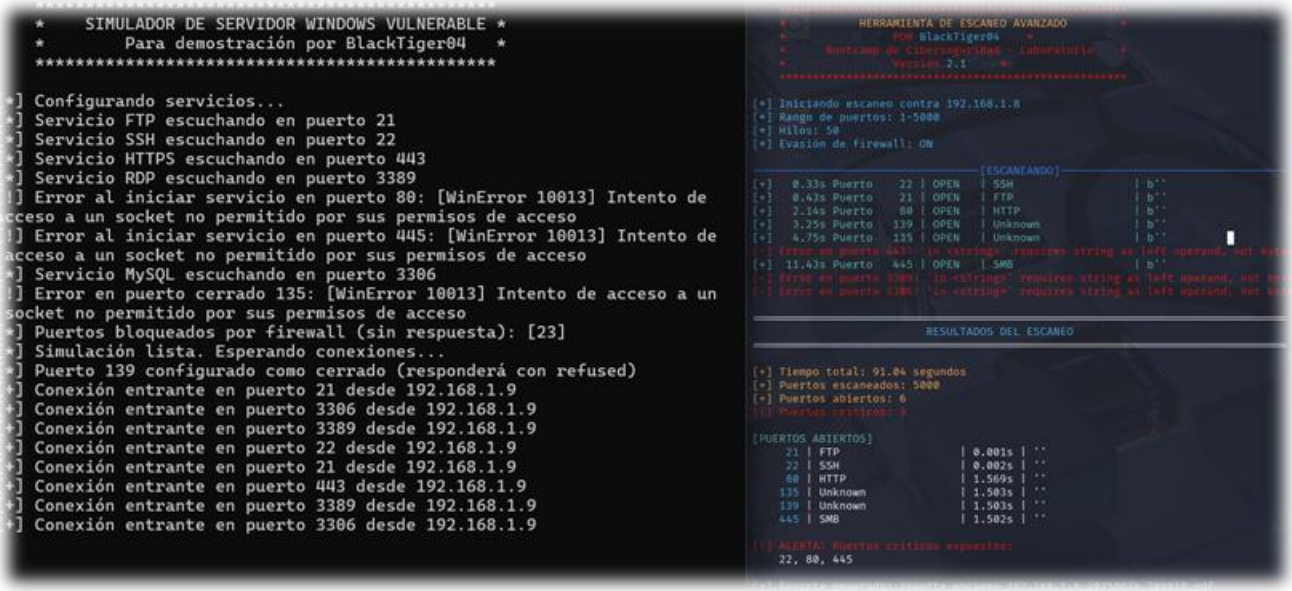
PROYECTO FINAL

Gestor de reportes de vulnerabilidades de red.



Presentado por:

Itamhar Daniel Barrios Castro

Talentotech Ciberseguridad Nivel Explorador (básico)

Popayán - Cauca

2025

Introducción

Este proyecto tiene como propósito simular un entorno realista de red para comprender cómo operan los servicios, cómo se realiza el escaneo de puertos, y cómo identificar posibles vulnerabilidades desde el punto de vista ofensivo y defensivo. Además de generar un reporte en pdf, el cual lista los puertos encontrados, que se simularon desde el server de Windows.

En Colombia, las actividades relacionadas con la seguridad informática, incluyendo la simulación y análisis de ataques a redes, están reguladas por diferentes leyes que buscan proteger la integridad, confidencialidad y disponibilidad de los sistemas informáticos, así como la privacidad de la información. El ejercicio integra conocimientos técnicos de redes, seguridad y programación, pero más importante aún, desarrolla el pensamiento crítico necesario para identificar riesgos, entender cómo operan los atacantes y anticiparse a sus movimientos con soluciones concretas.

El proyecto se divide en dos componentes principales:

Servidor simulado (Windows)

Ejecutado en una máquina virtual, este script emula varios servicios de red conocidos (como FTP, HTTP, SSH, SMB, etc.) en sus respectivos puertos. También simula puertos cerrados y bloqueados, y puede detectar comportamientos sospechosos como escaneos agresivos.

Herramienta de escaneo (Kali Linux – Máquina Virtual)

Desarrollada en Python, esta herramienta escanea un host dado, identifica qué puertos están abiertos, detecta banners para reconocer servicios, y genera un informe detallado en PDF. Usa técnicas como evasión de firewall mediante retrasos aleatorios y ejecución multihilo para mayor eficiencia. Sin embargo, es una versión limitada de su verdadero potencial de la aplicación de escaneo de red con Python.

Objetivo Propuesto

Generar reportes detallados que incluyan información relevante sobre los incidentes de seguridad detectados con una aplicación de scanner en Python de red (simulado), para su posterior análisis por parte de los administradores de red.

Instructivo.

1. Importaciones necesarias en Server Windows

```
import socket
import threading
import time
import random
from datetime import datetime
from collections import defaultdict
```

- **socket:** permite crear conexiones de red, como servidores y clientes.
- **threading:** permite ejecutar múltiples tareas al mismo tiempo (por ejemplo, atender varias conexiones).
- **time:** se usa para pausas (como sleep) o controlar el tiempo de ejecución.
- **random:** permite generar valores aleatorios, útil para simular tiempos de respuesta.
- **datetime:** aunque no se usa directamente en este fragmento, sirve para trabajar con fechas y horas.
- **defaultdict:** crea diccionarios que inicializan automáticamente valores por defecto (útil para contar conexiones por IP).

