PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

Trabajo de Titulación

Tema:

Desarrollo De Un Prototipo De Una Herramienta Para La Evaluación De Seguridad En Entornos Web Y Proporcionar Recomendaciones Para La Mitigación De Vulnerabilidades Comunes.

AUTOR:

PAOLO RAMIRO ORTIZ VILLAMAR

QUITO DM, 20 DE ABRIL DE 2025

DEDICATORIA

Esta tesis nació del esfuerzo diario y de las ganas de entender a fondo cada tema que se presentó en la carrera. Fueron muchas horas de lectura, de prueba y error, y de levantarse con hambre de aprender algo nuevo. Cada paso, por pequeño que pareciera, fue valioso para armar este trabajo y demostrar que con constancia se llega lejos.

Se entrega, sobre todo, a mi familia, que estuvo ahí en las buenas y en las malas. Su apoyo incondicional llenó de energía cada jornada de estudio y evitó que el cansancio pesara demasiado. Con su ánimo siempre presente, ningún reto pareció imposible.

A mis padres, gracias por inculcarme desde chiquito el valor del esfuerzo y de la disciplina. Con su ejemplo comprendí que el éxito no llega solo, sino que se construye día a día. Ellos crearon el ambiente perfecto para que me apasionara por investigar y profundizar en cada tema.

A mi mamá, esa mezcla de fortaleza y cariño que abre camino cuando todo parece confuso. Sus palabras de aliento fueron el empujón justo en los días complicados, y su ternura ayudó a que cada tropiezo se convirtiera en ganas de seguir adelante.

A mis hermanos, que con sus bromas y charlas me recordaron que distraerse no es pecado. Gracias a sus risas, aprendí que un descanso bien utilizado recarga el ánimo y mejora la concentración. Compartir momentos de relax fue clave para retomar el estudio con pilas renovadas.

A los profes y tutores que compartieron su experiencia y no dudaron en corregir cada detalle. Sus consejos metodológicos y sus críticas constructivas subieron el nivel de este prototipo. Sin ellos, muchos conceptos no habrían alcanzado la profundidad y claridad que necesitaban.

A los compañeros de la carrera, esos cómplices de largas sesiones de estudio y debates improvisados. Con ustedes descubrí que las ideas mejoran al intercambiarlas, y que el trabajo en equipo es el secreto para afinar cualquier proyecto.

Y al final, un reconocimiento a todas las comunidades académicas y de código abierto en ciberseguridad. Sus estándares, guías y librerías fueron el apoyo teórico y práctico que hizo posible este trabajo. Esto demuestra que el conocimiento crece de forma colaborativa y nos

enriquece a todos.

AGRADECIMIENTO

.

Este proyecto no habría visto la luz sin el empujón constante de la PUCE. Gracias a la Facultad de Ingeniería y, sobre todo, a la carrera de TI, que siempre brindaron el espacio, el material y el respaldo para armar este prototipo. Cada clase, cada ejercicio y cada reto dejaron su marca en el resultado final.

Un gran saludo a todos los ingenieros que compartieron su saber y no dudaron en dar feedback duro pero útil. Sin sus consejos para elegir las herramientas, diseñar la base de datos y plantear pruebas, este trabajo no tendría la misma fuerza. Su experiencia y compromiso fueron un modelo a seguir en cada paso.

Mención especial para el director de tesis, Juan Francisco Chafla Altamirano: sus preguntas y su guía semana a semana ayudaron a afinar tanto la parte técnica como la redacción. Con su supervisión al detalle, cada módulo de lo que está detrás del proyecto cada elemento de la interfaz alcanzaron la calidad que se buscaba.

A los compañeros de carrera: gracias por las madrugadas y las pruebas en grupo cuando la cabeza ya no daba más. Sus ideas y apoyo en los momentos de duda hicieron que este prototipo gane en creatividad y solidez, además de ser fieles acompañantes a lo largo de la carrera.

A mi familia, que estuvo allí más allá de lo académico. A mis papás, por creer en mí y por crearme un rincón tranquilo para estudiar. A mis hermanos, por recordarme que tomar un respiro y reírse también es parte del proceso. Sin su cariño, esto habría sido mucho más pesado.

A mis amigos de siempre, que entendieron cuando estaba a mil con la tesis y nunca faltaron con un empujón de ánimo. Sus mensajes de aliento y buenas vibras convirtieron la soledad de la pantalla en un viaje compartido.

RESUMEN

El prototipo que se desarrolló hace que revisar la seguridad de cualquier web sea tan fácil como lanzar un solo escaneo. En segundos detecta brechas, muestra qué datos podrían estar en riesgo y sugiere arreglos claros y al grano. Por debajo corre un *backend* en Python con Flask que invoca Nuclei, y en el navegador una UI sencilla en HTML/JavaScript muestra el cronómetro en tiempo real, con minutos y segundos. Toda la información sobre patrones de ataque y sus explicaciones vive en una base de datos PostgreSQL, así se pueden añadir nuevos chequeos sin cambiar una línea de código. Al juntar la rapidez de Nuclei con una interfaz interactiva y un almacenamiento bien organizado, este prototipo ahorra horas de trabajo a los desarrolladores y fortalece la protección de datos en forma que se actualice.

ABSTRACT

The developed prototype makes checking the security of any website as easy as running a single scan. It detects breaches in seconds, shows what data could be at risk, and suggests clear and concise fixes. Underneath, a Python backend with Flask runs that invokes Nuclei, and in the browser, a simple HTML/JavaScript UI displays the real-time timer, with minutes and seconds. All information about attack patterns and their explanations resides in a PostgreSQL database, so new checks can be added without changing a single line of code. By combining Nuclei's speed with an interactive interface and well-organized storage, this prototype saves developers hours of work and strengthens data protection in a timely manner.

ÍNDICE

Contenido

ÍNDICE DE FIGURAS, GRÁFICOS Y TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO DE TÉRMINOS

Clear-Site-Data: Encabezado HTTP que permite borrar al instante datos de caché, cookies y almacenamiento local cuando el usuario cierra sesión o cambia de estado, evitando que queden restos sensibles en el navegador.

XSS (*Cross-Site Scripting*): Vulnerabilidad que permite a un atacante inyectar y ejecutar scripts maliciosos en el navegador de la víctima, aprovechando la forma en que la aplicación web procesa contenido sin sanearlo correctamente. Suele usarse para robar cookies, secuestrar sesiones o redirigir al usuario a sitios maliciosos.

Inyección de SQL: Técnica de ataque en la que el atacante inserta comandos SQL maliciosos en campos de entrada de la aplicación (formularios, parámetros URL, etc.), logrando así leer, modificar o borrar datos de la base de datos sin autorización.

CSRF (*Cross-Site Request Forgery*): Vulnerabilidad que engaña al navegador de un usuario autenticado para que realice solicitudes no deseadas contra una aplicación web en la que ya ha iniciado sesión, permitiendo acciones como cambios de contraseña, transferencias de fondos o modificaciones de datos sin su consentimiento.

***Cross-Origin-Embedder-Policy (*COEP):** Política que obliga a que todos los recursos incrustados (imágenes, scripts, marcos) tengan permiso explícito para cargarse desde dominios distintos, fortaleciendo la defensa contra ataques cross-site.

***Cross-Origin-Resource-Policy* (CORP):** Controla qué orígenes externos pueden solicitar recursos de tu sitio, con él puedes restringir la carga de imágenes, fuentes u otros activos a orígenes de confianza.

***Permissions-Policy*:** Antigua "*Feature-Policy*" que te deja activar o desactivar APIs y capacidades del navegador (como geolocalización o cámara) en función de la fuente que hace la petición.

***Referrer-Policy*:** Define cuánta información de la URL de origen se envía como referencia al navegar a otro sitio, reduciendo la exposición de rutas internas o parámetros sensibles.

***Subresource Integrity* (SRI):** Mecanismo que añade un *hash* a recursos externos (scripts o estilos) para garantizar que no hayan sido manipulados en tránsito, el navegador verifica que coincida antes de ejecutar.

Strict-Transport-Security (HSTS): Encabezado que obliga al navegador a usar siempre HTTPS en tu dominio, evitando *downgrades* a conexiones inseguras y protegiendo frente a ataques *man-in-the-middle*.

***X-Frame-Options*:** Previene que tu sitio sea cargado en marcos (iframes) de otras páginas, bloqueando intentos de *clickjacking* al poder elegir “SAMEORIGIN” o “DENY”.

***Cookies HttpOnly*:** Atributo que impide el acceso a las cookies desde JavaScript, reduciendo el riesgo de robo de sesión vía ataques XSS.

TLS y *Cipher Suites*: Protocolo de cifrado (TLS 1.2/1.3) junto con un conjunto de algoritmos (*cipher suites*) que protegen la comunicación, deshabilitar versiones antiguas (TLS 1.0/1.1) y usar suites robustas es clave para evitar vulnerabilidades criptográficas.

robots.txt: Archivo que indica a los motores de búsqueda qué rutas no indexar, una mala configuración puede revelar directorios sensibles que no deberían estar accesibles.

security.txt: Archivo estándar que publica la forma de contactar al equipo de seguridad de un sitio, facilitando la comunicación de vulnerabilidades de forma responsable.

***Open-Redirect* genérico**: Vulnerabilidad que permite forzar redirecciones a dominios externos no controlados, usándose comúnmente para *phishing* o robo de credenciales.

CORS (misconfiguración de origen arbitrario): Configurar *Access-Control-Allow-Origin*, permite que cualquier web acceda a las respuestas del servidor, exponiendo datos sensibles a sitios no confiables.

***Content-Security-Policy* (CSP)**: Encabezado que define de forma explícita qué orígenes (scripts, estilos, imágenes, etc) pueden cargarse, bloqueando inyecciones de código malicioso.

X-Content-Type-Options: *nosniff* impide que el navegador adivine el tipo MIME de un

recurso, evitando interpretaciones erróneas que puedan provocar inyecciones.

