

# PRUEBAS DE SEGURIDAD ACTIVA Y ANÁLISIS PROFUNDO PARA APLICACIONES WEB

## Proyecto: Scan Pro Vulnerability Web

### Descripción del Programa

El software es una herramienta eficaz para la detección y evaluación de vulnerabilidades en aplicaciones web. Basado en las principales amenazas identificadas por OWASP, el sistema integra diversas técnicas de análisis de seguridad, como pruebas de inyección (SQLi, XXE, LFI), detección de configuración insegura y simulaciones de ataques como CSRF, SSRF y desbordamiento de búfer.

La implementación de una interfaz en **Flet** facilita su uso, permitiendo a los usuarios analizar de manera intuitiva y eficiente la seguridad de sus plataformas. Además, el software no solo identifica riesgos, sino que proporciona información detallada para su mitigación, contribuyendo a un entorno digital más seguro.

Con estas capacidades, la herramienta se posiciona como un recurso valioso en el ámbito de la ciberseguridad, ayudando a desarrolladores y profesionales de seguridad a detectar y corregir vulnerabilidades antes de que sean explotadas por atacantes.

### Características principales:

- Detección de vulnerabilidades basadas en OWASP Top 10

OWASP Top 10 (2021)	Ejemplo en tu Análisis
A01 – Inyección	SQLi, XXE, LFI, Path Traversal
A02 – Fallas en la Autenticación	Race Condition, Desbordamiento de búfer
A03 – Exposición de Datos Sensibles	SSL/TLS, HSTS, Encabezados HTTP
A04 – Inseguridad en el Diseño	Cacheo Inseguro, CSRF
A05 – Fallas en Configuración	Host Header Injection, Headers HTTP
A06 – Componentes Vulnerables	Subdomain Takeover, dependencias inseguras
A07 – Control de Acceso Defectuoso	Directory Traversal, LFI
A08 – CSRF	Protección contra CSRF
A09 – Fallas en Registro y Monitoreo	Uso de logger en detección de ataques
A10 – SSRF	Prueba de Server-Side Request Forgery (SSRF)

- Análisis de seguridad en cabeceras HTTP y respuestas del servidor.
- Escaneo automatizado de URLs para identificar configuraciones débiles.
- Uso de peticiones asíncronas para mejorar el rendimiento del escaneo.
- Posibilidad de integración con otras herramientas de análisis de seguridad.

- Registro detallado de hallazgos y generación de reportes.

### **Objetivos iniciales:**

Desarrollar un software especializado en el análisis profundo y detallado de vulnerabilidades y pruebas de penetración, enfocado en la protección integral de aplicaciones web. Este sistema permitirá identificar, evaluar y mitigar riesgos de seguridad mediante técnicas avanzadas de detección y simulación de ataques, garantizando la robustez y confiabilidad de los entornos digitales.

## **PROYECTOS SIMILARES**

### **Wapiti**

Wapiti es una herramienta de análisis de vulnerabilidades para aplicaciones web diseñada para ayudar a los desarrolladores, administradores y profesionales de la ciberseguridad a identificar fallas de seguridad en sus aplicaciones. Es un proyecto de código abierto, escrito en Python, que se centra en la automatización de pruebas de penetración para descubrir vulnerabilidades antes de que puedan ser explotadas por atacantes malintencionados.

Recorre el sitio web para identificar formularios, scripts y otros puntos de entrada que puedan ser vulnerables.

Funciona con aplicaciones web que utilizan HTTP y HTTPS.

Github: <https://github.com/sullo/nikto>

### **Dshell**

Es una herramienta de código abierto basada en Python que permite el desarrollo de módulos de análisis personalizados para comprender eventos de intrusión cibernética. Ofrece análisis profundo de paquetes, reensamblaje robusto de flujos y soporte para IPv4 e IPv6. Además, incluye geolocalización y mapeo de IP a ASN para cada conexión, facilitando la comprensión del tráfico de red.

Facilita la extracción de datos relevantes, como direcciones IP, puertos, protocolos y patrones de comunicación.

Ofrece una línea de comandos intuitiva para ejecutar análisis y explorar resultados.

Github: <https://github.com/USArmyResearchLab/Dshell/tree/master>

### **MobSF**

Es una herramienta de código abierto diseñada para realizar análisis de seguridad en aplicaciones móviles. Es ampliamente utilizada por profesionales

de ciberseguridad, desarrolladores y analistas para identificar vulnerabilidades en aplicaciones móviles, tanto en sus versiones compiladas como en su código fuente.

Compatible con aplicaciones Android (APK), iOS (IPA) y Windows (APPX).

Detecta patrones y comportamientos asociados con aplicaciones maliciosas.

Github: <https://github.com/MobSF/Mobile-Security-Framework-MobSF>

### **AndroBugs Framework**

Es una herramienta de análisis de seguridad para aplicaciones Android. Fue creada por el investigador de seguridad Kenny Leung. Esta herramienta se utiliza para realizar análisis estáticos de las aplicaciones Android, buscando vulnerabilidades y posibles riesgos de seguridad en el código fuente de las aplicaciones.

Funciona con cualquier archivo APK, lo que lo hace ideal tanto para aplicaciones en desarrollo como para aquellas ya publicadas.

Examina el código fuente o el archivo APK de una aplicación sin necesidad de ejecutarla.

Github: [https://github.com/AndroBugs/AndroBugs\\_Framework](https://github.com/AndroBugs/AndroBugs_Framework)

## **DESCRIPCION DE TODOS LOS VULNERBAILDADES QUE ESCANEA EL SOFTWARE**

### **Análisis de Seguridad General**

#### **Escaneo de puertos**

Analiza los **puertos abiertos** en el servidor, detectando servicios expuestos.

