

## Memoria de proyecto

# ESCÁNER DE PUERTOS Y DETECCIÓN DE VULNERABILIDADES EN PYTHON

**Pablo Manuel Jiménez Torres**

Graduado en Cybersecurity en la universidad St John de York

<i>Versión:</i>	1.0.1
<i>Fecha:</i>	29 - Septiembre - 2025
<i>Contacto:</i>	jimeneztorrespablomanuel@gmail.com

# 1. Introducción

La **seguridad informática** es un componente crítico en la protección de los sistemas tecnológicos modernos, tanto en entornos corporativos como académicos. La exposición de servicios y puertos abiertos en una red representa uno de los vectores de ataque más comunes, lo que hace que el análisis y la identificación de vulnerabilidades sean pasos esenciales en cualquier estrategia de ciberseguridad. Detectar y gestionar de manera proactiva estas vulnerabilidades permite reducir riesgos, prevenir accesos no autorizados y proteger la integridad, confidencialidad y disponibilidad de la información.

El presente proyecto tiene como objetivo el **desarrollo de una herramienta de análisis de seguridad automatizada en Python**, diseñada específicamente para sistemas Windows, que facilite la identificación de puertos abiertos y la correlación de servicios con vulnerabilidades conocidas. Esta herramienta integra varias funcionalidades clave:

1. **Escaneo de puertos TCP/UDP:**

La herramienta permite explorar un rango de puertos definido por el usuario en un host objetivo, identificando qué puertos se encuentran abiertos y disponibles para comunicación, utilizando técnicas de escaneo basadas en sockets.

2. **Detección de servicios y versiones mediante banner grabbing:**

A través del análisis de las respuestas de los servicios, la herramienta intenta determinar el tipo de servicio activo en cada puerto, así como su versión, lo que permite identificar posibles vulnerabilidades asociadas.

3. **Correlación con vulnerabilidades conocidas:**

La herramienta se conecta a la **National Vulnerability Database (NVD)** para consultar vulnerabilidades públicas (CVEs) asociadas a los servicios detectados. Además, incorpora la severidad de cada vulnerabilidad utilizando el **sistema CVSS**, facilitando la priorización de riesgos críticos frente a riesgos menores.

4. **Exportación de resultados en formatos JSON y CSV:**

Se generan reportes estructurados y legibles, que permiten un análisis posterior, integración con otras herramientas o almacenamiento histórico de auditorías de seguridad.

5. **Generación de un ejecutable portable (.exe) mediante PyInstaller:**

Esto permite ejecutar la herramienta en sistemas Windows sin necesidad de instalar Python ni dependencias adicionales, simplificando su distribución y uso en entornos donde no se dispone de un entorno de desarrollo Python completo.

La herramienta está orientada a **estudiantes, investigadores y profesionales de seguridad informática**, proporcionando un escáner ligero, eficiente y flexible, adecuado tanto para análisis rápidos de red como para prácticas educativas en ciberseguridad. Además, la implementación de **programación asíncrona con `asyncio`** permite que la herramienta realice múltiples escaneos de manera simultánea, aumentando significativamente la velocidad de ejecución sin comprometer la precisión de los resultados.

Como valor agregado, la herramienta incluye un **sistema de cronometraje** que permite registrar el tiempo total de escaneo, proporcionando métricas útiles para evaluar el rendimiento de la herramienta y optimizar futuras auditorías de seguridad.

En resumen, este proyecto combina **automatización, eficiencia y análisis avanzado**, proporcionando una solución integral para la identificación de servicios expuestos y la gestión de vulnerabilidades conocidas en entornos Windows, contribuyendo así a la mejora de la postura de seguridad de cualquier infraestructura tecnológica.

## 2. Objetivos

### 2.1 Objetivo general

Desarrollar y entregar una herramienta automatizada en Python para Windows capaz de realizar un escaneo eficiente de puertos TCP/UDP, identificar servicios mediante *banner grabbing*, correlacionar las detecciones con vulnerabilidades publicadas en la NVD (incluyendo puntuaciones CVSS) y generar reportes en formatos estándar (JSON y CSV). La herramienta deberá ser distributable como un ejecutable (.exe) y contar con documentación de uso profesional.

