplazamiento (para ver todos los resultados del escaneo).

threading: Permite hacer varias cosas al mismo tiempo (como escanear múltiples puertos o hosts en paralelo)

socket: Se usa para comunicarse con otros dispositivos en la red (como cuando intentamos abrir una conexión a un puerto).

ipaddress: Ayuda a manejar direcciones IP y calcular rangos de red (por ejemplo, saber todas las IPs de una red como 192.168.1.0/24).

scapy: Es una herramienta avanzada para enviar y recibir paquetes de red (como pings para detectar hosts activos

Diccionario de Puertos Comunes

Texto

El contenido generado por IA puede ser incorrecto.

Esta parte del código es para especificar cuales son los puertos que va a escanear

Este diccionario define los puertos más usados en redes y qué servicio suelen tener.

Por ejemplo:

* + Puerto 21 → FTP (transferencia de archivos).
  + Puerto 80 → HTTP (páginas web normales).
  + Puerto 443 → HTTPS (páginas web seguras).

El programa solo escaneará estos puertos para ver si están abiertos o cerrados. Y posteriormente nos dira que servicio tiene el puerto

Detección del Sistema Operativo

Texto

El contenido generado por IA puede ser incorrecto.

En esta parte del código, se le ordena al código que nos diga en que sistema operativo está el host activo, ya sea Windows , Linux , Mac OS ,etc .

Esta función envía un ping (ICMP) al dispositivo y analiza el TTL (Time To Live) de la respuesta.

Cada sistema operativo tiene un TTL por defecto:

* + Linux/Unix → TTL ≈ 64
  + Windows → TTL ≈ 128
  + Routers o BSD → TTL ≈ 255

Así el programa puede adivinar qué sistema operativo tiene el dispositivo escaneado.

Escaneo de Puertos

Texto

El contenido generado por IA puede ser incorrecto.

Esta parte del código esta encargada de dar la orden de ver que puertos están abiertos. Intenta conectarse a cada puerto del diccionario. Si la conexión tiene éxito (result == 0), el puerto está abierto. Si falla, el puerto está cerrado.

Para aumentar la velocidad del escaneo usamos threading Para escanear varios puertos al mismo tiempo (más rápido que uno por uno).

Escaneo Completo de Red

Texto

El contenido generado por IA puede ser incorrecto.

Esta parte del código se encarga de buscar todos los dispositivos activos adema Calcula todas lasIPsdelared (ejemplo: 192.168.1.0/24 → 192.168.1.1, 192.168.1.2, etc.). Envía un ping a cada IP para ver cuáles están activas. Para cada host activo, escanea sus puertos y detecta su sistema operativo. Muestra los resultados en la ventana.

Interfaz Gráfica

Texto

El contenido generado por IA puede ser incorrecto.

Esta parte del código se encarga de Crear una ventana con campos para ingresar IP y máscara. Tiene un botón Escanear. Que inicia el proceso. Muestra los resultados en un cuadro de texto con scroll.

Texto

El contenido generado por IA puede ser incorrecto.

Y acá tenemos el accionar del botón para que comience a ejecutar el escaneo

Función Principal

Texto, Logotipo

El contenido generado por IA puede ser incorrecto.

Este es el comando esencial para poder ejecutar la interfaz grafica

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Ejecución del código

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Primero colocamos nuestra dirección ip y luego nuestra mascara de red (como muestra el ejemplo en la foto )

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Luego de colocarlo nos comenzará a escanear los hosts activos y los no activos

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Después de identificar qué equipos están activos en la red, el programa nos mostrará detalles importantes de cada uno:

* Dirección IP del dispositivo activo
* Puertos abiertos, indicando su número y servicio asociado (ej: "Puerto 21 - FTP")
* Sistema operativo detectado (Windows, Linux, etc.)

Esto nos ayuda a conocer qué servicios están disponibles en cada equipo y qué tipo de sistema están usando.

Código entero

import tkinter as tk

from tkinter import scrolledtext

import threading

import socket

from ipaddress import IPv4Network, ip\_network, AddressValueError, NetmaskValueError

from scapy.all import sr1, IP, ICMP

# Diccionario de puertos comunes

ports = {

    21: 'FTP',

    22: 'SSH',

    23: 'Telnet',

    80: 'HTTP',

    443: 'HTTPS',

    3389: 'RDP',

    135: 'Microsoft RPC',

    445: 'SMB'

}

# Obtener red local desde IP obtenida automáticamente

def get\_local\_network():

    try:

        hostname = socket.gethostname()

        local\_ip = socket.gethostbyname(hostname)

        return ip\_network(local\_ip + '/24', strict=False)

    except Exception:

        return None

# Detección del sistema operativo por TTL

def detect\_os(ip):

    ttl = None

    try:

        pkt = IP(dst=ip)/ICMP()

        reply = sr1(pkt, timeout=1, verbose=0)

        if reply:

            ttl = reply.ttl

    except Exception:

        return "OS desconocido"

    if ttl is not None:

        if ttl <= 64:

            return "Linux/Unix"

        elif ttl <= 128:

            return "Windows"

        elif ttl <= 255:

            return "Dispositivo de red o BSD"

    return "OS desconocido"