#### **Verificación de Certificado SSL/TLS**

Comprueba si el sitio usa **HTTPS seguro** y si el certificado es válido.

#### **Verificación de encabezados**

Revisa encabezados HTTP para detectar configuraciones incorrectas o faltantes, como:

- X-Frame-Options (protección contra Clickjacking)
- Content-Security-Policy (mitigación de XSS)

### **Vulnerabilidades Críticas**

#### **Buscando Vulnerabilidades SQL Injection (SQLi)**

Intenta inyectar código malicioso en formularios o URL para acceder a la base de datos.

### **Verificación de protecciones CSRF**

Prueba si el sitio está protegido contra ataques de **Cross-Site Request Forgery**.

### **Probando Directory Traversal**

Busca fallas en rutas para acceder a archivos sensibles (/etc/passwd, config.php).

### **Buscando Local File Inclusion (LFI)**

Verifica si se pueden incluir archivos locales en la página, lo que puede exponer información del servidor.

### **Probando Server-Side Request Forgery (SSRF)**

Simula un ataque donde el servidor accede a recursos internos o externos no autorizados.

### **Verificando protecciones Clickjacking**

Evalúa si el sitio es vulnerable a ataques donde se superpone una página falsa sobre la real.

### **Probando Open Redirect**

Busca si la web permite redireccionamientos a sitios no confiables.

### **Probando XML External Entity (XXE)**

Envía payloads XML maliciosos para extraer datos internos del servidor.

### **Verificando posibles Subdomain Takeovers**

Analiza si un subdominio mal configurado puede ser **tomado por un atacante**.

### **Probando Deserialización Insegura**

Intenta cargar objetos maliciosos para ejecutar código en el servidor.

### **Buscando scripts inseguros (XSS)**

Evalúa si el sitio permite **Cross-Site Scripting**, inyectando código malicioso en formularios.

## **Pruebas de Estabilidad y Seguridad Avanzada**

### **Detectar y simular condiciones de carrera en el servidor**

Evalúa **Race Conditions**, donde múltiples solicitudes simultáneas pueden afectar la lógica de la aplicación.

### **Prueba de carga con múltiples solicitudes**

Simula **múltiples usuarios simultáneos** para analizar el rendimiento del sitio.

### **Prueba de desbordamiento de búfer**

Intenta explotar errores de memoria enviando datos excesivos.

### **Verificación de Host Header Injection**

Prueba si el encabezado Host puede ser manipulado para acceder a sitios no autorizados.