Cross-Origin-Opener-Policy **(COOP)**: Encabezado que aísla el contexto de navegación entre pestañas, evitando que otras ventanas puedan acceder a la página y robar datos.

X-Permitted-Cross-Domain-Policies: Controla si se permiten o no archivos policy.xml desde orígenes externos, impidiendo que dominios ajenos definan políticas sobre el contenido propio.

CAPÍTULO I: INTRODUCCIÓN

Este primer capítulo pone en contexto los fundamentos que explican por qué vale la pena construir una herramienta independiente de escaneo de vulnerabilidades. Se aborda el origen de la idea y su planteamiento, los problemas que se pueden resolver y las metas a alcanzar con el proyecto.

1. Marco de referencia

1.1. Justificación

En un mundo digital cada vez más interconectado, la seguridad de las aplicaciones web es fundamental para garantizar la protección de datos sensibles y mantener la confianza de los usuarios. La presencia de vulnerabilidades como XSS, inyección de SQL y CSRF en aplicaciones web no solo pone en riesgo la seguridad de los datos, sino que también afecta la reputación y la estabilidad financiera de las empresas. Ante este escenario, es esencial tener herramientas eficaces que permitan a los desarrolladores identificar y remediar estas amenazas de manera proactiva.

El desarrollo de una herramienta automatizada para la evaluación de seguridad no solo ayudará a mejorar la seguridad de las aplicaciones web, sino que también ofrecerá una ventaja competitiva significativa al reducir el tiempo y el costo asociados con las pruebas de seguridad manuales.

1.2. Planteamiento del problema

A pesar de los avances en tecnologías de desarrollo web, muchas aplicaciones siguen siendo vulnerables a ataques que podrían evitarse con pruebas de seguridad adecuadas. Sin embargo, la realización de estas pruebas de forma manual no es solo costosa y propensa a errores, sino que también resulta insostenible en el ciclo de vida de desarrollo de software moderno, donde la rapidez y la eficiencia son clave. La falta de herramientas automatizadas accesibles y eficientes para realizar pruebas de seguridad complica aún más este contexto, dejando a muchas empresas expuestas a riesgos significativos. Según distintos estudios recientes, las aplicaciones web siguen presentando un número preocupante de vulnerabilidades:

Inyección SQL: El informe OWASP Top Ten 2021 muestra que el 63 % de las aplicaciones analizadas presentan al menos una vulnerabilidad de inyección SQL, una

falla crítica que permite a un atacante modificar o consultar bases de datos(OWASP Foundation, 2021).

Cross-Site Scripting (XSS): En el mismo reporte, se detectó que el 54 % de los sitios probados eran susceptibles a XSS, lo que posibilita la ejecución de código JavaScript malicioso en el navegador de la víctima (OWASP Foundation, 2021).

Número medio de vulnerabilidades por aplicación: Un estudio de Forrester Research en 2020 estima que, de media, cada aplicación web en producción contiene 72 vulnerabilidades activas, lo cual muestra lo difícil que es mantener un nivel de seguridad óptimo sin automatización (Forrester Research, 2020).

Patrones de ataque en la nube: Según el informe State of Cloud Security de Amazon Web Services, el 68 % de los ataques web dirigidos en entornos *cloud* emplean técnicas de XSS y un 32 % explotan debilidades de CSRF para secuestrar sesiones de usuario (Amazon Web Services, 2021).

Estas cifras demuestran la necesidad de integrar pruebas de seguridad automatizadas en el ciclo de vida del desarrollo de software, para detectar y corregir de forma temprana las vulnerabilidades más comunes antes de que puedan ser explotadas (OWASP Foundation, 2021).

De acuerdo con la información presentada se puede plantear la pregunta principal que es:

¿Cómo puede una herramienta automatizada mejorar la identificación y mitigación de vulnerabilidades comunes en aplicaciones web?

Y basado en la pregunta principal, también se puede considerar las siguientes preguntas secundarias:

¿Qué metodologías y tecnologías pueden integrarse más efectivamente para detectar vulnerabilidades de manera automática en diferentes tipos de aplicaciones web?

¿Cuál es el impacto de una herramienta de evaluación de seguridad automatizada en el proceso de desarrollo de software en términos de eficiencia y reducción de riesgos?

¿Cómo se puede diseñar una interfaz de usuario para que la herramienta sea accesible para desarrolladores con variados niveles de dominio en seguridad?

1.3. Objetivo General

Desarrollar un prototipo de herramienta automatizada que permita evaluar la seguridad

en aplicaciones web y proporcionar recomendaciones para la mitigación de vulnerabilidades comunes, mejorando la eficiencia del proceso de desarrollo seguro.

1.4. Objetivos Específicos

Estudiar y seleccionar las herramientas adecuadas para el ambiente de escaneo de vulnerabilidades.

Diseñar e implementar un sistema integrado de escaneo de vulnerabilidades, y la generación de recomendaciones de mitigación.

Validar el prototipo mediante escenarios de prueba para medir y analizar su efectividad y precisión.

1.5. Alcance

En este trabajo de titulación se llevará a cabo la creación de un prototipo de una herramienta para la evaluación de la seguridad en aplicaciones web. Este prototipo incluirá:

- Desarrollo de módulos de detección automática de vulnerabilidades web comunes, enfocándose en inyección de SQL, XSS y CSRF.
- Evaluación de las funcionalidades de la herramienta en entornos de prueba controlados, analizando su efectividad en la identificación de vulnerabilidades.
- Desarrollo de un reporte que proporcione al usuario información detallada sobre las vulnerabilidades encontradas y sugerencias para su mitigación.
- Diseño de la interfaz de usuario, orientado a facilitar el uso de la herramienta por desarrolladores de diferentes niveles de dominio.

Además, se llevará a cabo:

- Pruebas de rendimiento y precisión, que involucraran la evaluación de la capacidad de detección de vulnerabilidades en escenarios específicos, tomando en cuenta la rapidez y exactitud de los resultados.

Limitaciones: Este proyecto no incluirá la implementación de un escáner de seguridad comercial completo ni su integración con *pipelines* de CI/CD. El prototipo se limitará a la

detección de las vulnerabilidades más comunes y no abarca todas las posibles amenazas en aplicaciones web.

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

En este capítulo se explorará los conceptos y antecedentes que sustentan el proyecto, abordando el por qué la web sigue siendo un blanco fácil, qué prácticas existen para defenderla y cómo el panorama de la seguridad evoluciona día a día. La idea es sentar las bases teóricas antes de entrar al desarrollo práctico y sustentar el proyecto.

2.1. Antecedentes

Hoy en día, cualquier sitio web puede convertirse en un blanco fácil si no se revisa su configuración de seguridad. Muchos administradores descubren tarde que faltan cabeceras clave, que sus certificados TLS no están bien configurados o que exponen información sensible sin querer. Esos descuidos pueden resultar en pérdidas de datos, accesos no autorizados o malentendidos con sus usuarios.

La mayoría de las personas que mantienen páginas web no son expertos en hacking ético ni cuentan con equipos dedicados a pruebas de penetración. Por eso es común que, pese a las mejores intenciones, la seguridad quede en un segundo plano hasta que aparece un incidente. Además, chequear manualmente cada cabecera, cada respuesta del servidor o cada posible filtración de información consume mucho tiempo y conocimiento técnico.

Con esa realidad en mente, surge la necesidad de una herramienta accesible y rápida que ofrezca un detector de manera ágil: algo que te muestre en lenguaje claro qué está bien, qué falta y cómo arreglarlo, sin meterte en líneas de comando ni documentación compleja. Así, cualquier persona desde un desarrollador independiente hasta el administrador de un pequeño negocio puede validar su sitio en cuestión de segundos y tomar decisiones inmediatas para reforzar su seguridad.

2.2. Marco Teórico

2.2.1. **Ciberseguridad**

La ciberseguridad es el conjunto de prácticas y tecnologías destinadas a proteger la integridad, confidencialidad y disponibilidad de la información en entornos digitales, estableciendo salvaguardas en cada capa de la infraestructura tecnológica

(ISO/IEC 27000:2018). Esta área abarca desde el diseño de arquitecturas seguras hasta la implementación de políticas de acceso y autenticación, asegurando que solo los usuarios autorizados puedan interactuar con los activos críticos de la organización.

En un mundo cada vez más interconectado, las amenazas evolucionan con gran velocidad. Amenazas como *malware*, *ransomware*, ataques de denegación de servicio y vulneraciones en la cadena de suministro representan riesgos constantes que pueden detener sistemas y exponer datos sensibles (NIST SP 800-53 Rev. 5, 2020). La correcta identificación y clasificación de estos riesgos permite priorizar controles y optimizar recursos, evitando impactos en reputación y económicos que pueden derivar de una brecha de seguridad.

Para mitigar estos peligros se emplean controles técnicos como firewalls, sistemas de detección y prevención de intrusos (IDS e IPS), soluciones de gestión de información de seguridad y eventos (SIEM) y cifrado de datos en reposo y tránsito (NIST SP 800-53 Rev. 5, 2020). Cada medida es seleccionada según el nivel de exposición y criticidad de los activos, lo que permite construir obstáculos que dificultan el avance de un atacante y permiten detectar actividad inusual en tiempo real.

Además de las defensas a nivel técnico, los marcos de gobernanza y gestión de riesgos, como el Ciclo de Vida de Gestión de Riesgos de NIST (NIST SP 800-37, 2018) y las normas ISO 27001:2013, proporcionan un enfoque sistemático para evaluar, tratar y monitorear los riesgos a lo largo del tiempo (ISO/IEC 27000:2018). La adopción de estos estándares facilita el cumplimiento normativo y fomenta la mejora continua mediante auditorías internas y revisiones periódicas de políticas y procedimientos.

Las amenazas avanzadas persistentes (APT), las vulnerabilidades de dispositivos IoT y los ataques dirigidos a la cadena de suministro son retos emergentes que requieren un perfil de defensa proactivo y colaborativo (CNSSI 4009, 2010). La coordinación entre equipos de seguridad, proveedores y organismos reguladores fortalece la posición defensiva, compartiendo inteligencia de amenazas y estándares de respuesta para contrarrestar ataques sofisticados.