# Escanear puertos con hilos

def scan\_host(ip):

    host\_data = {"ip": ip, "puertos": {}, "os": "OS desconocido"}

    threads = []

    def scan\_port(port):

        try:

            with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

                s.settimeout(0.3)

                result = s.connect\_ex((ip, port))

                host\_data["puertos"][port] = result == 0

        except Exception:

            host\_data["puertos"][port] = False

    for port in ports:

        t = threading.Thread(target=scan\_port, args=(port,))

        threads.append(t)

        t.start()

    for t in threads:

        t.join()

    host\_data["os"] = detect\_os(ip)

    return host\_data

# Escaneo de red con límite de hilos para velocidad moderada

def scan\_network(ip, mask, output):

    try:

        output.insert(tk.END, f"Escaneando red: {ip}/{mask}\n")

        output.see(tk.END)

        network = IPv4Network(f"{ip}/{mask}", strict=False)

        active\_hosts = []

        lock = threading.Lock()

        semaphore = threading.Semaphore(10)  # Máximo 10 hilos simultáneos

        def check\_host(host\_ip):

            with semaphore:

                pkt = IP(dst=host\_ip)/ICMP()

                reply = sr1(pkt, timeout=0.5, verbose=0)

                with lock:

                    if reply:

                        output.insert(tk.END, f"Host: {host\_ip} - ACTIVO\n")

                        active\_hosts.append(host\_ip)

                    else:

                        output.insert(tk.END, f"Host: {host\_ip} - NO ACTIVO\n")

                    output.see(tk.END)

                    output.update\_idletasks()

        threads = []

        for host in network.hosts():

            t = threading.Thread(target=check\_host, args=(str(host),))

            threads.append(t)

            t.start()

        for t in threads:

            t.join()

        output.insert(tk.END, "\n--- DETALLES DE HOSTS ACTIVOS ---\n")

        output.see(tk.END)

        for host in active\_hosts:

            datos = scan\_host(host)

            output.insert(tk.END, f"\nHost: {host} - ACTIVO\n")

            for port, disponible in datos["puertos"].items():

                estado = "Puerto activo" if disponible else "Puerto no disponible"

                output.insert(tk.END, f"{ports[port]} ({port}) - {estado}\n")

            output.insert(tk.END, f"OS - {datos['os']}\n")

            output.see(tk.END)

            output.update\_idletasks()

        output.insert(tk.END, "\nEscaneo finalizado.\n")

        output.see(tk.END)

    except Exception as e:

        output.insert(tk.END, f"Error en el escaneo: {e}\n")

        output.see(tk.END)

# Validar entrada y permisos de red

def start\_scan(ip\_entry, mask\_entry, output):

    ip = ip\_entry.get()

    mask = mask\_entry.get()

    try:

        user\_network = ip\_network(f"{ip}/{mask}", strict=False)

        local\_network = get\_local\_network()

        if local\_network is None:

            output.insert(tk.END, "⚠️ Error: No se pudo determinar tu red local.\n")

            output.see(tk.END)

            return

        if str(user\_network.network\_address) != str(local\_network.network\_address):

            output.insert(tk.END, "⚠️ Error: La IP ingresada es incorrecta o no está permitida.\n")

            output.see(tk.END)

            return

        if str(user\_network.netmask) != str(local\_network.netmask):

            output.insert(tk.END, "⚠️ Error: La máscara de subred ingresada no coincide con la de tu red local.\n")

            output.see(tk.END)

            return

    except (ValueError, AddressValueError, NetmaskValueError):

        output.insert(tk.END, "⚠️ Error: Dirección IP o máscara inválida.\n")

        output.see(tk.END)

        return

    output.delete("1.0", tk.END)

    threading.Thread(target=scan\_network, args=(ip, mask, output)).start()

# Interfaz gráfica

def main\_gui():

    window = tk.Tk()

    window.title("Escáner de Red")

    window.geometry("750x520")

    frame = tk.Frame(window)

    frame.pack(pady=10)

    tk.Label(frame, text="Dirección IP:").grid(row=0, column=0, sticky="e")

    ip\_entry = tk.Entry(frame, width=20)

    ip\_entry.grid(row=0, column=1, padx=5)

    tk.Label(frame, text="Máscara de subred:").grid(row=1, column=0, sticky="e")

    mask\_entry = tk.Entry(frame, width=20)

    mask\_entry.grid(row=1, column=1, padx=5)

    scan\_button = tk.Button(frame, text="Scanner", command=lambda: start\_scan(ip\_entry, mask\_entry, output),

                            bg="#4CAF50", fg="white", padx=10, pady=5)

    scan\_button.grid(row=0, column=2, rowspan=2, padx=10, pady=5)

    output = scrolledtext.ScrolledText(window, wrap=tk.WORD, width=90, height=25)

    output.pack(padx=10, pady=10)

    window.mainloop()

main\_gui()