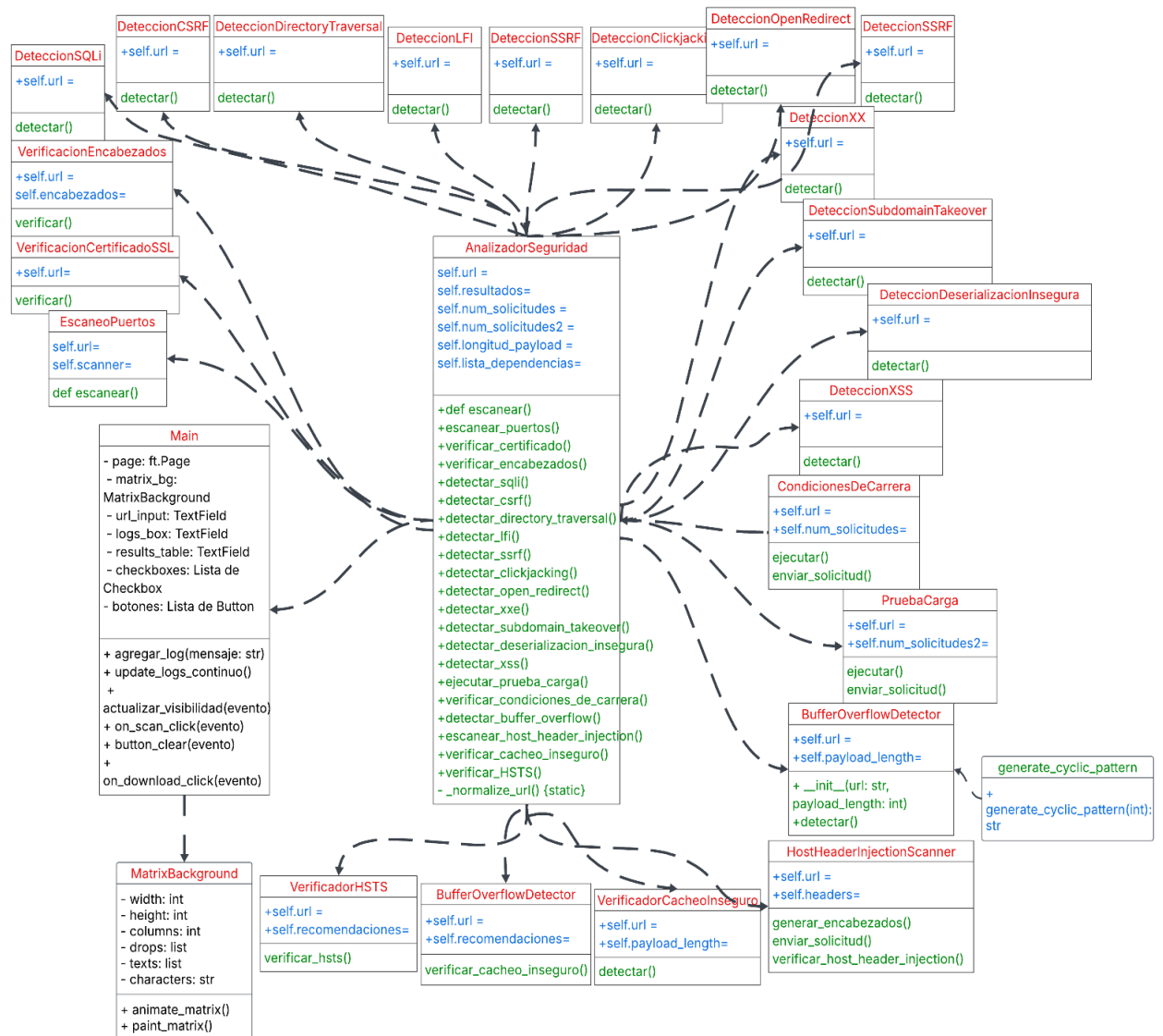
## Análisis de Cacheo Inseguro

Examina si el servidor almacena **información sensible en caché**, lo que puede causar filtraciones de datos.

## Verificación de HSTS

Confirma si el sitio usa **HTTP Strict Transport Security** para obligar conexiones HTTPS seguras.

## Diagrama UML



Pre requisitos para que funcione correctamente el programa:

Instalar aiohttp usando pip install aiohttp

Instalar BeautifulSoup usando pip install BeautifulSoup

Instalar bs4 usando pip install bs4

Instalar nmap usando pip install python-nmap

Instalar flet usando pip install flet

Instalar request usando pip install requests

Instalar pyfiglet pip install pyfiglet

## Convertir a un archivo exe para Windows:



**Recomendación:** Desactivar el antivirus para que no lo elimine al encontrarlo como un archivo infectado

### Pasos para ejecutar correctamente:

#### 1. Abre la terminal o línea de comandos

En **Windows**, usa cmd o PowerShell e instalamos pyinstaller

```
sh  
  
pip install pyinstaller
```

 Copiar  Editar

#### 2. Ubicar los archivos en una misma carpeta

Asegúrate de que en la misma carpeta tengas estos archivos:


**scanvulnerbility.py** → Tu script en Python.

**imagen.ico** → El ícono que quieres usar (opcional).

#### 3. Navega hasta la carpeta donde están los archivos

Si tus archivos están en C:\Users\TuUsuario\Proyecto, ejecuta:

```
bash  
  
cd C:\Users\TuUsuario\Proyecto
```



 Copiar  Editar

#### 4. Crear el ejecutable

Ejecuta el siguiente comando:

- convertir scripts de Python (.py) en archivos ejecutables (.exe en Windows) **sin ícono**

```
sh  
  
pyinstaller --onefile -w scanvulnerbility.py
```

 Copiar  Editar



**pyinstaller:** Es el comando principal de PyInstaller.

**onefile:** Genera un único archivo ejecutable en lugar de una carpeta con múltiples archivos.

**-w:** (Equivalente a `--windowed`) Oculta la consola de comandos al ejecutar el programa, útil para aplicaciones con interfaz gráfica.

- convertir scripts de Python (.py) en archivos ejecutables (.exe en Windows) **con icono**

bash

 Copiar  Editar

```
pyinstaller --onefile --icon=imagen.ico scanvulnerbility.py
```



**--onefile:** Empaqueta todo en un solo archivo .exe, en lugar de generar múltiples archivos.

**--icon=imagen.ico:** Asigna un ícono personalizado (imagen.ico) al ejecutable.

**scanvulnerbility.py:** Es el script que quieres convertir en un .exe.

Si **NO** quieres que se abra una ventana de consola al ejecutar el .exe, usa:

bash

 Copiar  Editar

```
pyinstaller --onefile --windowed --icon=imagen.ico scanvulnerbility.py
```

```
330 INFO: PyInstaller: 6.12.0, contrib hooks: 2025.1
330 INFO: Python: 3.11.2
348 INFO: Platform: Windows-10-10.0.22631-SP0
348 INFO: Python environment: C:\Users\Nilver\AppData\Local\Programs\Python\Python311
348 INFO: wrote C:\Users\Nilver\Downloads\Proyecto finalizado\Scanvulnerbility.spec
358 INFO: Module search paths (PYTHONPATH):
['C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\Scripts\\pyinstaller.exe',
 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\python311.zip',
 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\DLLs',
 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\Lib',
 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311',
 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages',
 'C:\\Users\\Nilver\\Downloads\\Proyecto finalizado']
926 INFO: checking Analysis
926 INFO: Building Analysis because Analysis-00.toc is non existent
926 INFO: Running Analysis Analysis-00.toc
926 INFO: Target bytecode optimization level: 0
926 INFO: Initializing module dependency graph...
928 INFO: Initializing module graph hook caches...
961 INFO: Analyzing modules for base_library.zip ...
2152 INFO: Processing standard module hook 'hook-encodings.py' from 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyI
nstaller\\hooks'
2883 INFO: Processing standard module hook 'hook-heapq.py' from 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInsta
ller\\hooks'
3845 INFO: Processing standard module hook 'hook-pickle.py' from 'C:\\Users\\Nilver\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInst
aller\\hooks'
```






```

28653 INFO: Extra DLL search directories (PATH): []
29543 INFO: Warnings written to C:\Users\Nilver\Downloads\Proyecto finalizado\build\Scanvulnerability\warn-Scanvulnerability.txt
29702 INFO: Graph cross-reference written to C:\Users\Nilver\Downloads\Proyecto finalizado\build\Scanvulnerability\xref-Scanvulnerability.html
29761 INFO: checking PYZ
29761 INFO: Building PYZ because PYZ-00.toc is non existent
29761 INFO: Building PYZ (ZlibArchive) C:\Users\Nilver\Downloads\Proyecto finalizado\build\Scanvulnerability\PYZ-00.pyz
31535 INFO: Building PYZ (ZlibArchive) C:\Users\Nilver\Downloads\Proyecto finalizado\build\Scanvulnerability\PYZ-00.pyz completed successfully.
31610 INFO: checking PKG
31610 INFO: Building PKG because PKG-00.toc is non existent
31610 INFO: Building PKG (CArchive) Scanvulnerability.pkg
39107 INFO: Building PKG (CArchive) Scanvulnerability.pkg completed successfully.
39121 INFO: Bootloader C:\Users\Nilver\AppData\Local\Programs\Python\Python311\Lib\site-packages\PyInstaller\bootloader\Windows-64bit-intel\runw.exe
39121 INFO: checking EXE
39121 INFO: Building EXE because EXE-00.toc is non existent
39121 INFO: Building EXE from EXE-00.toc
39121 INFO: Copying bootloader EXE to C:\Users\Nilver\Downloads\Proyecto finalizado\dist\Scanvulnerability.exe
39128 INFO: Copying icon to EXE
39134 INFO: Copying 0 resources to EXE
39134 INFO: Embedding manifest in EXE
39138 INFO: Appending PKG archive to EXE
39178 INFO: Fixing EXE headers
39319 INFO: Building EXE from EXE-00.toc completed successfully.
39333 INFO: Build complete! The results are available in: C:\Users\Nilver\Downloads\Proyecto finalizado\dist