El factor humano sigue siendo uno de los eslabones más vulnerables y débiles, errores de configuración, contraseñas débiles y campañas de ingeniería social pueden debilitar las medidas más avanzadas (ISO/IEC 27000:2018). Programas de concientización,

simulacros de phishing y capacitación continua son esenciales para cerar una cultura de seguridad, donde cada empleado comprenda su papel en la protección de la información corporativa.

Finalmente, la capacidad de respuesta a incidentes y la automatización de procesos de contención, erradicación y recuperación permiten minimizar el tiempo de exposición y el impacto de los ataques. Herramientas de orquestación de seguridad, análisis forense y ejercicios de *tabletop* aseguran que la organización esté preparada para enfrentar incidentes con agilidad, adaptándose a nuevas tácticas adversarias y mejorando sus defensas de manera constante (NIST SP 800-53 Rev. 5, 2020). Además, para entender el ciclo de vida de la ciberseguridad se puede observar la figura 1.

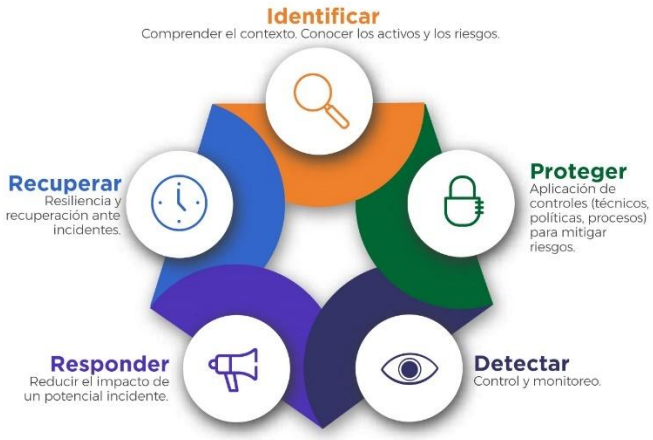

Figura 1 Ciclo de Vida de la Ciberseguridad (Agencia de Gobierno Electrónico y Sociedad de la Información y del Conocimiento, s.f.)

2.2.2. **Vulnerabilidades Web**

Las vulnerabilidades en aplicaciones web son posibles huecos que se encuentran en el diseño o en la implementación que permiten al atacante afectar la confidencialidad, integridad o disponibilidad de los datos (OWASP Foundation, 2021). Estos “huecos de seguridad” pueden presentarse en el código de servidor o en la configuración del entorno, y su explotación hace que hayan posibles pérdidas como robo de información, suplantación de identidad o toma de control del sistema (OWASP Foundation, 2021).

Uno de los fallos más importantes y difíciles de contener son las vulnerabilidades de inyección, como la inyección SQL, donde un atacante inserta código malicioso en una consulta que va a la la base de datos (Halfond et al., 2006). Si no se validan ni escapan

adecuadamente los datos de entrada, es posible ejecutar comandos no previstos, exfiltrar registros sensibles o incluso modificar la estructura de la base de datos (Halfond et al., 2006).

El Cross-Site Scripting (XSS) es otra vulnerabilidad frecuente, se produce cuando una aplicación refleja o almacena contenido suministrado por el usuario sin ningún proceso de snaitización, permitiendo la ejecución de scripts en el navegador de otras víctimas (Stuttard & Pinto, 2011). Con XSS, un atacante puede robar cookies de sesión, realizar acciones en nombre del *usuario o redirigirlo a sitios maliciosos (Stuttard & Pinto, 2011).*

El Cross-Site Request Forgery (CSRF) engaña al navegador de un usuario autenticado para que envíe solicitudes no deseadas a la aplicación, aprovechando la confianza que esta tiene en la sesión activa (Barth et al., 2008). Sin contramedidas como tokens anti--CSRF únicos por sesión, es sencillo que un atacante provoque cambios en los datos o en la configuración de la cuenta de la víctima (Barth et al., 2008).

Las implementaciones de autenticación y gestión de sesiones defectuosas también son críticas. Sesiones predecibles, cookies sin atributos de seguridad o políticas de bloqueo insuficientes facilitan el secuestro de cuentas y el escalado de privilegios (National Institute of Standards and Technology, 2017). Un control bastante fuerte de expiración de tokens y autenticación multifactor son asuntos recomendables para mitigar estos riesgos (National Institute of Standards and Technology, 2017).

La mala configuración de servidores, *frameworks* o componentes de terceros es un concepto cotidiano y habitual, puertos que están expuestos, credenciales por defecto o directivas de seguridad que no existen, pueden dejar al descubierto información sensible como rutas de archivos, configuraciones internas o datos personales (Mitre Corporation, 2020). Estas fallas permiten a un atacante descubrir de una manera inicial la infraestructura y planificar ataques más directos (Mitre Corporation, 2020).

Para protegerse de estas vulnerabilidades, se recomienda tomar estándares de verificación como el OWASP ASVS, realizar pruebas de seguridad (*pentesting*, análisis dinámico y estático) y desplegar soluciones de defensa como WAFs (National Institute of Standards and Technology, 2008, OWASP Foundation, 2019). La combinación de codificación segura, auditorías periódicas y controles de configuración hace que reduzca la superficie de ataque y fortalece la seguridad de la aplicación (National Institute of

Standards and Technology, 2008, OWASP Foundation, 2019), para demostrar las vulnerabilidades web más comunes según OWASP se puede observar la figura 2.

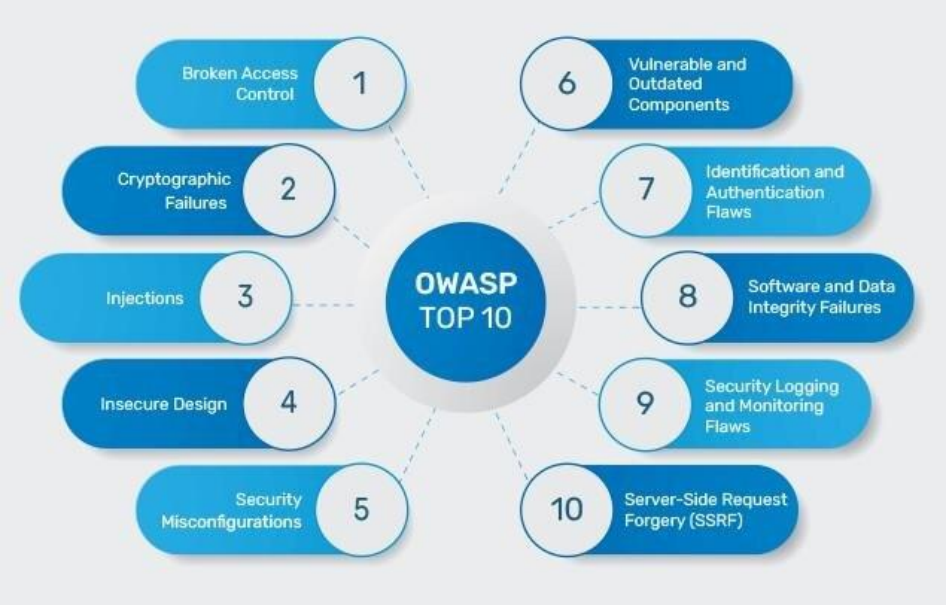

Figura 2 Top 10 vulnerabilidades identificadas en OWASP 2021 (OWASP Foundation, 2021)

2.2.3. **Seguridad en la Web**

La seguridad en la web es el conjunto de políticas, controles y prácticas diseñadas para salvaguardar aplicaciones, servicios y datos que se ofrecen a través de navegadores, mitigando riesgos como inyección de código, secuestro de sesiones y exposición de información sensible (OWASP, 2021). Pero hoy en día no basta con solucionar cuando algo sale mal: las prácticas *DevSecOps* (es un enfoque que integra la seguridad desde el inicio del ciclo de desarrollo de software, automatizando prácticas seguras del equipo de desarrollo y operaciones en cada etapa del proceso DevOps), nos empujan a mover la seguridad “hacia la izquierda”, integrando análisis SAST (análisis de código estático), escaneo de dependencias y *tests* de contenedores dentro del *pipeline* de CI/CD (Integración Continua y Entrega/Despliegue Continuo, es una práctica que automatiza la construcción, prueba y entrega de software, asegurando entregas rápidas, seguras y constantes en entornos de desarrollo y producción), para encontrar vulnerabilidades antes de que nadie lo note.

Además, cada vez es más común usar IaC *scanning* (análisis de infraestructura como código) para detectar malas configuraciones en *Terraform* o *CloudFormation*, y desplegar alertas automáticas y *dashboards* de cumplimiento que avisen al equipo en tiempo real. Todo esto, sumado a técnicas tradicionales como autenticación fuerte, cifrado TLS,

validación de entradas y monitoreo continuo, ayuda a mantener la integridad, confidencialidad y disponibilidad de nuestros recursos web (NIST SP 800-44 Rev. 2, 2007).

Asimismo, dentro de estas prácticas automatizadas destacan los enfoques de análisis DAST (Dynamic Application Security Testing) y SAST (Static Application Security Testing). SAST analiza el código fuente, bytecode o binarios en fases tempranas del desarrollo sin ejecutar la aplicación, permitiendo detectar fallos como inyecciones o errores de lógica antes del despliegue. Por otro lado, DAST evalúa la seguridad de la aplicación en ejecución, simulando ataques externos para identificar vulnerabilidades en tiempo real, como fallos de autenticación o exposición de datos. La combinación de ambos enfoques proporciona una cobertura robusta y complementaria que refuerza la postura de seguridad en todo el ciclo de vida del software.

CAPÍTULO III: ESTUDIO Y SELECCIÓN DE HERRAMIENTAS

En este capítulo veremos todo lo necesario para poner en marcha el entorno de escaneo de seguridad web. Primero se abordará la selección de las piezas clave: desde el motor de detección basado en plantillas hasta el *framework* que levantará el servidor, la base de datos donde se guardarán los patrones y explicaciones, y las librerías que harán de puente entre ellos. La idea es entender por qué elegimos cada herramienta, qué aporta al flujo de trabajo y cómo encajan entre sí para formar una solución unida.