2. Definición de servicios simulados

```
# Configuración de servicios simulados
SERVICIOS = {
    21: {
        "name": "FTP",
        "banner": b"220 Microsoft FTP Service (Version 5.0)\r\n",
        "response": b"230 User logged in.\r\n"
    },
    22: {
        "name": "SSH",
        "banner": b"SSH-2.0-OpenSSH_8.2p1\r\n",
        "response": b""
    },
```

Este diccionario contiene los puertos comunes de servicios de red, como FTP (21), SSH (22), HTTP (80), etc. Cada entrada contiene:

- **name:** nombre del servicio.
- **banner:** el mensaje que envía el servidor al conectarse, como si fuera real.
- **response:** lo que devuelve si el cliente envía algo.

Los banners están codificados en bytes (b"") porque las conexiones de red trabajan a nivel binario.

3. Puertos cerrados y bloqueados

```
# Puertos cerrados (responden con refused)
CLOSED_PORTS = [135, 139]

# Puertos bloqueados por firewall (no responden)
BLOCKED_PORTS = [23]
```

- Puertos cerrados: simulan una respuesta "Connection Refused".
- Bloqueados por firewall: no responden, como si estuvieran filtrados.

Esto ayuda a simular un entorno más realista para pruebas de escaneo.

4. . Detección de escaneos

Este diccionario cuenta cuántas veces una IP se conecta al servidor.
Si una IP se conecta muchas veces en poco tiempo, se interpreta como un escaneo.

```
# Detección de escaneos
conexiones = defaultdict(int)

def handle_client(conn, addr, port):
    """Maneja la conexión entrante para un puerto específico"""
    global conexiones
    conexiones[addr[0]] += 1

    try:
        # Detección de escaneo (más de 10 conexiones desde la misma IP)
        if conexiones[addr[0]] > 10:
            print(f"[!] Detección de escaneo desde {addr[0]}. Bloqueando...")
            conn.close()
            return

        service = SERVICIOS.get(port)
        if service:
            # Enviar banner
            conn.sendall(service["banner"])

            # Simular interacción básica
            data = conn.recv(1024)
            if data:
                conn.sendall(service["response"])

            # Simular delay aleatorio
            time.sleep(random.uniform(0.1, 0.5))

    except Exception as e:
        print(f"[!] Error en puerto {port}: {str(e)}")
    finally:
        conn.close()
```

Este fragmento de código gestiona las conexiones entrantes a un puerto específico mediante la función "handle_client". Utiliza un diccionario (`conexiones`) para contar cuántas veces se conecta cada dirección IP. Si una IP realiza más de 10 conexiones, se considera un posible escaneo y se bloquea la comunicación cerrando la conexión. Si la IP no supera este límite, se busca el servicio asociado al puerto y, si existe, se le envía un banner (mensaje inicial). Luego se simula una interacción básica: se recibe un mensaje del cliente y se responde con un mensaje predefinido. Finalmente, se introduce una pequeña pausa con duración aleatoria para imitar el comportamiento real de un servicio. Al terminar, la conexión siempre se cierra, ya sea que haya ocurrido un error o no.

```python
def start_server(port):
    """Inicia un servidor en el puerto especificado"""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            s.bind(('0.0.0.0', port))
            s.listen(5)
            print(f"[*] Servicio {SERVICIOS[port]['name']} escuchando en puerto {port}")

            while True:
                conn, addr = s.accept()
                print(f"[+] Conexión entrante en puerto {port} desde {addr[0]}")
                threading.Thread(target=handle_client, args=(conn, addr, port)).start()

    except Exception as e:
        print(f"[!] Error al iniciar servicio en puerto {port}: {str(e)}")
```

Este fragmento de código define una función llamada `start_server` que inicia un servidor en un puerto específico. Usa la biblioteca `socket` para crear un socket TCP (IPv4) y permite reutilizar el puerto con `setsockopt`, luego lo vincula a todas las interfaces de red (`0.0.0.0`) y empieza a escuchar conexiones entrantes. Cuando una conexión llega, acepta al cliente y lanza un nuevo hilo para manejar esa conexión usando la función `handle_client`, permitiendo así manejar múltiples clientes simultáneamente. Si ocurre un error al iniciar el servidor, se muestra un mensaje con la descripción del problema.

```python
def simulate_closed_port(port):
    """Simula un puerto cerrado (respondiendo con connection refused)"""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            s.bind(('0.0.0.0', port))
            s.listen(5)
            print(f"[*] Puerto {port} configurado como cerrado (responderá con refused)")

            while True:
                conn, addr = s.accept()
                conn.close()  # Cierra inmediatamente para simular refused

    except Exception as e:
        print(f"[!] Error en puerto cerrado {port}: {str(e)}")
```

Esta función llamada `simulate_closed_port` simula un puerto cerrado en un servidor. Crea un socket TCP y lo configura para escuchar en el puerto indicado, pero en lugar de aceptar y procesar conexiones, las cierra inmediatamente cuando un cliente intenta conectarse. Esto genera el efecto de "connection refused" en el cliente, como si el puerto estuviera cerrado o no disponible. Es útil para pruebas o simulaciones de seguridad. Si ocurre algún error al intentar configurar el puerto, se imprime un mensaje con la descripción del fallo.

```
# Iniciar servicios abiertos
for port in SERVICIOS:
    threading.Thread(target=start_server, args=(port,), daemon=True).start()

# Iniciar puertos cerrados
for port in CLOSED_PORTS:
    threading.Thread(target=simulate_closed_port, args=(port,), daemon=True).start()

print("[*] Puertos bloqueados por firewall (sin respuesta):", BLOCKED_PORTS)
print("[*] Simulación lista. Esperando conexiones...")

try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("\n[*] Deteniendo servidor de simulación...")
```

Este fragmento de código inicia la simulación de servicios de red. Primero, crea un hilo por cada puerto en la lista `SERVICIOS` para simular servicios abiertos usando la función `start_server`. Luego, hace lo mismo para los puertos en `CLOSED_PORTS`, lanzando hilos que simulan puertos cerrados con la función `simulate_closed_port`. Los puertos en `BLOCKED_PORTS`, que representan puertos bloqueados por firewall (sin respuesta), solo se listan en pantalla. El programa imprime un mensaje indicando que la simulación está lista y luego entra en un bucle infinito en espera de conexiones. Si el usuario interrumpe con el teclado (Ctrl+C), el programa muestra un mensaje y se detiene limpiamente.