```

## 5. Ubicar el archivo .exe generado

Cuando termine el proceso, busca en la carpeta **dist**

▼ hoy				
	Scanvulnerability.spec	2/03/2025 01:23	Archivo SPEC	1 KB
	Scanvulnerability.py	2/03/2025 01:20	Archivo de origen ...	83 KB
	dist	2/03/2025 01:24	Carpeta de archivos	
	build	2/03/2025 01:23	Carpeta de archivos	
▼ ayer				
	logo.ico	1/03/2025 07:16	Archivo ICO	75 KB

## 6. Probar el .exe

Para ejecutar tu programa, ve a la carpeta dist/ y abre scanvulnerability.exe.

	Scanvulnerability.exe	2/03/2025 01:24	Aplicación	36,712 KB
---	-----------------------	-----------------	------------	-----------





Si usaste **--windowed**, no verás la consola.

Si hay errores, revisa el archivo **scanvulnerability.spec** para modificar la configuración.

## 7. Opcional: Empaquetar el archivo .exe

Si quieres compartir tu ejecutable, puedes **comprimir** la carpeta **dist/** en un **.zip**.

Para mejorar la compatibilidad en otros equipos, usa:

```
bash

pyinstaller --onefile --noconsole --icon=imagen.ico scanvulnerability.py
```

## Ejecutar el programa en un Kali linux:

Ahora clona del github el repositorio en un Kali:

Antes de hacer cualquier instalación, revisa si ya tienes Git con:

```
sh

git --version
```



si no tienes instalado de esta forma:

```
sh

sudo apt update && sudo apt install git -y
```

Luego clona del sitio el repositorio:



sh

 Copiar  Editar

```
git clone https://github.com/SR10SCAN/Escaneo-de-Vulnerabilidad-de-paginas-Web.git
```



Usar un Entorno Virtual (Segura y Recomendada) en Kali sigue estos pasos:

sh

 Copiar  Editar

```
sudo apt install python3-venv -y
```



sh

 Copiar  Editar

```
python3 -m venv venv  
source venv/bin/activate
```

### 3. Instala flet dentro del entorno virtual



sh

 Copiar  Editar

```
pip install "flet[all]"
```



Usando un Entorno Virtual (Si estás en un venv) instala lo siguiente

sh

 Copiar  Editar



```
pip install aiohttp
```

sh

 Copiar  Editar

```
pip install beautifulsoup4
```

sh

 Copiar  Editar

```
pip install python-nmap
```

Finalmente ejecuta el archivo .py

```
sh

python3 scanvulnerbility.py
```

Copiar Editar

Y después si quieres salir pones en el terminal “exit”

Y si quiere ingresar nuevamente y ha seguido los pasos anteriores ingrese lo siguiente:

```
sh

cd ~/Escaneo-de-Vulnerabilidad-de-paginas-Web
```

Copiar Editar

```
sh

source venv/bin/activate
```

Copiar Editar

```
(venv)-(kali@kali)-[~/Escaneo-de-Vulnerabilidad-de-paginas-Web]
$
```

Luego solo ejecuta es archivo .py

```
sh

python3 scanvulnerbility.py
```

Copiar Editar

The screenshot displays a Kali Linux terminal on the left and the Cyber Scan Pro Vulnerability Web application interface on the right. The terminal shows the execution of commands to navigate to the project directory, activate the virtual environment, and run the scanvulnerbility.py script. The application interface, titled "CYBER SCAN PRO VULNERABILITY WEB", features a dark theme with green accents and lists various security checks under three main categories: Seguridad de Red, Ataques de Inyección, and Seguridad de Servidor.