3.1. Selección de las herramientas a utilizar

Para el desarrollo de este prototipo de evaluación de seguridad web se consideraron los siguientes criterios: flexibilidad, madurez de la herramienta que se pueda conectar, licencia de código abierto, facilidad de integración con la arquitectura del prototipo y rendimiento. Para diseñar una solución de evaluación de seguridad web, se presentan tres opciones para analizar y comparar alternativas de escáneres de vulnerabilidades. A continuación, se presenta las tres herramientas candidatas principales y los criterios empleados:

- Flexibilidad: ¿Permite personalizar plantillas o reglas?
- Madurez: ¿Existe documentación, plantillas y soporte activo?
- Rendimiento: ¿Es capaz de procesar gran cantidad de objetivos de forma rápida?
- Integración: ¿Se adapta bien al *stack* del prototipo (Flask + JS)?
- Licencia: ¿Es de código abierto y permisiva para su uso?

Tras comparar varios escáneres, se eligió comparar entre, Nuclei, Nikto y OWASP ZAP. Cada uno cumple un objetivo diferente y distinto que podría usarse en el prototipo final:

3.1.1. Nuclei

Es un escáner que funciona con plantillas en YAML para armar las reglas de detección, así que resulta ultra modular y muy fácil de actualizar. Como tiene una biblioteca enorme de "*templates*" que mantiene la comunidad, se pueden agregar pruebas nuevas rápidamente y ajustarlas a lo que se necesite. Además, su motor está configurado para

escaneos a gran escala, permitiendo revisar cientos de objetivos en minutos sin perder ni un poco de exactitud.

3.1.2. Nikto

Posee un escáner que detecta la configuración insegura y las rutas típicas de servidores web. Se encarga de chequear cabeceras HTTP mal puestas, detectar software anticuado o antiguo y descubrir archivos que sobran, con más de 6 700 chequeos listos para usar. Y lo mejor es que arma reportes en HTML o CSV, así compartir los hallazgos con el equipo de desarrollo o auditoría es mucho más sencillo.

3.1.3. OWASP ZAP

Tiene la funcionalidad de un *proxy* de interceptación que combina pruebas manuales y automáticas, ideal para auditorías interactivas. Su interfaz gráfica es intuitiva, pero también dispone de una API REST que permite integrar escaneos pasivos y activos dentro de flujos *DevSecOps*. De este modo, ZAP complementa perfectamente a las herramientas basadas en plantillas, añadiendo un análisis más profundo del flujo de la aplicación y simulando ataques reales en tiempo de ejecución.

3.2. Comparación de Herramientas

Para mejorar la comparación y entender cuál se debe elegir en base del prototipo se propone la siguiente tabla comparativa entre herramientas:

Herramienta	Tipo de escaneo	Plantillas / Rules	Rendimiento	Integración	Comunidad (GitHub)
Nuclei	Basado en plantillas	YAML (modular)	Muy alto	CLI / API / Python	★★★★☆ (4.8k estrellas)
Nikto	Chequeos predefinidos	Internas (perl)	Medio	CLI / Perl	★★★☆☆ (1.9k estrellas)
ZAP	Proxy activo/pasivo	XML / Políticas	Variable	GUI / REST API / Java	★★★★☆ (5.6k estrellas)

Tabla 1 Tabla Comparativa de Escáneres

Tras comparar distintas opciones, finalmente se decantó por Nuclei como motor principal de escaneo de vulnerabilidades. Nuclei trabaja con plantillas YAML que facilitan la creación y reutilización de reglas de detección de manera muy modular, cuenta con una comunidad activa que mantiene una amplia librería de "*templates*" y destaca por su alto rendimiento cuando necesitamos lanzar escaneos masivos sobre múltiples objetivos.

3.3. Selección y análisis de herramientas complementarias

3.3.1. **Arquitectura del Prototipo**

La arquitectura del prototipo combina una interfaz liviana en el navegador con un servidor *RESTful* que orquesta el escaneo y el post-procesamiento. Cuando el usuario pulsa “Iniciar escaneo”, el cliente JavaScript (*Fetch* API) envía la URL al *endpoint* de Flask. Allí, Flask ejecuta Nuclei con plantillas predefinidas para detectar vulnerabilidades, guarda la salida en un archivo temporal y luego consulta los mapeos cargados en memoria (patrones, explicaciones, soluciones y recomendaciones) que se precargaron al arrancar el servicio.

Una vez Nuclei termina, el servidor enriquece cada hallazgo con la información de la base de datos y devuelve un objeto JSON al navegador. El cliente recibe ese JSON y actualiza dinámicamente la interfaz: primero muestra un cronómetro, luego despliega secciones con vulnerabilidades, datos comprometidos, posibles soluciones y recomendaciones. Este flujo optimiza la latencia al reducir las consultas a la base de datos durante cada escaneo y mantiene la experiencia de usuario ágil y fluida. Para entender de forma visual esto, ver la figura 3.

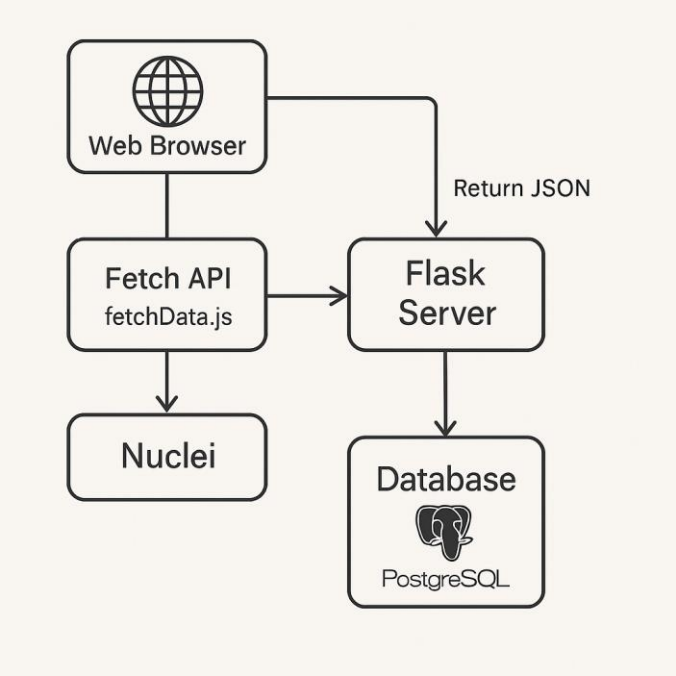

Figura 3 Diagrama de arquitectura del prototipo de escaneo de seguridad web (ChatGPT, 2025).

3.3.2. **Flask (Python)**

Para coordinar el conjunto del escaneo desde el servidor y devolver resultados en

JSON se optó por Flask, un *microframework* de Python súper liviano y fácil de estirar con *plugins*. En un par de líneas se levanta la API que escucha las peticiones, y con la extensión *Flask-CORS* se evita cualquier problema y confusión de orígenes cruzados al recibir llamadas desde el JavaScript del cliente.

3.3.3. **PostgreSQL**

En la parte de almacenamiento se eligió PostgreSQL ya que, soporta transacciones *ACID* de excelente manera, maneja de serie datos JSON y arreglos, y rinde parejo con muchos registros. Ahí se guarda patrones de vulnerabilidad, explicaciones, datos expuestos, posibles soluciones y recomendaciones. Para conectar Python con la base de datos se usó *Psycopg2*, que gestiona consultas preparadas y tipos complejos sin mucha complejidad.

3.3.4. **JavaScript (*Fetch* API)**

El cliente en el navegador usa la *Fetch* API para todo: al hacer clic al botón de “Iniciar escaneo” se envía la URL al servidor, pone en marcha un cronómetro en tiempo real y va mostrando los resultados en vivo. Al no necesitar herramientas de empaquetado extra, el código queda sencillo y ligero, y hasta puede convertirse en extensión de navegador sin mayor complejidad.

3.3.5. **HTML5 & CSS3 (Tailwind/CSS puro)**

La estructura de la interfaz se arma con HTML5 y CSS3, mientras que *Tailwind* acelera el estilizado y garantiza que todo se vea bien en navegadores actuales (que sea responsivo y con un diseño atractivo para el usuario). Así se evita complicarse con compilaciones: el HTML marca el contenido, CSS puro o con clases de *Tailwind* le da forma, y JavaScript se encarga de la lógica dinámica sin necesidad de enredarse el uno al otro.

3.3.6. **Herramientas de desarrollo**

Finalmente, para el flujo de desarrollo se recurrió a Git como sistema de control de versiones, Visual Studio Code como editor (con extensiones para Python, SQL y Live Server) y herramientas de pruebas manuales como *Postman* o *Insomnia* para validar la API antes de integrarla con el *frontend*.

CAPÍTULO IV: DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE ESCANEO DE VULNERABILIDADES

En este capítulo se describe en detalle cómo se diseñó e implementó el sistema integrado de escaneo de vulnerabilidades. El proceso incluyó el modelado de la base de datos para almacenar patrones y sus explicaciones, el diseño de la arquitectura de la aplicación y la interfaz de usuario, la creación del motor de detección automática (SQLi, XSS, CSRF y otras plantillas) y la implementación del módulo de generación de reportes con recomendaciones de mitigación.

Como se explicó en el punto 3.1.1, el prototipo necesita de varios componentes en su arquitectura a continuación durante el desarrollo de este capítulo se explicarán como se configuraron y cómo funcionan estos componentes para orquestar de una buena forma el prototipo, con el fin de que cada uno cumpla la función antes mencionada.

Además, para el manejo de código el cual no está explícitamente en este capítulo, se puede referenciar el Anexo 2 que incluye el repositorio donde están las carpetas de ambos proyectos (Prototipo con UI en interfaz web y Prototipo con UI de Extensión para Navegadores).