### 2.2 Objetivos específicos (SMART)

1. **Automatizar el análisis de puertos abiertos en un host objetivo.**
  - a. *Específico: Implementar un motor de escaneo que cubra puertos TCP y una opción básica para UDP.*
  - b. *Medible: Debe poder escanear rangos de hasta 1000 puertos por host en menos de 120 segundos en condiciones de red normales (medición en una red LAN estándar).*
  - c. *Alcanzable: Uso de **asyncio** para paralelizar conexiones.*
  - d. *Relevante: Provee el reconocimiento inicial necesario en cualquier auditoría.*
  - e. *Temporal: Funcionalidad entregada en la fase 1 de desarrollo (primer hito).*
2. **Obtener información de servicios activos mediante captura de banners (banner grabbing).**
  - a. *Implementar técnicas de banner grabbing TCP (envío de payloads genéricos y lectura de respuesta).*
  - b. *Proporcionar al menos la identificación de servicio y versión cuando esté disponible; si no, registrar la cabecera recibida o la respuesta parcial.*

3. **Consultar la NVD para identificar vulnerabilidades asociadas a las versiones detectadas.**
  - a. *Integrar consultas a la API pública de la NVD (keywordSearch) y mapear respuestas a estructuras internas.*
  - b. *Limitar el número de resultados por búsqueda (configurable) y respetar políticas de rate limiting (pausa configurable entre llamadas).*
4. **Calcular y mostrar la severidad CVSS de las vulnerabilidades encontradas.**
  - a. *Extraer base score y categoría (Low/Medium/High/Critical) de las métricas CVSS (v3.1 prioritaria y fallback a v2 si procede).*
  - b. *Mostrar y exportar la puntuación CVSS junto al CVE en los reportes.*
5. **Optimizar la eficiencia del escaneo mediante programación asíncrona con `asyncio`.**
  - a. *Arquitectura basada en tareas asincrónicas con semáforos para controlar la concurrencia.*
  - b. *Parchear la configuración para ajustar `CONCURRENCY` según recursos de la máquina objetivo.*
6. **Medir el tiempo de ejecución del análisis.**
  - a. *Incorporar un cronómetro de alta resolución (`time.perf_counter()`) que reporte tiempo total y, opcionalmente, tiempos parciales por etapa (escaneo, banner grabbing, consulta NVD).*
7. **Distribuir la herramienta en formato ejecutable (.exe) con permisos de administrador preconfigurados.**
  - a. *Empaquetar con PyInstaller y proporcionar una opción `--uac-admin` en el proceso de empaquetado para que el ejecutable solicite elevación UAC al ejecutarse (documentar implicaciones).*
  - b. *Alternativa: permitir no incluir UAC para ejecuciones en entornos donde no se deseen elevaciones automáticas.*
8. **Proporcionar un manual de uso claro y accesible y la documentación técnica.**
  - a. *Entregar `manual1.txt` para usuarios finales y un documento técnico (memoria) con: arquitectura, diagrama de flujo, explicación de módulos y guía de empaquetado (.exe).*
  - b. *Incluir ejemplos de ejecución, formato de salida y recomendaciones legales/éticas (solo usar con permisos).*

## 2.3 Criterios de aceptación

- El ejecutable **scanner.exe** debe:
  - Ejecutarse en Windows 10/11 sin necesidad de instalar Python.
  - Solicitar UAC si fue empaquetado con **--uac-admin**.
  - Pedir al usuario host y rango de puertos y completar un escaneo TCP básico mostrando puertos abiertos.
  - Guardar resultados en **scan\_results.json** y **scan\_results.csv** con la estructura documentada.
  - Incluir para cada CVE al menos: ID, descripción corta, CVSS base score y categoría de severidad cuando la NVD lo proporcione.
- El rendimiento debe estar dentro de las metas medibles fijadas en los objetivos SMART (o documentar razones técnicas si no se alcanzan).