**Seguridad de Red**

- Escaneo de puertos
- Verificación de Certificado SSL/TLS
- Verificación de encabezados

**Ataques de Inyección**

- Buscando Vulnerabilidades SQL injection
- Verificación de protecciones CSRF
- Probando Directory Traversal
- Buscando Local File Inclusion
- Probando Server-Side Request Forgery
- Verificando protecciones Clickjacking
- Probando Open Redirect
- Probando XML External Entity
- Verificando posibles Subdomain Takeovers
- Probando Deserialización Insegura
- Buscando scripts inseguros

**Seguridad de Servidor**

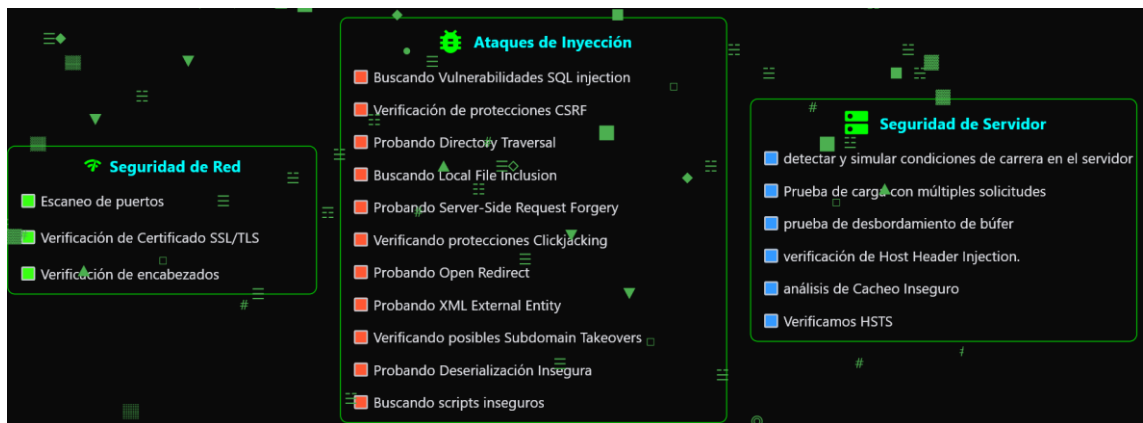
- detectar y simular condiciones de carrera en el servidor
- Prueba de carga con múltiples solicitudes
- prueba de desbordamiento de búfer
- verificación de Host Header Injection.
- análisis de Cacheo Inseguro
- Verificamos HSTS