4.1. Modelado de la base de datos

El prototipo emplea un modelo lógico que agrupa cada patrón de vulnerabilidad con su explicación, datos comprometidos, soluciones y recomendaciones. Este esquema se traduce en un diseño físico en PostgreSQL donde cada tabla refleja uno de estos conceptos y aprovecha columnas de tipo JSONB para almacenar listas variables de soluciones o recomendaciones. Así, la incorporación de nuevas reglas no requiere alterar el esquema, sino únicamente insertar nuevos registros.

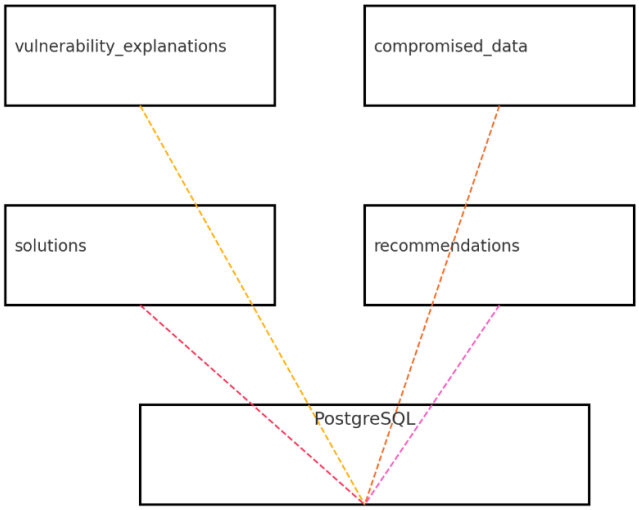

Figura 4 Modelo físico de la base de datos en PostgreSQL (Paolo Ortiz, 2025).

Además, para entender como el usuario, que utilizará el prototipo, interactuará con la base de datos, se puede ver el diagrama de secuencia del sistema en la figura 5.

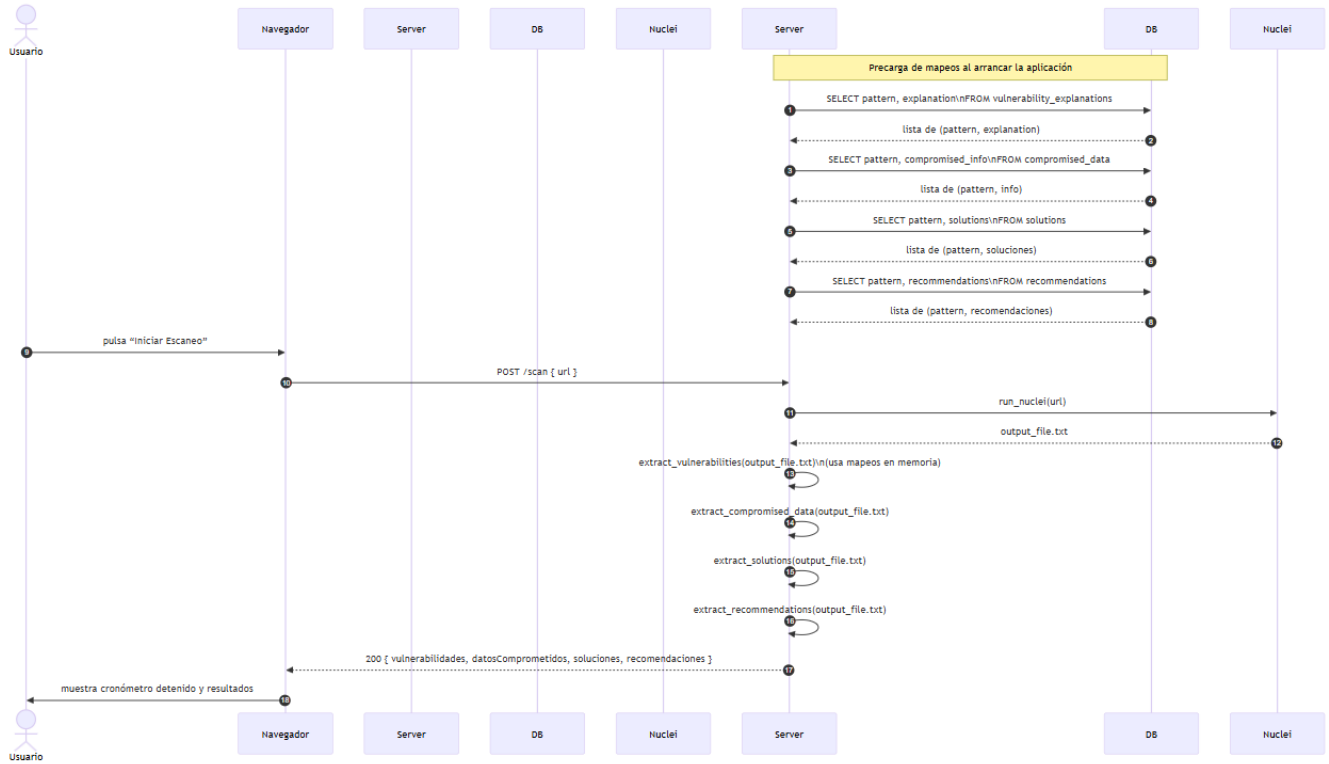

Figura 5 Diagrama de secuencia de la interacción entre el usuario, el servidor y la base de datos (Paolo Ortiz, 2025).

4.2. Diseño de la arquitectura e interfaz de usuario

La capa de servicio se basa en Flask, que expone un único *endpoint* /*scan* recogiendo la URL que el prototipo va a escanear y haciendo el escaneo con Nuclei. La interfaz, construida con *HTML5* y *Fetch API,* mantiene una apariencia bonita y fácil de entender: al pulsar “Iniciar escaneo” se muestra un cronómetro y, al completarse, cuatro secciones se muestran en formato cuadrícula, una tras otra para presentar vulnerabilidades, datos comprometidos, soluciones y recomendaciones. Esta ejecución hace que no existan dependencias y acelera el tiempo de respuesta al usuario.

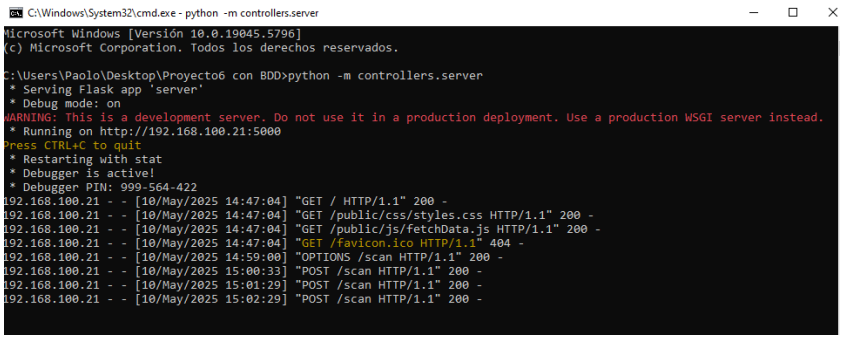

Figura 6 Ejecución del Servidor FLASK encargado de la visualización de la UI del prototipo (Paolo Ortiz, 2025).

El servidor Flask se logra ejecutar desde la línea de comandos (esto puede hacerse en Windows, Linux, etc.). Además, nos puede avisar de posibles notificaciones de la ejecución, tales como: inicio del escaneo, publicación (POST) de los resultados y también, nos avisa de las conexiones desde otros dispositivos al servidor. Esto se explica en la figura 6.

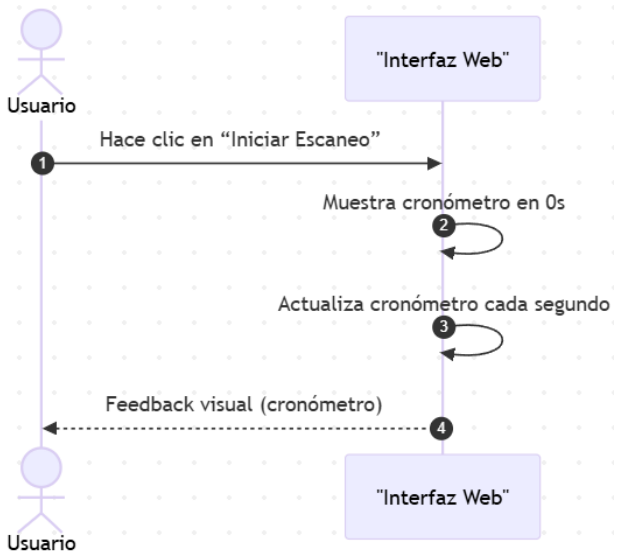

Figura 7 Diagrama de Secuencia de la Interacción Usuario–UI de la extensión Web (Paolo Ortiz, 2025).

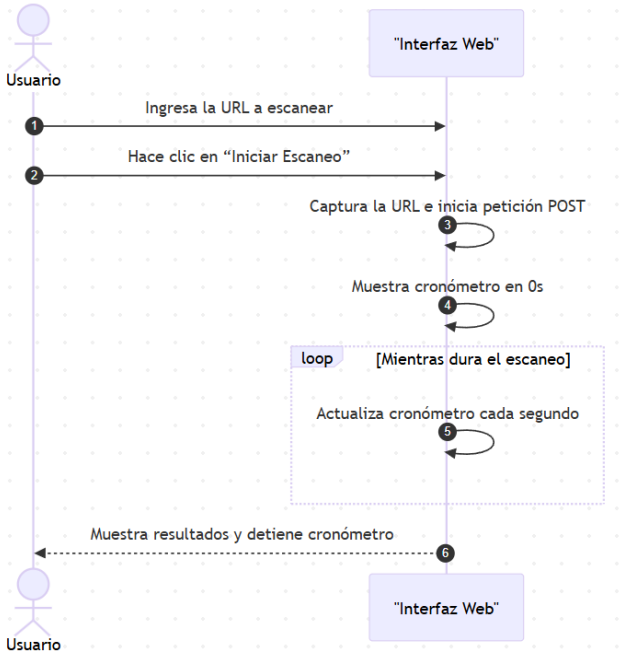

Figura 8 Diagrama de Secuencia de la Interacción Usuario–UI de la Interfaz Web (Paolo Ortiz, 2025).