Importaciones necesarias en Escanner.py para Kali Linux

```
# -*- coding: utf-8 -*-
import socket
import time
import random
from concurrent.futures import ThreadPoolExecutor
import argparse
from datetime import datetime
import sys
from colorama import Fore, Style, init
from fpdf import FPDF
```

Este bloque importa las bibliotecas necesarias para el funcionamiento del programa. Se incluye `socket` para manejar conexiones de red, `time` y `datetime` para trabajar con fechas y tiempos, y `random` para generar valores aleatorios. `ThreadPoolExecutor` de `concurrent.futures` permite ejecutar tareas en paralelo mediante hilos. `argparse` sirve para gestionar argumentos desde la línea de comandos, y `sys` permite interactuar con el sistema (como cerrar el programa). La librería `colorama` se usa para imprimir texto en color en la consola, facilitando la lectura de mensajes, y `fpdf` permite generar archivos PDF, posiblemente para reportes o logs del programa.

```
# Inicializar colores para terminal
init(autoreset=True)

# Configuración global
VERSION = "2.1"
AUTHOR = "BlackTiger04"
MAX_THREADS = 50
DELAY_MIN = 0.05
DELAY_MAX = 0.5
```

Este fragmento configura opciones iniciales del programa. Primero, se llama a `init(autoreset=True)` de la librería `colorama` para que los colores en la terminal se restablezcan automáticamente después de cada impresión, mejorando la legibilidad. Luego, se definen variables globales: la versión del programa (`VERSION`), el nombre del autor (`AUTHOR`), el número máximo de hilos permitidos en paralelo (`MAX_THREADS`), y los valores mínimo y máximo de retardo entre acciones (`DELAY_MIN` y `DELAY_MAX`), lo que sugiere que el programa ejecuta tareas con tiempos aleatorios o controlados.

```
class PDFReport(FPDF):
    def header(self):
```

Este fragmento tiene dos partes principales: una clase para generar reportes en PDF y una función que muestra un banner colorido en consola. La clase `PDFReport` hereda de `FPDF` y define cómo se verá el reporte de simulación. Tiene tres métodos personalizados:

- `header()`: agrega un encabezado centrado con el título del reporte.
- `footer()`: añade un pie de página con el número de página actual.
- `chapter_title(title)` y `chapter_body(body)`: permiten incluir secciones en el PDF con título y contenido.

La función `mostrar_banner()` imprime en la terminal un banner decorativo en colores usando `colorama`, con información como el nombre del autor (`AUTHOR`), la versión del programa (`VERSION`), y un título llamativo. El uso de colores y formato mejora la presentación visual al iniciar la herramienta.

```
def parse_args():
    """Parseo de argumentos de línea de comandos"""
    parser = argparse.ArgumentParser(description="Herramienta de escaneo de puertos avanzado")
    parser.add_argument("target", help="IP del objetivo")
    parser.add_argument("-p", "--ports", help="Rango de puertos (ej: 20-100,80,443)", default="1-1024"
    parser.add_argument("-t", "--threads", type=int, help="Hilos de ejecución", default=MAX_THREADS)
    parser.add_argument("-v", "--verbose", help="Modo verboso", action="store_true")
    parser.add_argument("--no-evasion", help="Desactivar evasión de firewall", action="store_true")
    return parser.parse_args()
```

Este fragmento define la función `parse_args`, que se encarga de leer y procesar los argumentos ingresados por el usuario desde la línea de comandos. Utiliza la librería `argparse` para permitir especificar la IP del objetivo, un rango de puertos a escanear, la cantidad de hilos a usar, si se desea activar el modo detallado (`verbose`), y si se desactiva la evasión de firewall. Estos parámetros permiten personalizar el comportamiento de la herramienta de escaneo de puertos.

```python
def generar_rango_puertos(rango_str):
    """Convierte un string de rango a lista de puertos"""
    puertos = []
    for parte in rango_str.split(','):
        if '-' in parte:
            inicio, fin = map(int, parte.split('-'))
            puertos.extend(range(inicio, fin+1))
        else:
            puertos.append(int(parte))
    return sorted(set(puertos))  # Eliminar duplicados
```

Este fragmento define la función “generar_rango_puertos”, que toma como entrada una cadena con rangos de puertos (por ejemplo, `"20-25,80,443"`) y la convierte en una lista de números de puerto individuales. Si la cadena contiene un rango (como `"20-25"`), se genera una secuencia con todos los puertos en ese rango; si es un número suelto (como `"80"`), se agrega directamente. Al final, la función elimina duplicados y ordena los puertos antes de devolver la lista resultante. Esto permite al programa manejar rangos de puertos flexibles y personalizados para el escaneo.

```python
def detectar_servicio(banner, puerto):
    """Detección mejorada de servicios"""
    if not banner:
        # Detección por puerto conocido
        servicios = {
            21: "FTP",
            22: "SSH",
            80: "HTTP",
            443: "HTTPS",
            3389: "RDP",
            445: "SMB",
            3306: "MySQL"
        }
        return servicios.get(puerto, "Unknown")

    banner_str = str(banner).lower()

    if b"ftp" in banner_str.lower():
        return "FTP"
    elif b"http" in banner_str:
        return "HTTP"
    elif b"microsoft-iis" in banner_str.lower():
        return "IIS"
    elif b"smb" in banner_str.lower() or b"\xffSMB" in banner:
        return "SMB"
    elif b"rdp" in banner_str.lower() or b"\x03\x00\x00" in banner:
        return "RDP"
    elif b"mysql" in banner_str.lower():
        return "MySQL"
    else:
        return "Unknown Service"
```