## Guía de Uso del software:

1. Ingreso de la URL objetivo y selección de pruebas a realizar.

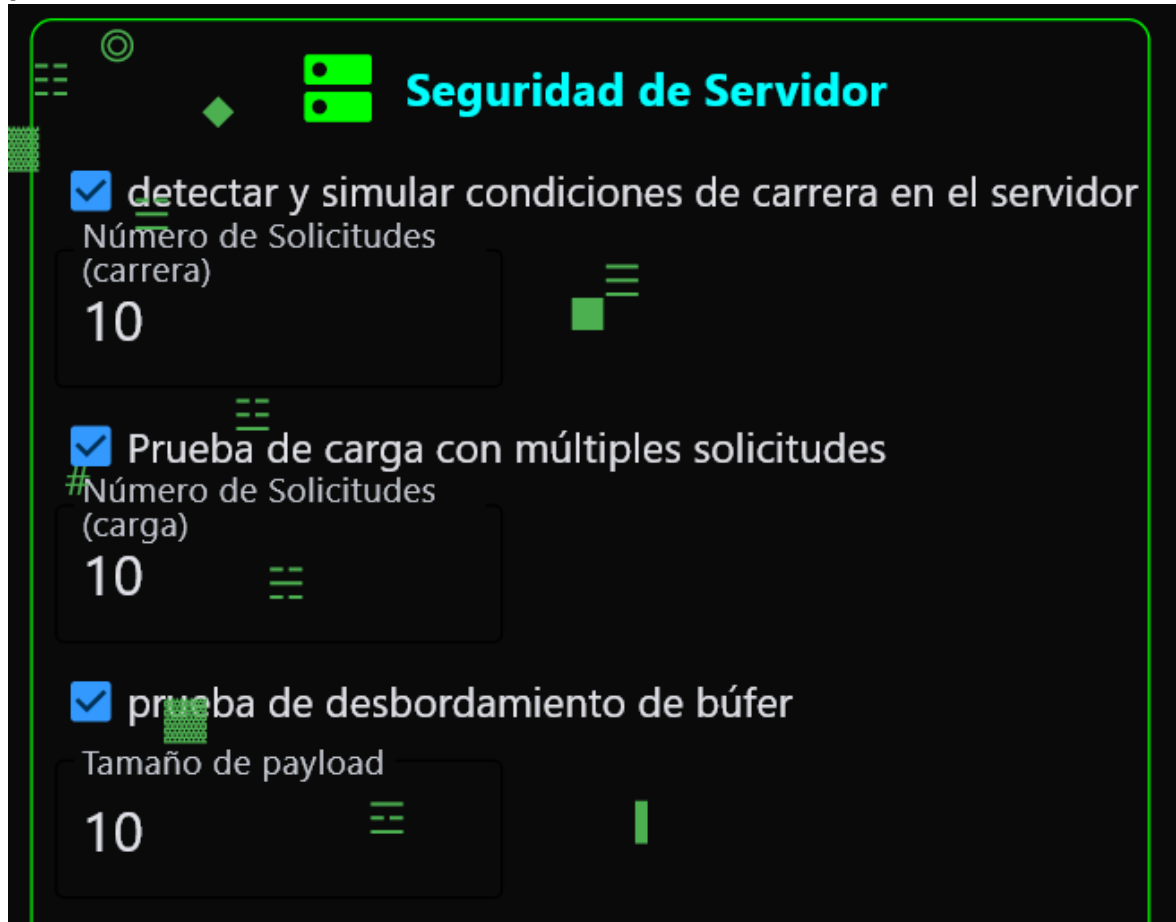


2. Escogemos cualquiera de las vulnerabilidades a escanear haciendo click en los casilleros:

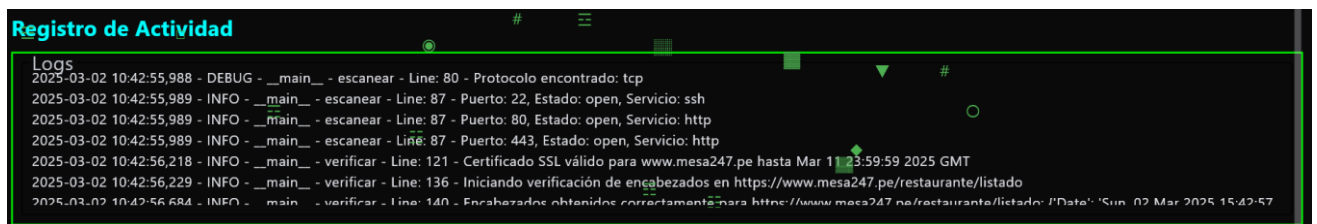


3. Envío de solicitudes maliciosas simuladas para evaluar la respuesta del servidor. En esta parte tenemos 3 vulnerabilidades que se

puede analizar:



4. En la sección de logs después de un escaneo de vulnerabilidades, generalmente se encuentran registros detallados sobre el análisis realizado.



5. El Cuadro de Resultados muestra los hallazgos del escaneo de vulnerabilidades. Dependiendo del tipo de vulnerabilidad detectada, se incluyen detalles adicionales como cabeceras HTTP, puertos abiertos o respuestas del servidor

```
Resultados
Puerto 443: OPEN (Servicio: http)

Verificación de Certificado SSL/TLS:
Certificado válido hasta: Mar 11 23:59:59 2025 GMT

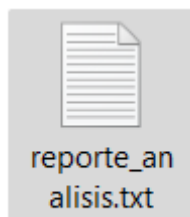
Verificación de encabezados HTTP:
{
  "Date": "Sun 02 Mar 2025 15:42:57 GMT"
```

## 6. Generación de un informe.

Hacemos click en:

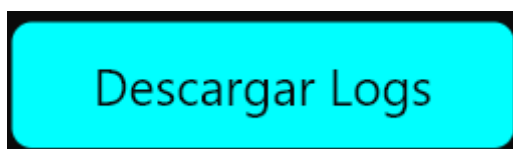


Luego de ello visualizaras un archivo llamado "reporte\_analisis" en la carpeta de descargas

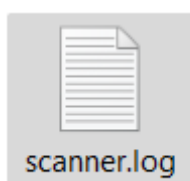


## 7. Generación de un informe de logs.

Hacemos click en:



Luego de visualizaras un archivo llamado "scanner.log" en la carpeta de descargas