A partir de esto, se tiene el prototipo de interfaz de usuario del prototipo, donde se puede observar que en realidad se mantiene un principio que se ha aprendido a lo largo de la carrera el cuál es “hazlo simple, mantenlo simple”, para que el usuario pueda saber que hacer sin perderse y le parezca muy “simple”, para entender cómo funciona el flujo entre el usuario y la interfaz se puede observar la figura 7, mientras que, para entender la UI se puede observar la figura 8.

Figura 9 UI del Prototipo (Interfaz Web) (Paolo Ortiz, 2025).

Figura 10 UI del Prototipo (Extensión de Navegador) (Paolo Ortiz, 2025).

Al mencionar una herramienta web nos podemos referir también, a una extensión para navegadores, por lo que se pensó en diseñar también, esta interfaz, la cual también mantiene un diseño simple, un diseño que sea bastante fácil de tender y que, al ser una extensión, analice el link actual de la página donde está el usuario para poder escanearla, como se puede observar en la figura 9.

4.3. Detección automática de amenazas

El motor de escaneo integra Nuclei ejecutándose en segundo plano, sus plantillas

YAML determinan qué vulnerabilidades buscar. Al iniciar el servidor, todos los mapeos se cargan en memoria, reduciendo la latencia de consulta durante el procesamiento. Cuando Nuclei finaliza el análisis, su salida se combina con esos mapeos para generar un listado ordenado por severidad, enriquecido con explicaciones que sean entendibles para el usuario y permitan conocer de forma en realidad bastante fácil como es que pueden llegar a mitigar esto.

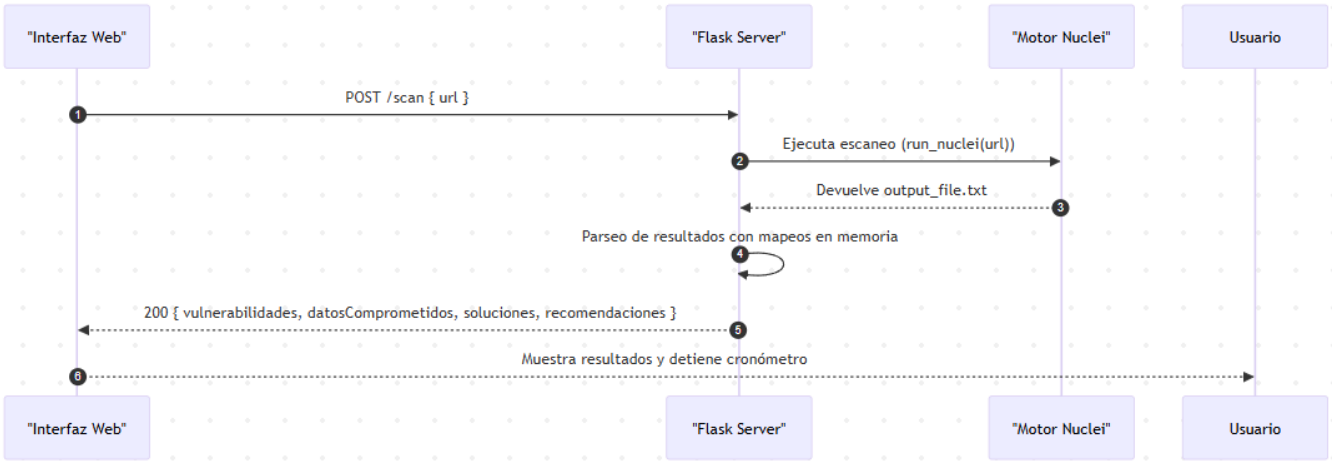

Figura 11 Diagrama de Secuencia del Flujo UI–Nuclei–UI (Paolo Ortiz, 2025).

A partir de la figura 10, se puede entender que la UI se comunica con Nuclei a través del archivo output que generó Nuclei, para poder tomar las decisiones en el siguiente paso.

4.4. Generación de recomendaciones de mitigación

Una vez procesados los resultados, el prototipo arma un objeto JSON que agrupa cada hallazgo junto a su recomendación de corrección que se imprime en la UI tanto web como la de extensión. Además, el usuario puede visualizar lo resultados en la página web con estructura: título de vulnerabilidad, explicación breve y sugerencias de mitigación. Este mecanismo facilita compartir y archivar los resultados fuera de la aplicación.

Figura 12 Generación de recomendaciones automáticas a partir de mapeos en UI Web (Paolo Ortiz, 2025).

En la figura 11 (UI Web), 12 y 13 (UI Extensión Web) se puede observar cómo se generan las recomendaciones a partir de los resultados, combinados con un mapeo previo de soluciones, recomendaciones, explicaciones, en la base de datos y que a partir de esto se muestran en la UI, para que el usuario al igual que en todos los pasos, tenga un acceso fácil a esta información sin la necesidad de hacer "*clicks*" extras.

Figura 13 Generación de recomendaciones automáticas a partir de mapeos en UI Extensión Web (Paolo Ortiz, 2025).

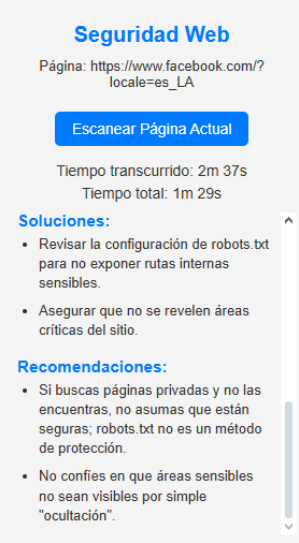

Figura 14 Generación de recomendaciones automáticas a partir de mapeos en UI Extensión Web II (Paolo Ortiz, 2025).

4.5. Precarga de mapeos y optimización del flujo de escaneo

Para garantizar que el análisis sea fluido incluso bajo carga, todos los mapeos de base de datos se precargan al arrancar Flask. De esta forma, las funciones de extracción de explicaciones, datos comprometidos, soluciones y recomendaciones operan sobre estructuras en memoria, evitando viajes continuos a la base de datos y asegurando tiempos de respuesta consistentes en cada solicitud de escaneo.

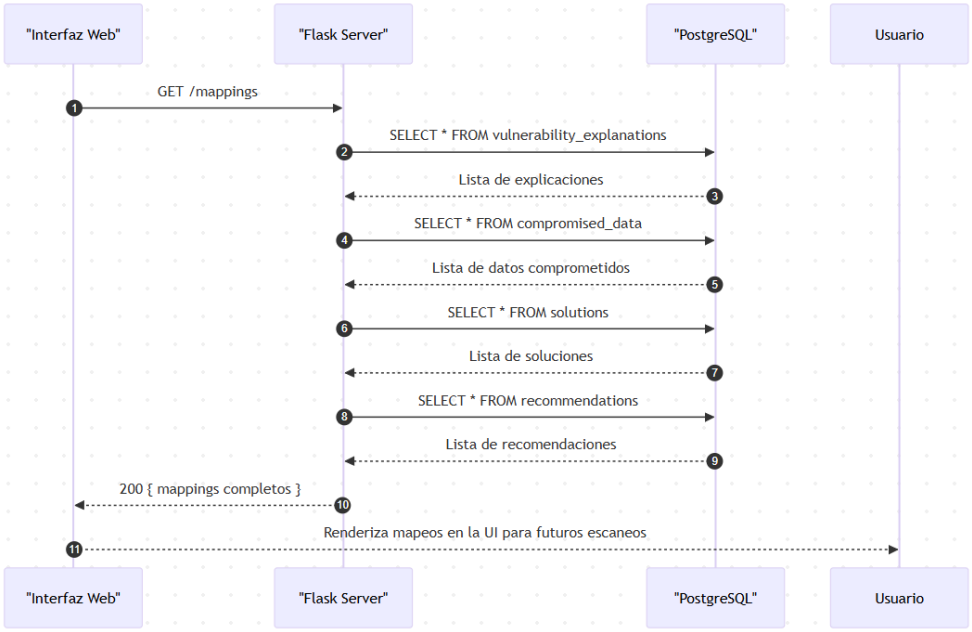

Figura 15 Diagrama de Secuencia del Flujo UI–Base de Datos–UI (Paolo Ortiz, 2025).

Se puede observar en la figura 14, cómo funciona el flujo de comunicación entre la interfaz de usuario y la base de datos, para que a través de los resultados de escaneo hechos por Nuclei, se pueda mapear las vulnerabilidades encontradas con las explicaciones guardadas en la base de datos del prototipo para hacer todas las explicaciones necesarias.

Para entender de manera resumida la función de cada uno de los elementos sobre el prototipo, ver la tabla 2.

Componente	Aporte del proyecto
Modelado de la base de datos	Permite almacenar de forma flexible patrones, explicaciones, datos comprometidos, soluciones y recomendaciones sin necesidad de modificar el esquema al agregar nuevas reglas.
Diseño de la arquitectura e interfaz de usuario	Ofrece un flujo ágil: un único *end-point* REST para el escaneo y una UI minimalista que muestra cronómetro y resultados en cuatro secciones ordenadas.
Detección automática de amenazas	Integra Nuclei con plantillas YAML para identificar SQLi, XSS, CSRF y cualquier otra plantilla soportada, enriqueciendo cada hallazgo con su explicación correspondiente.

Generación de reportes con recomendaciones	Agrupa los resultados en un JSON estructurado y genera informes descargables en texto plano, facilitando la revisión y el seguimiento de las acciones de mitigación.
Precarga de mapeos y optimización del flujo	Carga todos los mapeos en memoria al arrancar el servidor, reduciendo latencia y garantizando tiempos de respuesta consistentes bajo cualquier carga de trabajo.