Este fragmento define la función “detectar_servicio”, que tiene como objetivo identificar el tipo de servicio que se está ejecutando en un puerto, ya sea analizando el banner recibido o basándose en el número de puerto. Si no hay banner disponible, la detección se realiza por comparación con una lista de puertos comúnmente asociados a ciertos servicios (como el 80 para HTTP o el 22 para SSH). Si hay un banner, se convierte a texto en minúsculas y se analiza buscando palabras clave o patrones específicos (como "ftp", "http", "smb", etc.) que permitan identificar servicios como FTP, HTTP, SMB, RDP o MySQL. Si no se detecta ningún patrón conocido, se devuelve "Unknown Service", indicando que el tipo de servicio no se pudo determinar.

```python
def escanear_puerto(ip, puerto, args, inicio_escaneo):
    """Escanea un puerto individual con técnicas avanzadas"""
    try:
        # Técnica de evasión
        if not args.no_evasion:
            time.sleep(random.uniform(DELAY_MIN, DELAY_MAX))

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(1.5)

            inicio = time.time()
            resultado = s.connect_ex((ip, puerto))

            if resultado == 0:  # Puerto abierto
                # Banner grabbing avanzado
                try:
                    banner = s.recv(1024) if puerto not in [21, 22] else b''
                except:
                    banner = b''

                tiempo_respuesta = time.time() - inicio
                servicio = detectar_servicio(banner, puerto)
                estado = "OPEN"

                if args.verbose:
                    tiempo_transcurrido = time.time() - inicio_escaneo
                    print(f"{Fore.GREEN}[+] {tiempo_transcurrido:6.2f}s Puerto {puerto:5d} | {estado:6} | {servicio:20} | {banner[:30]!r}{Style.RESET_ALL}")

                return (puerto, True, tiempo_respuesta, servicio, banner.decode(errors='ignore')[:100])
            else:
                return (puerto, False, 0, None, None)

    except (socket.timeout, ConnectionRefusedError):
        return (puerto, False, 0, None, None)
    except Exception as e:
        if args.verbose:
            print(f"{Fore.RED}[-] Error en puerto {puerto}: {str(e)}{Style.RESET_ALL}")
        return (puerto, False, 0, None, None)
```

Este fragmento define la función “escanear_puerto”, que se encarga de escanear un único puerto de una dirección IP utilizando técnicas avanzadas. Primero, si no se ha desactivado la evasión de firewall (“args.no_evasion”), introduce un retardo aleatorio para simular un comportamiento menos detectable. Luego, intenta establecer una conexión TCP al puerto objetivo con un tiempo límite de 1.5 segundos. Si la conexión es exitosa (puerto abierto), intenta capturar un banner del servicio —salvo para ciertos puertos como 21 y 22— y calcula el tiempo de respuesta. Con el banner recibido, llama a “detectar_servicio” para identificar el tipo de servicio. Si el modo detallado (`verbose`) está activado, imprime la información del puerto, estado, servicio y parte del banner capturado. Si la conexión falla o se produce un error, devuelve que el puerto está cerrado o inaccesible. De esta forma, la función recopila datos útiles sobre cada puerto escaneado.

```python
def mostrar_resultados(resultados, tiempo_total):
    """Muestra los resultados del escaneo"""
    print("\n" + "="*80)
    print(f"{Fore.CYAN}RESULTADOS DEL ESCANEO".center(80))
    print("="*80 + Style.RESET_ALL)

    abiertos = [r for r in resultados if r[1]]
    criticos = [p for p, _, _, s, _ in abiertos if s in ["FTP", "HTTP", "HTTPS", "RDP", "SMB", "MySQL"]]

    print(f"\n{Fore.YELLOW}[+] Tiempo total: {tiempo_total:.2f} segundos{Style.RESET_ALL}")
    print(f"{Fore.YELLOW}[+] Puertos escaneados: {len(resultados)}{Style.RESET_ALL}")
    print(f"{Fore.YELLOW}[+] Puertos abiertos: {len(abiertos)}{Style.RESET_ALL}")
    print(f"{Fore.RED}[!] Puertos críticos: {len(criticos)}{Style.RESET_ALL}")

    if abiertos:
        print(f"\n{Fore.GREEN}[PUERTOS ABIERTOS]{Style.RESET_ALL}")
        for puerto, _, tiempo, servicio, banner in sorted(abiertos, key=lambda x: x[0]):
            print(f"  {Fore.CYAN}{puerto:5d}{Style.RESET_ALL} | {servicio:20} | {tiempo:.3f}s | {banner[:50]!r}")
```

Este fragmento define la función “mostrar_resultados”, encargada de presentar en consola los resultados del escaneo de puertos. Imprime un encabezado decorativo y resume información clave como el tiempo total del escaneo, la cantidad de puertos analizados, cuántos están abiertos y cuántos son considerados críticos (como FTP, HTTP, HTTPS, RDP, SMB o MySQL). Luego, si hay puertos abiertos, muestra una lista detallada con el número de puerto, el servicio detectado, el tiempo de respuesta y parte del banner recibido. El uso de colores con `colorama` mejora la legibilidad y destaca la información más relevante para el usuario.

```python
def generar_reporte(resultados, tiempo_total, target):
    """Genera un reporte PDF con los resultados"""
    pdf = PDFReport()
    pdf.add_page()
    pdf.set_auto_page_break(auto=True, margin=15)

    # Información general
    pdf.chapter_title('Información del Escaneo')
    pdf.chapter_body(f"""
    Fecha: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}
    Objetivo: {target}
    Tiempo total de escaneo: {tiempo_total:.2f} segundos
    Puertos escaneados: {len(resultados)}
    """)

    # Puertos abiertos
    abiertos = [r for r in resultados if r[1]]
    pdf.chapter_title('Puertos Abiertos Detectados')
```