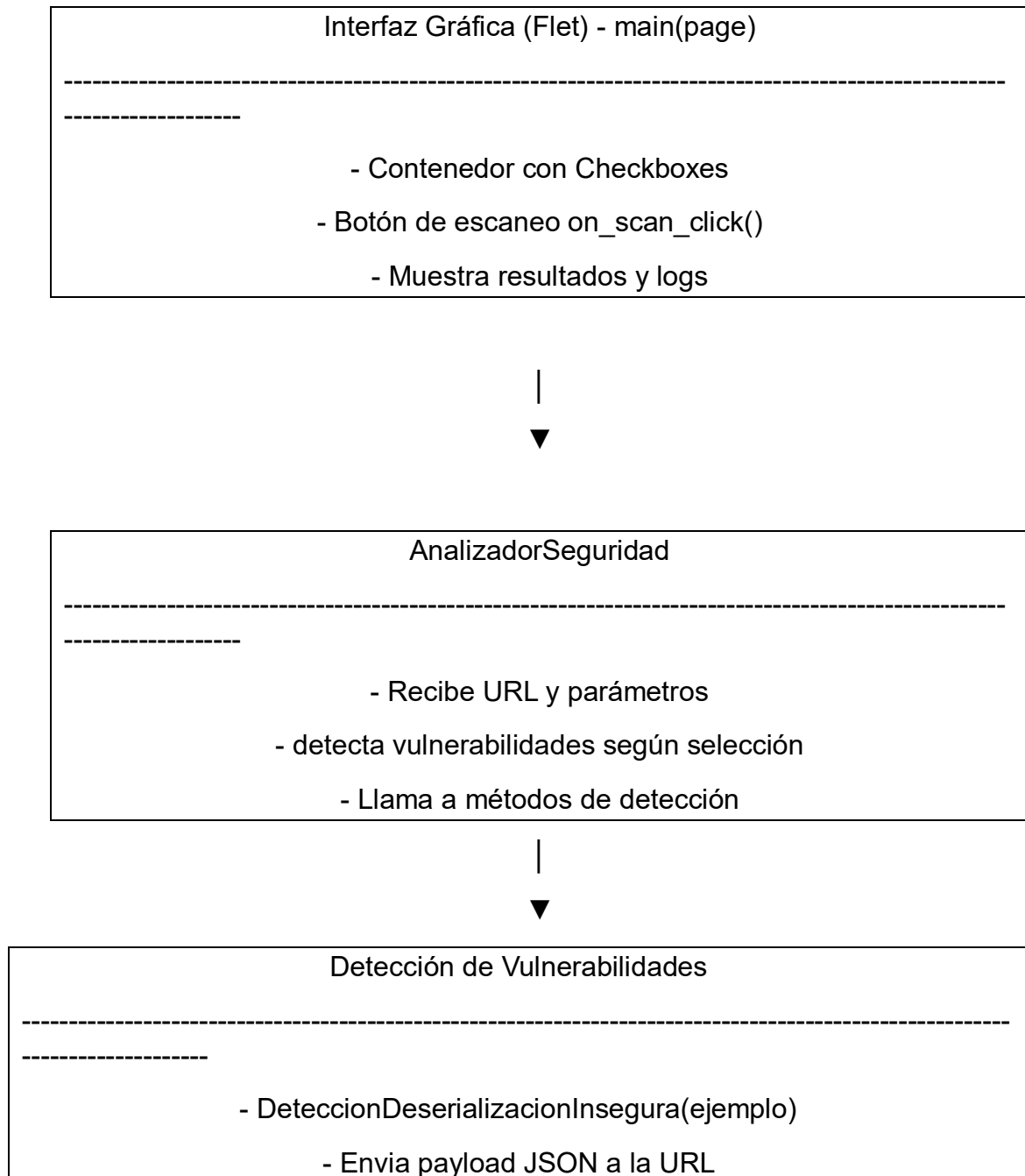


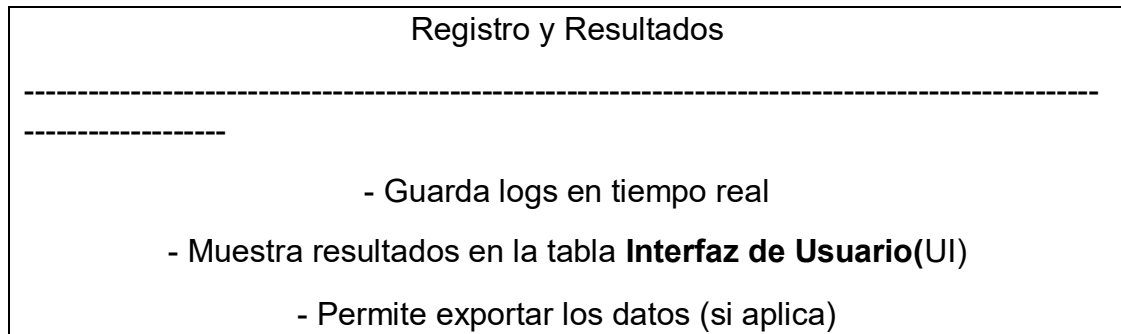
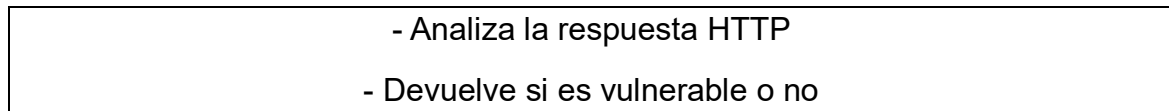
8. Finalmente limpiamos los cuadros dónde se ingresó el url y el cuadro de resultados con el botón “limpiar”:



### Funcionamiento Interno

- **Flujo de ejecución** (puedes incluir un diagrama de flujo).





- **Componentes principales** (explicar las partes clave del código).

## 1. Clase de Detección de Vulnerabilidades

Cada vulnerabilidad tiene su propia clase.  
por ejemplo:

```
class DeteccionXXE:
    def __init__(self, url):
        self.url = url

    async def detectar(self):
        logger.info("Probando XML External Entity...")
        xml_payloads = ["""<?xml version="1.0"?><!DOCTYPE data [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]><data>&xxe;</data>"""]
        async with aiohttp.ClientSession() as session:
            headers = {'Content-Type': 'application/xml'}
            for i, xml_payload in enumerate(xml_payloads):
                try:
                    async with session.post(self.url, data=xml_payload.encode('utf-8'), headers=headers, ssl=False) as response:
                        content = await response.text()
                        if "root:" in content:
                            logger.warning(f"Vulnerabilidad XXE detectada con payload {i+1}")
                            return True
                except Exception as e:
                    logger.error(f"Error probando XXE: {e}")
            return False
```

## 2. Clase AnalizadorSeguridad (Orquestador de Escaneos)

Esta clase gestiona la ejecución de múltiples pruebas de seguridad.



```

class AnalizadorSeguridad:
    def __init__(self, url: str, num_solicitudes: int = 10, num_solicitudes2: int = 10, longitud_payload: int = 1000):
        self.url = self._normalize_url(url)
        self.resultados: Dict[str, Any] = {}
        self.num_solicitudes = num_solicitudes
        self.num_solicitudes2 = num_solicitudes2
        self.longitud_payload = longitud_payload
        self.lista_dependencias = []

    @staticmethod
    def _normalize_url(url: str) -> str:
        if not url.startswith(('http://', 'https://')):
            return 'https://' + url
        return url

```

```

def detectar_deserializacion_insegura(self) -> Tuple[str, str]:
    #Ejecuta la detección de deserialización insegura mediante payloads JSON."""
    deteccion = DeteccionDeserializacionInsegura(self.url)
    try:
        resultado = asyncio.run(deteccion.detectar())
        estado = "Vulnerable" if resultado else "No vulnerable"
        return "Deserialización Insegura", estado
    except Exception as e:
        return "Deserialización Insegura", f"Error: {e}"

```

### Descripción de la imagen

Se **instancia** la clase DeteccionDeserializacionInsegura.

Se **ejecuta el análisis** con asyncio.run().

Se **devuelve el resultado** indicando si el sitio es vulnerable

### 3. Interfaz de Usuario (UI) con Flet

La interfaz permite seleccionar vulnerabilidades y ver los resultados.

```

check_DeteccionDeserializacionInsegura = ft.Checkbox(
    label="Detectar Deserialización Insegura",
    fill_color="#FF5733",
    label_style=ft.TextStyle(size=18)
)

```

#### Explicación:

**Checkbox** para que el usuario seleccione si quiere ejecutar este escaneo.

```
def on_scan_click(e):
    resultado_final = ""
    if check_DeteccionDeserializacionInsegura.value:
        nombre, resultado = analizador.detectar_deserializacion_insegura()
        resultado_final += f"{nombre}:\n{resultado}\n\n"
    results_table.value = resultado_final
    page.update()
```

### Explicación:

Si el usuario marca el **checkbox**, se llama al método `detectar_deserializacion_insegura()`.