## 2.4 Métricas e indicadores de éxito (KPIs)

- **Tiempo total de escaneo** (segundos).
- **Puertos escaneados por segundo** (throughput).
- **Tasa de detección de banners** (% de puertos abiertos que proporcionan banner útil).
- **Número de CVEs retornadas por servicio** (media y mediana).
- **Tasa de falsos positivos/negativos** (documentada mediante pruebas controladas en entornos de laboratorio).

## 2.5 Alcance y limitaciones

- **Alcance:** host único (con posibilidad de extender a rangos en próximas versiones), escaneo TCP completo y UDP básico (opcional), consultas NVD por banner/keywords, exportación JSON/CSV, empaquetado como .exe.
- **Limitaciones:** la correlación con la NVD depende de la calidad del banner; servicios que oculten o no devuelvan versiones disminuirán la precisión. La detección UDP es intrínsecamente más lenta y menos fiable por la naturaleza sin conexión del protocolo. El acceso a la API NVD está sujeto a rate limits y disponibilidad externa.

## 2.6 Entregables

1. Código fuente **scanner.py** (documentado y comentado).
2. Ejecutable **dist/scanner.exe** (opcionalmente con **manual.txt** embebido).
3. **manual.txt** de usuario y **README.md** para desarrolladores.
4. **scan\_results.json** y **scan\_results.csv** (muestras de ejecución).
5. Memoria técnica del proyecto (documento Word/PDF).
6. Script/guía para compilar con PyInstaller (comandos y opciones).

## 2.7 Riesgos y mitigaciones

- **Riesgo:** bloqueo por rate limiting de la API NVD.
  - **Mitigación:** implementar pausas configurables entre llamadas, uso de paginación y caché local de resultados.
- **Riesgo:** detección insuficiente por banners ausentes o ofuscados.
  - **Mitigación:** combinar keywordSearch con búsquedas por puerto y patrones (ej. "Apache", "OpenSSH"), permitir entrada manual de versiones.
- **Riesgo legal/ética:** uso indebido del escáner contra sistemas sin autorización.
  - **Mitigación:** incluir advertencias legales en el manual y en la interfaz; diseñar la herramienta para entornos de prueba y auditorías autorizadas.

### 3. Herramientas y Tecnologías

- **Lenguaje de programación:** Python 3.13.
- **Librerías utilizadas:**
  - `socket` → escaneo de puertos.
  - `asyncio` → paralelización de tareas.
  - `aiohttp` → peticiones a la API de NVD.
  - `json` y `csv` → exportación de reportes.
  - `time` → cronometraje de ejecución.
  - `os`, `sys`, `subprocess` → automatización en Windows.
- **Empaquetado:**
  - `PyInstaller` para generar `.exe`.
  - Opción `--uac-admin` para ejecución con permisos elevados.

## 4. Desarrollo del Código

### 4.1. Detección automática de IP local

El programa detecta automáticamente la IP de la máquina desde la cual se ejecuta, evitando configuraciones manuales iniciales.

```
# autodetección de IP local
detected_ip = detect_local_ip()
# mostrar y ofrecer como valor por defecto
print(f"IP local detectada: {detected_ip}")
target_input = input(f"Introduce IP o hostname objetivo (enter para usar '{detected_ip}'): ").strip()
target = target_input if target_input else detected_ip
if not target:
    print("Host vacío, saliendo.")
    sys.exit(1)
```

### 4.2. Escaneo de puertos

Se utilizan **sockets TCP/UDP** para comprobar la disponibilidad de puertos en paralelo con **asyncio**.

- Si la conexión es exitosa, se marca el puerto como abierto.
- Se realiza un intento de *banner grabbing* enviando peticiones genéricas para identificar versiones de servicio.