Este fragmento define la función “generar_reporte”, que crea un archivo PDF con los resultados del escaneo utilizando la clase personalizada `PDFReport`. Primero, se añade una nueva página y se habilita el salto de página automático. Luego, se agrega una sección titulada "Información del Escaneo", que incluye la fecha actual, la dirección IP del objetivo, el tiempo total del análisis y la cantidad de puertos escaneados. A continuación, se filtran los resultados para identificar los puertos abiertos, y se prepara una nueva sección

titulada "Puertos Abiertos Detectados" para documentarlos en el reporte. Esta función permite generar una salida clara, profesional y fácilmente archivable en formato PDF para su posterior análisis.

```
if abiertos:
    # Encabezado de tabla
    pdf.set_font('Arial', 'B', 10)
    pdf.cell(20, 10, 'Puerto', 1, 0, 'C')
    pdf.cell(40, 10, 'Servicio', 1, 0, 'C')
    pdf.cell(20, 10, 'Tiempo', 1, 0, 'C')
    pdf.cell(110, 10, 'Banner', 1, 1, 'C')

    # Contenido de tabla
    pdf.set_font('Arial', '', 10)
    for puerto, _, tiempo, servicio, banner in sorted(abiertos, key=lambda x: x[0]):
        pdf.cell(20, 10, str(puerto), 1, 0, 'C')
        pdf.cell(40, 10, servicio, 1, 0, 'C')
        pdf.cell(20, 10, f"{tiempo:.3f}s", 1, 0, 'C')
        pdf.cell(110, 10, banner[:50] if banner else "N/A", 1, 1, 'L')
else:
    pdf.chapter_body("No se encontraron puertos abiertos.")
```

Este fragmento complementa la función de generación del reporte en PDF, añadiendo una tabla con los detalles de los puertos abiertos encontrados durante el escaneo. Si existen puertos abiertos, primero se crea un encabezado de tabla con las columnas: "Puerto", "Servicio", "Tiempo" y "Banner", usando una fuente en negrita. Luego, se recorre la lista de puertos abiertos y se agregan filas a la tabla con la información correspondiente a cada uno, incluyendo el número de puerto, el nombre del servicio detectado, el tiempo de respuesta y el banner recibido (hasta 50 caracteres). Si no se encontraron puertos abiertos, se añade un mensaje indicando esto en lugar de la tabla. Este bloque mejora la presentación del reporte, proporcionando un resumen estructurado y legible.

```
# Guardar PDF
filename = f"reporte_escaneo_{target}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.pdf"
pdf.output(filename)
print(f"\n{Fore.GREEN}[+] Reporte generado: {filename}{Style.RESET_ALL}")
```

Este fragmento finaliza la generación del reporte PDF guardando el archivo con un nombre que incluye la IP del objetivo (`target`) y la fecha y hora actuales en formato `YYYYMMDD_HHMMSS`, lo que garantiza que cada archivo tenga un nombre único. Luego, se utiliza el método `output` de la clase `FPDF` para guardar el reporte en disco. Finalmente, se imprime en la consola un mensaje en color verde que confirma que el reporte ha sido generado correctamente y muestra el nombre del archivo creado. Esta parte asegura la persistencia del informe y notifica visualmente al usuario sobre su éxito.

```python
def main():
    """Función principal"""
    mostrar_banner()
    args = parse_args()

    try:
        puertos = generar_rango_puertos(args.ports)
        inicio_escaneo = time.time()

        print(f"{Fore.CYAN}[*] Iniciando escaneo contra {args.target}{Style.RESET_ALL}")
        print(f"{Fore.CYAN}[*] Rango de puertos: {args.ports}{Style.RESET_ALL}")
        print(f"{Fore.CYAN}[*] Hilos: {args.threads}{Style.RESET_ALL}")
        print(f"{Fore.CYAN}[*] Evasión de firewall: {'ON' if not args.no_evasion else 'OFF'}{Style.RESET_ALL}")
        print(f"\n{Fore.BLUE}{'[ESCANEANDO]'.center(80, '-')}{Style.RESET_ALL}")

        with ThreadPoolExecutor(max_workers=args.threads) as executor:
            resultados = list(executor.map(
                lambda p: escanear_puerto(args.target, p, args, inicio_escaneo),
                puertos
            ))

        tiempo_total = time.time() - inicio_escaneo
        mostrar_resultados(resultados, tiempo_total)
        generar_reporte(resultados, tiempo_total, args.target)

    except KeyboardInterrupt:
        print(f"\n{Fore.YELLOW}[!] Escaneo interrumpido por el usuario{Style.RESET_ALL}")
        sys.exit(1)
    except Exception as e:
        print(f"\n{Fore.RED}[!] Error crítico: {str(e)}{Style.RESET_ALL}")
        sys.exit(1)

if __name__ == "__main__":
    main()
```

Por último, este fragmento contiene la función `main`, que actúa como punto de entrada principal del programa. Primero, muestra un banner de presentación y analiza los argumentos ingresados por el usuario. Luego, genera la lista de puertos a escanear y marca el tiempo de inicio del escaneo. Imprime información general como el objetivo, el rango de puertos, el número de hilos y si la evasión de firewall está activada. A continuación, lanza el escaneo en paralelo utilizando `ThreadPoolExecutor`, ejecutando la función `escanear_puerto` para cada puerto de forma concurrente. Una vez finalizado el escaneo, calcula el tiempo total transcurrido, muestra los resultados por consola y genera un reporte en PDF. Además, el bloque incluye manejo de errores: si el usuario interrumpe el proceso (Ctrl+C) o ocurre una excepción crítica, se imprime un mensaje de advertencia y se detiene el programa de forma segura. Finalmente, la condición `if __name__ == "__main__":` asegura que la función `main` se ejecute solo cuando el script se ejecuta directamente.