Tabla 2 Aportes de los componentes del sistema de escaneo de vulnerabilidades

CAPÍTULO V: PRUEBAS Y ANÁLISIS

Este capítulo incluirá tres escenarios de prueba para demostrar la eficacia del primer prototipo de la herramienta, donde se podrá visualizar a grandes rasgos, cómo funciona la misma, el manejo de tiempos y de resultados, además permitirá discernir entre vulnerabilidades comunes en sitios en los que se navega cotidianamente.

Se han planteado cuatro escenarios para realizar pruebas (tres sitios web creados con vulnerabilidades para su reconocimiento y la página de la universidad), en donde se va a encontrar distinta información que nos permita escanear y encontrar vulnerabilidades, y estos sitios web creados están referenciados en el Anexo 3.

5.1. Caso Uno – Página Web sin Vulnerabilidades:

Esta página es súper simple: puro HTML y nada de JavaScript. Es como un lienzo en blanco para ver qué pasa cuando no hay scripts de por medio. Sirve para medir un “caso base” y para asegurar que, si todo está bien configurado, no aparezca ninguna alerta rara ni vulnerabilidad tonta que haga que el prototipo no funcione.

Al probar con esta página, solo se espera ver el contenido estático y el cronómetro corrido sin encontrar nada extraño. Es como la referencia limpia antes de meterse con cosas más fuertes o vulnerabilidades que involucren el uso del prototipo.

Figura 16 Página HTML del primer caso (Paolo Ortiz, 2025).

Como se mencionó antes, el propósito de esta página es descartar la idea de que el prototipo lance falsos positivos, entonces sobre la página web HTML, el prototipo procede

a realizar el escaneo con el prototipo para así poder evaluar los resultados, en la figura 17 se puede observar que el usuario únicamente ingresa el link de la página que desea escanear y procede a dar clic en el botón de iniciar escaneo, lo cual empezará a realizar pruebas sobre la página deseada, para así mostrar información relevante, además aporta *feedback* visual en cuanto al tiempo que se demora en hacer el escaneo.

Figura 17 Despliegue de Resultados Generales en la UI Web (Paolo Ortiz, 2025).

Analizando los resultados presentes en la figura 17 se puede comprobar que en realidad solo hay vulnerabilidades de severidad INFO, es decir informativas, las cuales son en realidad mínimas y se debe a que la página al ser bastante básica, le faltan ciertas configuraciones que le ayuden a ser lo más segura posible, además de que el escaneo se realizó en un minuto y diecisiete segundos lo que quiere decir que fue bastante ágil y

rápido, ya que no habían michos detalles que escanear porque como se dice arriba, fue una página con simple texto estático sin ninguna otra configuración.

5.2. Caso Dos – Página con Formulario Vulnerable a SQL Injection:

Aquí se puede observar la función del prototipo ya que es un caso real, esta página toma parámetros de la URL y los refleja tal cual, en el formulario, sin filtrar nada. Eso la convierte en la prueba perfecta para inyectar etiquetas de script y ataques SQL con solo cambiar el parámetro en la barra de direcciones o URL.

Con esta página se puede ver en tiempo real cómo un simple formulario explota la página o cómo una comilla extra rompe la consulta de la base de datos. Es ideal para practicar y ver de cerca dos vulnerabilidades clásicas como son: XSS y SQL Injection.

Figura 18 Página HTML del segundo caso (Paolo Ortiz, 2025).

A partir de ahora, se realizarán páginas con mucha mayor configuración en cuanto a elementos para poder visualizar que son esas cosas que tienen los sitios web que lo podían hacer vulnerables, en este caso como se introdujo arriba se puede decir que es una página de la que se puede extraer datos ya que no existe una configuración de encriptación de los datos al momento de llenar un formulario, por lo que en las figuras 19-24 se puede ver ahora el prototipo de la herramienta de tipo extensión web, el cual tiene incluso un mejor funcionamiento para el usuario, permitiendo realizar el escaneo sobre la página en la cual está el usuario, haciendo que este, no tenga que ingresar el link de la página a escanear.

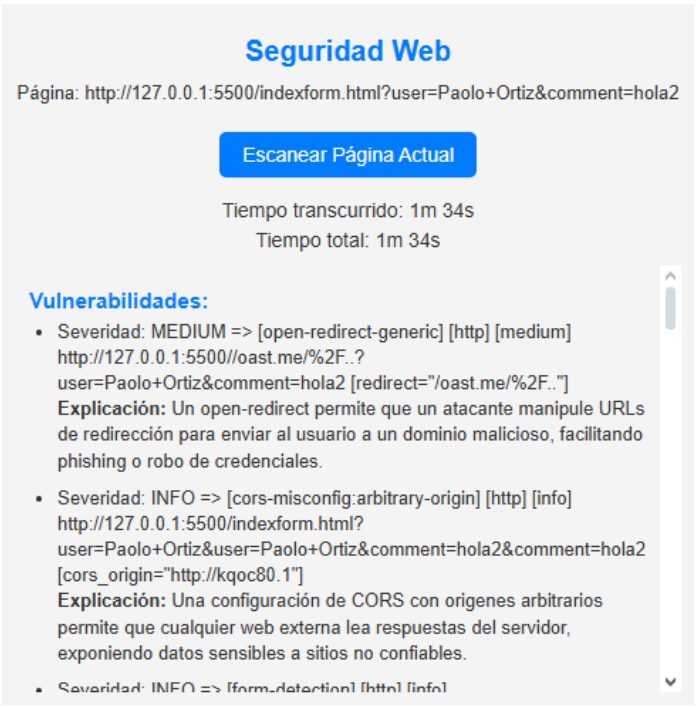

Figura 19 Despliegue de Resultados Generales de la UI Extensión Web Parte Vulnerabilidades (Paolo Ortiz, 2025).

En la figura 19, se observa el primer despliegue de la interfaz de extensión web en el navegador OperaGX, y se destaca las vulnerabilidades encontradas en la segunda página de pruebas, mostrando que hay una vulnerabilidad de severidad MEDIUM de tipo redirección genérica, y esto puede derivar en robo de información o credenciales.

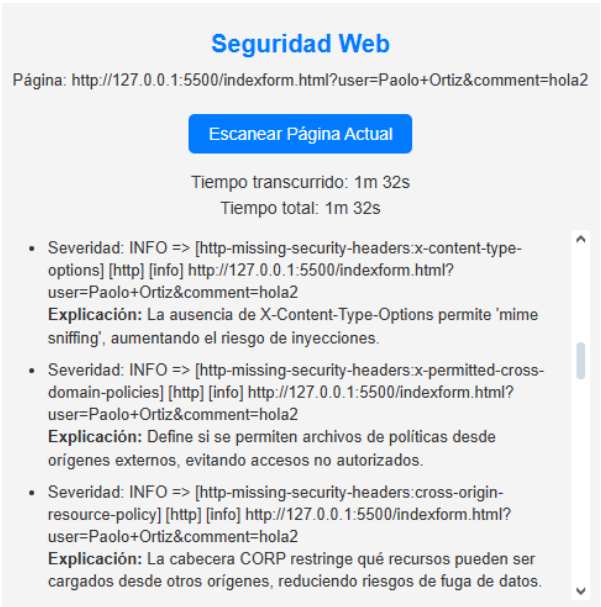

Figura 20 Despliegue de Resultados Generales de la UI Extensión Web Parte Vulnerabilidades II (Paolo Ortiz, 2025).

En la figura 20, se observa el segundo despliegue de la interfaz de extensión web en el navegador OperaGX, y se destaca las vulnerabilidades encontradas en la segunda página de pruebas, donde se presenta vulnerabilidades de tipo INFO por lo que en realidad estás solo cumplen la función de informar posibles mejoras o información del sitio.

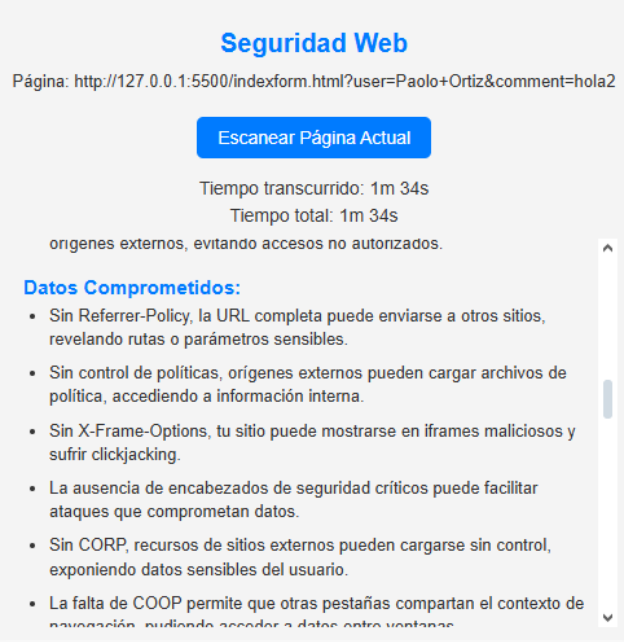

Figura 21 Despliegue de Resultados Generales de la UI Extensión Web Parte Datos Comprometidos (Paolo Ortiz, 2025).

En la figura 21, se observa el despliegue de la interfaz de extensión web en el navegador OperaGX, y se destaca los datos comprometidos a partir de las vulnerabilidades encontradas, estas se realizan a través del mapeo con la base de datos, y se muestra que principalmente el dato con mayor riesgo es información personal del usuario.

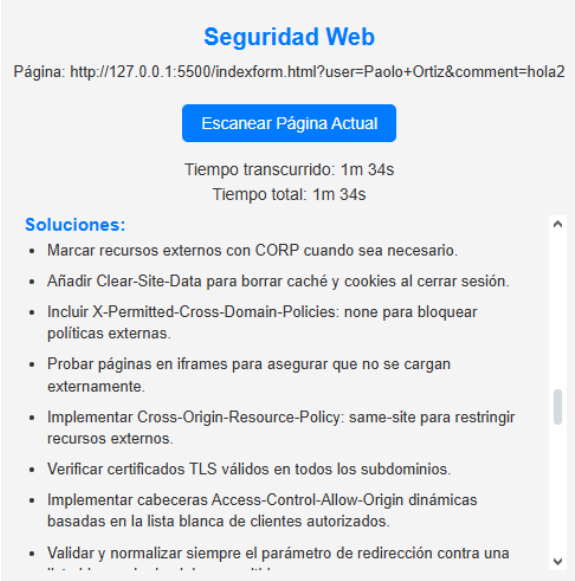

Figura 22 Despliegue de Resultados Generales de la UI Extensión Web Parte Posibles Soluciones (Paolo Ortiz, 2025).