### 4.3. Integración con la NVD

Se consulta la **API pública de NVD** para verificar vulnerabilidades asociadas a los servicios detectados.

- Se procesan los campos **cve.CVE\_data\_meta.ID** y **baseMetricV3.cvssV3.baseSeverity** para obtener **ID de CVE** y **severidad CVSS**.
- Esto permite priorizar las vulnerabilidades críticas frente a las de bajo impacto.

CVSS Puntuación Base	CVSS Nivel de Gravedad
0	Ninguna
0.1 - 3.9	Baja
4.0 - 6.9	Media
7.0 - 8.9	Alta
9.0 - 10.0	Crítica



## 4.4. Exportación de resultados

Los resultados se almacenan en **JSON** y **CSV**, con campos como:

- Puerto.
- Estado (abierto/cerrado).
- Servicio detectado.
- CVE relacionados.
- Severidad CVSS.
- Tiempo total del análisis.

## 4.5. Cronómetro de ejecución


El programa mide el tiempo de inicio y finalización del escaneo para informar de la duración exacta de la operación.

```
print(f"\n✅ Escaneo finalizado en {elapsed:.2f} segundos.")
```

## 4.6. Creación de ejecutable

El empaquetado en **.exe** se realizó con:

```
bash
py -m PyInstaller --onefile --uac-admin scanner.py
```

 Copiar código

Opciones añadidas:

- **--uac-admin**: ejecución directa con privilegios elevados.
- **--add-data "manual.txt;."**: incluir manual de uso.

## 5. Manual de Uso

### 5.1. Ejecución

- Ubicar el archivo **scanner.exe** en cualquier directorio.
- Ejecutar con doble clic o desde la terminal.
- Aceptar la ventana de permisos de administrador (UAC).

### 5.2. Opciones

El programa solicitará:

1. **IP objetivo** (por defecto, usa la IP local detectada).
2. **Rango de puertos** a escanear (ejemplo: **20-1000**).
3. **Modo de exportación**: JSON o CSV.

### 5.3. Resultados

- Se generará un archivo **resultados.json** o **resultados.csv** en el mismo directorio.
- Se mostrará en pantalla un resumen con:
  - Total de puertos abiertos.
  - Servicios identificados.
  - Vulnerabilidades detectadas (con CVE y severidad).
  - Tiempo de ejecución.

### 5.4. Requisitos

- Ejecutar como Administrador.
- Conexión a Internet para consultar la API de NVD.
- Windows 10 / 11

## 6. Conclusiones

La herramienta desarrollada constituye un **mini escáner de seguridad automatizado**, integrando descubrimiento de puertos, identificación de servicios y correlación con vulnerabilidades de la NVD.

Sus principales ventajas son:

- **Portabilidad** gracias al formato **.exe**.
- **Automatización total** (no requiere configuración avanzada).
- **Eficiencia** gracias al uso de **asyncio**.
- **Orientación práctica** para usuarios en pruebas rápidas de seguridad.

Como líneas de mejora futuras:

- Implementar una interfaz gráfica (GUI) en Tkinter o PyQt.
- Añadir soporte para escaneo distribuido en múltiples hosts.
- Generar reportes en PDF con gráficos.
- Integración con bases de datos para gestión centralizada de resultados.
- Implementación de IA para soluciones personalizadas

## 7. Bibliografía

- Anderson, R. (2020). *Security engineering: A guide to building dependable distributed systems* (3rd ed.). Wiley.
- National Institute of Standards and Technology. (s.f.). *National Vulnerability Database*. <https://nvd.nist.gov>
- Python Software Foundation. (s.f.). *Python 3.13 documentation*. <https://docs.python.org/3.13/>
- PyInstaller. (s.f.). *PyInstaller official documentation*. <https://pyinstaller.org>
- SANS Institute. (s.f.). *Port scanning techniques*. <https://www.sans.org>
- OpenAI. (2025). *ChatGPT (GPT-5 mini) [Large language model]*. <https://openai.com/chatgpt>