Funcionamiento

Para que este proyecto funcione correctamente en cualquier entorno, es necesario cumplir con ciertos requisitos técnicos, instalaciones previas, y tener en cuenta aspectos estructurales del código, es necesario contar con conocimientos básicos previos de configuración de máquinas virtuales y comandos por consola para Kali Linux A continuación, se detallan todos los elementos necesarios:

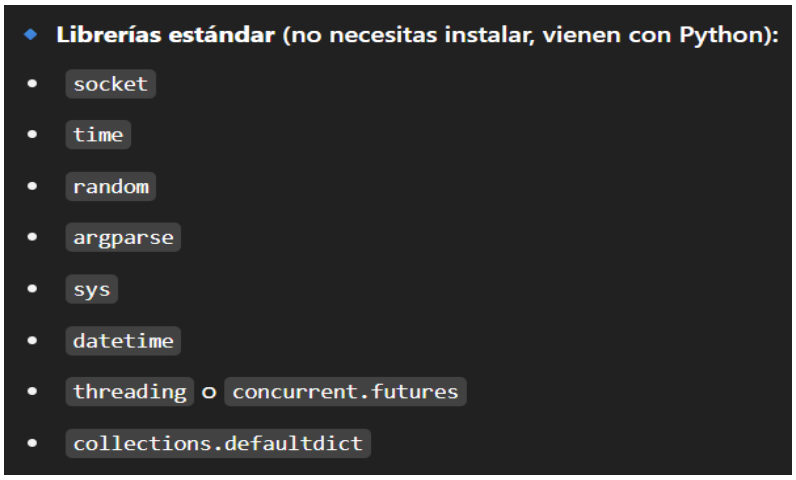
1. **Requisitos del sistema**

Maquina virtual de preferencia con sistema operativo Linux. Para la ejecución de pruebas.
Sistema operativo: Windows, Linux o macOS (preferiblemente Kali Linux para pruebas de red avanzadas).
Python: Versión 3.6 o superior instalada.
Permisos de red: Algunos puertos (por debajo del 1024) pueden requerir permisos de administrador/root para abrir o escanear.
Acceso a consola o terminal: Para ejecutar el script con argumentos.
Dos maquinas bajo la misma red, atacante y objetivo.
Recursos necesarios de cómputo para la configuración de la(s) maquina(s) virtual(es).

2. **Estructura del proyecto**

```
proyecto_escaneo/
│
├── escaner.py              # Script principal (donde está la función main)
├── simulador.py            # (opcional) Para separar la lógica del servidor simulado
├── requirements.txt        # Lista de dependencias
└── README.md               # Documentación del proyecto
```

En tu editor de código como Visual Studio Code, se hace necesario tener instaladas las librerías necesarias.

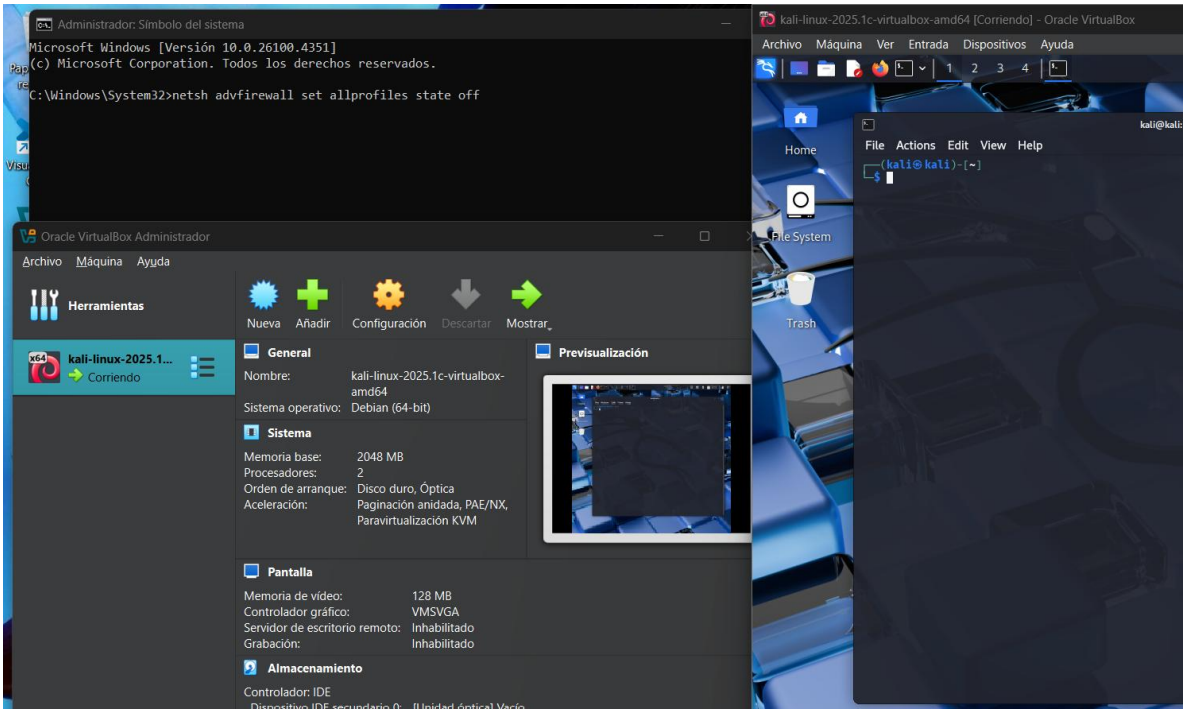


Librerías externas que debes instalar en activar un entorno virtual, en la consola de Kali Linux al iniciar un entorno aislado de Python, llamado entorno virtual, puedes instalar paquetes y dependencias sin afectar el sistema global. Se hace necesario para estas librerías:

- **colorama:** para imprimir texto en color en la consola (compatible con Windows y Unix).
- **fpdf:** para generar el reporte PDF de los resultados del escaneo.