Ejemplo: Registro de una vulnerabilidad detectada.

## 4. Registro de Logs con logger

```
logger.warning("Vulnerabilidad XXE detectada")
```

### Explicación:

Cuando se detecta una vulnerabilidad, se **genera un mensaje de advertencia** en el log.

El programa almacena información sobre cada escaneo en el log.

Se actualiza la **tabla de resultados** en la UI.

## Algoritmos Utilizados en el Escáner de Vulnerabilidades

### 1. Programación Orientada a Objetos (POO)

- Se implementan clases para cada tipo de vulnerabilidad, como:
  - **DeteccionXXE**
  - **DeteccionDeserializacionInsegura**
- Cada clase encapsula su lógica dentro de un método `detectar()`, lo que facilita la escalabilidad y reutilización.

### 2. Programación Asíncrona con `asyncio` y `aiohttp`

- Se usa `asyncio` para ejecutar múltiples escaneos de forma simultánea sin bloquear el programa.

- aiohttp permite hacer solicitudes HTTP de manera eficiente.
- **Ejemplo:**
  - Se crea una sesión asíncrona con `aiohttp.ClientSession()`.
  - Se envía una petición async with `session.post(...)`.
  - Se obtiene la respuesta de manera no bloqueante con `await response.text()`.

### 3. Envío de Payloads Maliciosos para la Detección

- Se diseñan **payloads específicos** según la vulnerabilidad:
  - **XXE (XML External Entity):** Se envía un XML malicioso para acceder a archivos del servidor.
  - **Deserialización Insegura:** Se manda un JSON manipulado para forzar la ejecución de código.
- Se analizan las respuestas HTTP para determinar si la vulnerabilidad existe.

### 4. Análisis de Respuestas HTTP

- Se observa el código de estado (`response.status`).
- Se analizan patrones en la respuesta (`await response.text()`).
- **Ejemplo de detección:**
  - Si la respuesta incluye "root:", es una señal de vulnerabilidad en XXE.
  - Si el código de respuesta es **500**, puede indicar deserialización insegura.

### 5. Registro de Eventos con logger

- Se usa logger para almacenar información relevante sobre los análisis.
- Se registran:
  - Mensajes de inicio y finalización del escaneo.
  - Alertas cuando se detecta una vulnerabilidad.
  - Errores durante la ejecución.

### 6. Interfaz de Usuario con Flet

- Se utiliza **Flet** para construir una interfaz interactiva.
- Se implementan:
  - **Checkboxes** para seleccionar qué vulnerabilidades analizar.
  - **Botón** que activa el escaneo (`on_scan_click()`).

- **Tabla de logs y resultados** donde se muestran los resultados en tiempo real.

## Interfaz del programa



## Conclusión y Futuras Mejoras

El desarrollo de este software especializado en **análisis de seguridad en aplicaciones web** ha permitido la detección y evaluación de múltiples vulnerabilidades críticas, basadas en los riesgos identificados por **OWASP**. Gracias a la implementación de técnicas avanzadas de prueba, como inyecciones de código (**SQLi, XXE, LFI**), análisis de encabezados y certificados, verificación de configuraciones inseguras y simulación de ataques (**CSRF, SSRF, desbordamiento de búfer**), se ha logrado crear una herramienta capaz de **fortalecer la seguridad digital de manera automatizada y eficiente**.

Además, la **interfaz intuitiva desarrollada en Flet** facilita el uso del sistema, permitiendo a los usuarios analizar sus plataformas de forma accesible y con resultados detallados. Esta solución contribuye a la protección de entornos web frente a amenazas cibernéticas, mejorando la seguridad y reduciendo la exposición a ataques.

## Futuras Mejoras y Expansión

Si bien el software ya ofrece un análisis profundo de seguridad, existen oportunidades para futuras mejoras, como:

### Ampliación del análisis y detección de vulnerabilidades

- **Escaneo avanzado y fuzzing:** Incorporación de técnicas de fuzzing para detectar vulnerabilidades mediante pruebas automatizadas con variaciones dinámicas en los datos de entrada.
- **Incluir un escáner activo y pasivo con más técnicas de ataque:** Mejorar la capacidad de detección con análisis pasivos (sin interacción directa) y activos (con pruebas intrusivas).
- **Integración con bases de datos de amenazas en tiempo real:** Implementar una conexión con fuentes de inteligencia de amenazas para identificar riesgos emergentes con mayor precisión.

### **Optimización del rendimiento y escalabilidad**

- **Optimización del rendimiento para escaneos más grandes y rápidos:** Mejorar la eficiencia del análisis para soportar auditorías de seguridad en aplicaciones web complejas y de alto tráfico.

### **Intercepción y manipulación de tráfico**

- **Implementación de un proxy de interceptación HTTP:** Capturar, analizar y modificar tráfico en tiempo real, permitiendo evaluar la seguridad de las comunicaciones entre cliente y servidor.
- **Agregar una función de modificación de solicitudes y respuestas:** Permitir a los usuarios alterar parámetros y contenido de las peticiones HTTP para pruebas manuales más avanzadas.
- **Manipulación de parámetros y payloads personalizados:** Generación de payloads específicos para pruebas más precisas, simulando ataques sofisticados como SQL Injection, XSS o XXE.

### **Automatización de pruebas y explotación de vulnerabilidades**

- **Automatización de explotación guiada:** Desarrollar módulos que guíen en la explotación controlada de vulnerabilidades detectadas, proporcionando estrategias de mitigación y recomendaciones.