Nota: Recomendable que se instalen en la misma carpeta donde esta el script de Python para evitar errores.

Preparando escenario



Es necesario que se desactive temporalmente el firewall de la maquina objetivo para optimizar los tiempos de respuesta y mejorar la experiencia de la simulación, ya que se bloquean ciertas características de la conexión atreves del escanner.

Comando por consola para Windows (Objetivo): netsh advfirewall set allprofiles state off

Para comprobar la comunicación puedes hacer un Ping en ambas maquinas y verificar la conexión, para hacerlo es necesario que conozcas la dirección IP de tus maquinas.
Para esto debe usar el comando **ifconfig** para Linux y **ipconfig** para Windows. En mi caso:

En Kali Linux

En Windows

Ping de confirmación en Windows.

```
Haciendo ping a 192.168.98.56 con 32 bytes de datos:
Respuesta desde 192.168.98.56: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.98.56: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.98.56: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.98.56: bytes=32 tiempo<1m TTL=64
```

Ping de confirmación en Kali Linux

```
└─$ ping 192.168.98.41
PING 192.168.98.41 (192.168.98.41) 56(84) bytes of data.
64 bytes from 192.168.98.41: icmp_seq=1 ttl=128 time=0.521 ms
64 bytes from 192.168.98.41: icmp_seq=2 ttl=128 time=0.703 ms
64 bytes from 192.168.98.41: icmp_seq=3 ttl=128 time=0.471 ms
64 bytes from 192.168.98.41: icmp_seq=4 ttl=128 time=0.449 ms
```

Al ingresar en tu maquina objetivo se coloca a correr el simulador vulnerable a ataque por red. De la siguiente manera.

1. Entra por consola a la carpeta donde se ubica
2. Digita en la terminal python servidor_simulado.py

```
C:\Users\ASUS>cd C:\Users\ASUS\Documents\PROYECTO

C:\Users\ASUS\Documents\PROYECTO>python servidor_simulado.py

    ******************************************
    *    SIMULADOR DE SERVIDOR WINDOWS VULNERABLE *
    *        Para demostración por BlackTiger04   *
    ******************************************

[*] Configurando servicios...
[*] Servicio FTP escuchando en puerto 21
[*] Servicio SSH escuchando en puerto 22
[!] Error al iniciar servicio en puerto 80: [WinError 10013] Intento de acceso a un socket no permitido por sus permisos
 de acceso
[*] Servicio HTTPS escuchando en puerto 443
[*] Servicio RDP escuchando en puerto 3389
[*] Servicio MySQL escuchando en puerto 3306
[!] Error al iniciar servicio en puerto 445: [WinError 10013] Intento de acceso a un socket no permitido por sus permiso
s de acceso
[*] Puertos bloqueados por firewall (sin respuesta): [23]
[*] Simulación lista. Esperando conexiones...
[!] Error en puerto cerrado 135: [WinError 10013] Intento de acceso a un socket no permitido por sus permisos de acceso
[*] Puerto 139 configurado como cerrado (responderá con refused)
```

Si todo es correcto debes mirar un panel como este por consola que muestra el simulador corriendo de manera satisfactoria.

Nota: es necesario tener Python instalado en las máquinas para realizar el ejercicio.

Maquina atacante

```
┌──(kali㉿kali)-[~/Documents/SIMULADO/RESULTADO]
└─$ source venv/bin/activate

┌──(venv)─(kali㉿kali)-[~/Documents/SIMULADO/RESULTADO]
└─$ python3 escanner.py 192.168.98.56 -p 1-5000 -t 50 -v
```

En primer lugar, la línea "**cd /home/kali/Documents/SIMULADO/RESULTADO**" indica al sistema que se debe cambiar el directorio de trabajo actual. En este caso, se accede a la ruta específica donde se encuentran los archivos del proyecto o simulación de escaneo de red. Esto es necesario para que las siguientes instrucciones se ejecuten en el contexto correcto, donde están alojados tanto el entorno virtual como el script Python correspondiente.

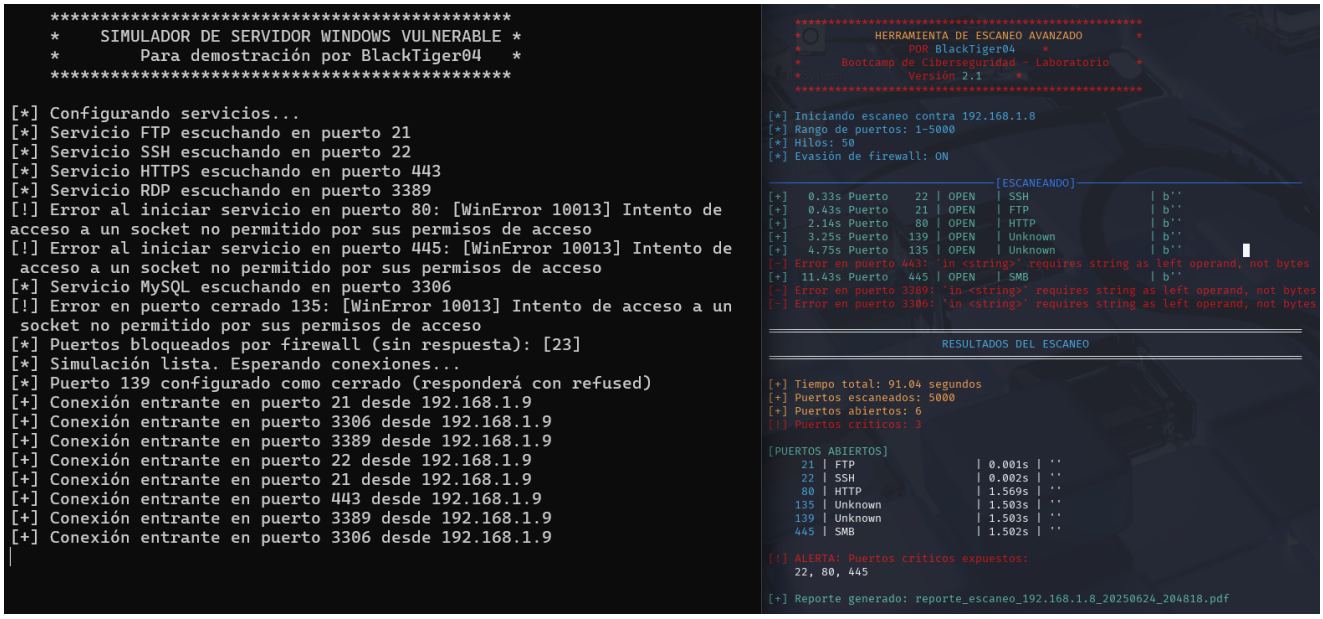
A continuación, con el comando "**source venv/bin/actívate**", se procede a activar un entorno virtual de Python previamente creado en la carpeta `venv`. Esta acción permite trabajar dentro de un espacio aislado del sistema principal, donde están instaladas únicamente las dependencias necesarias para el proyecto. Activar este entorno garantiza que el script se ejecute con las librerías y configuraciones exactas que requiere, sin interferencias externas o conflictos con otras versiones de paquetes.

Por último, se ejecuta el comando "**python3 escanner.py 192.168.98.56 -p 1-5000 -t 50 -v**". Esta instrucción lanza un script llamado "**escanner.py**", que ha sido programado para realizar un escaneo de puertos en un equipo con la dirección IP "192.168.98.56". El parámetro "**-p 1-5000**" indica que se deben escanear los puertos desde el 1 hasta el 5000, mientras que "**-t 50**" especifica que se deben utilizar 50 hilos de ejecución en paralelo, lo que acelera el proceso al distribuir la carga del escaneo. Finalmente, el parámetro "**-v**" activa el modo detallado o "**verbose",** el cual proporciona información adicional en tiempo real durante la ejecución del escaneo.

```
┌──(venv)─(kali㉿kali)-[~/Documents/SIMULADO/RESULTADO]
└─$ python3 escanner.py 192.168.98.56 -p 1-5000 -t 50 -v

    ******************************************************
    *           HERRAMIENTA DE ESCANEO AVANZADO          *
    *                POR BlackTiger04     *
    *      Bootcamp de Ciberseguridad - Laboratorio      *
    *                Versión 2.1      *
    ******************************************************

[*] Iniciando escaneo contra 192.168.98.56
[*] Rango de puertos: 1-5000
[*] Hilos: 50
[*] Evasión de firewall: ON

------------------------------[ESCANEANDO]------------------------------
[!] Alerta IDS/IPS: Paquete bloqueado en puerto 22 (Simulación)
```

Ejecución de la aplicación, una vez se ponga a correr la aplicación, lo primero que muestra es la presentación de la herramienta, un cuadro con un titulo para la aplicación, el seudónimo del autor, y el enfoque por le que se realiza el proyecto, la versión 2.1 es para tipo educativo, la versión 1 es más para entornos reales y en la que se basa el actual proyecto.

Paquetes conseguidos



Del lado derecho se observa las conexiones atrapadas por la aplicación puertos abiertos y una pequeña vista previa de los resultados obtenidos, en el lado izquierdo confirma la conexión entrante desde cada uno de los puertos previamente configurados para su escucha.

Al final de la línea del lado derecho, se puede observar la generación del archivo en pdf, el cual es el resultado o culminación del proyecto, dando cumplimiento al objetivo presentado al inicio del proyecto el cual buscaba reportes detallados que incluyan información relevante sobre los incidentes de seguridad detectados con una aplicación de scanner en Python de red (simulado), para su posterior análisis por parte de los administradores de red.

Conclusión

En conclusión, el conjunto de códigos analizados representa una herramienta avanzada de simulación y detección de escaneo de puertos, diseñada tanto para fines educativos como para prácticas de ciberseguridad defensiva. Su estructura se divide en dos grandes componentes: uno que simula servicios reales y falsos en distintos puertos, y otro que actúa como escáner, detectando servicios activos, registrando información y generando reportes.

Nota: Se anexa el resultado del PDF. Como es generada por la aplicación.

Reporte de Simulación de Escaneo

Información del Escaneo

Fecha: 2025-06-24 20:48:18

Objetivo: 192.168.1.8

Tiempo total de escaneo: 91.04 segundos

Puertos escaneados: 5000

Puertos Abiertos Detectados

Puerto	Servicio	Tiempo	Banner
21	FTP	0.001s	N/A
22	SSH	0.002s	N/A
80	HTTP	1.569s	N/A
135	Unknown	1.503s	N/A
139	Unknown	1.503s	N/A
445	SMB	1.502s	N/A

Análisis de Seguridad

Puertos críticos expuestos: 3

Servicios vulnerables detectados: SMB, FTP, HTTP, SSH, Unknown

Recomendaciones:

- Cerrar puertos no esenciales

- Actualizar servicios expuestos

- Implementar reglas de firewall más estrictas

- Monitorear tráfico en puertos críticos

Nota

Este trabajo fue realizado por el campista BlackTiger04, para el Bootcamp de Ciberseguridad 2025.