Después de realizar el escaneo, tal y como se menciona en EveryTI, siempre debemos organizar las vulnerabilidades desde la más crítica “HIGH” o “MEDIUM” a las más bajas “LOW” o “INFO” esto en cuanto a severidad de estas, además también se menciona que debemos desarrollar el plan de mitigación (EveryTI, s.f.), el cuál en este caso serían las posibles soluciones y las recomendaciones al usuario, entonces de cierta forma esto apoya al análisis de las vulnerabilidades y que sea un proceso estandarizado, que es lo que se busca con el prototipo.

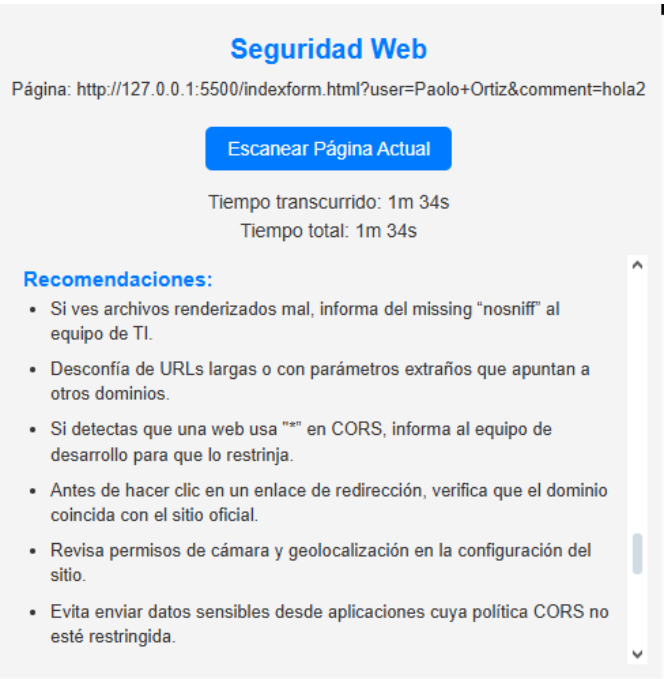

Figura 23 Despliegue de Resultados Generales de la UI Extensión Web Parte Recomendaciones al Usuario (Paolo Ortiz, 2025).

Tras obtener todos los resultados, se analiza que, en esta página creada, se puede realizar un ataque de *Sql Injection* ya que, el formulario al no borrarse después de dar clic puede hacer que en el enlace se cargue información como contraseñas, usuarios, etc. Además de que no estandariza comandos SQL por lo que podría resultar en borrado de base de datos, búsquedas no autorizadas o incluso filtración de datos privados y posiblemente confidenciales. Además, al ser una página con un poco más de configuraciones se demoró un poco más el escaneo llegando a un minuto y treinta y cuatro segundos.

5.3. Caso Tres – Página Web con Token Repetido:

Esta página simula un mini panel de usuario donde cada vez que se envía un formulario se manda un parámetro que, en vez de ser único, es siempre el mismo o sigue un patrón bastante fácil de reconocer. Así se puede probar fácilmente ataques CSRF y ver cómo, con solo copiar ese parámetro enviado, se puede forzar acciones no deseadas hacia el usuario en estas páginas web.

Con esta página se puede aprender de forma práctica por qué un token fijo es un peligro: cualquier atacante puede adivinarlo y hacer “clics” maliciosos en nombre de otro

usuario, que en este caso sería el usuario que utiliza la página web. Es el ejemplo perfecto de por qué los tokens tienen que ser necesarios en un sitio web.

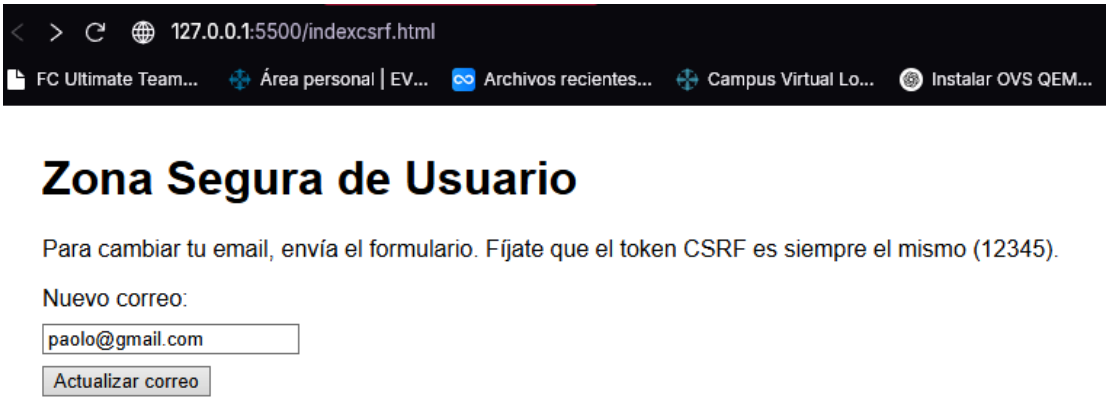

Figura 24 Página HTML del tercer caso (Paolo Ortiz, 2025).

Se puede observar una configuración de la página bastante simple en la figura 25, incluyendo únicamente un formulario de actualización sin pedir un token diferente ya que no se borra el contenido inicial, lo cual puede llevar a posibles ataques de filtración de datos del usuario.

Figura 25 Despliegue de Resultados Generales en la UI Web (Paolo Ortiz, 2025).

A partir de estos resultados se puede detectar que no se presentan muchas debilidades críticas en cuanto a severidad, en realidad ninguna y esto se debe a que el prototipo considera que en realidad el token no podría ser mal utilizado, ya que se habla de un “correo” información qué se podría encontrar incluso en perfiles de usuario de cualquier red social, pero si recomienda igual tener cuidado en cuanto a posible robo de datos si es que se ingresa información extra, además al contener menos información y configuraciones que el caso 2, se demora menos la herramienta siendo un minuto y dieciocho segundos los requeridos para terminar el escaneo.

5.4. Caso Cuatro – Página Web Usualmente Utilizada (Intranet):

En este caso, el prototipo se utilizará para evaluar la seguridad en un sitio web que es

muy utilizado por la comunidad universitaria que es el banner de la universidad en su sede Santo Domingo, y se observará las posibles vulnerabilidades que puede llegar a presentar este sitio.

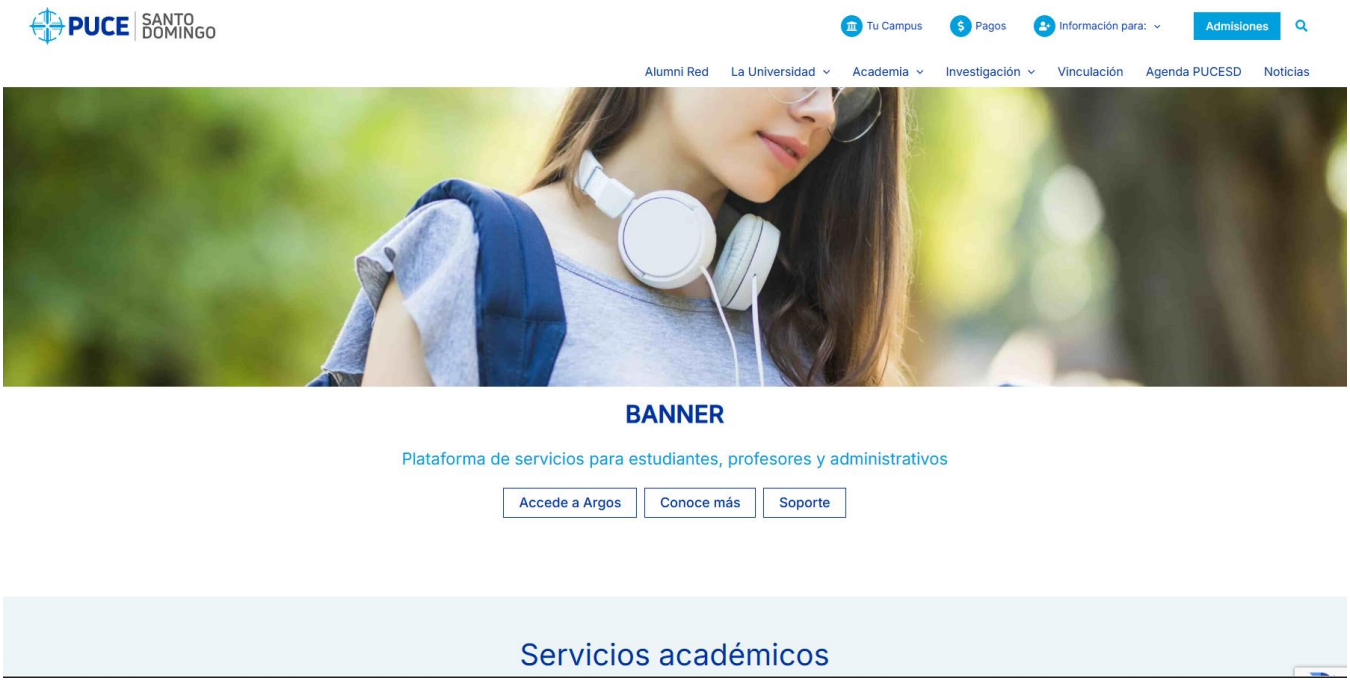

Figura 26 Página Banner de la PUCE (BANNER PUCE, 2025).

Figura 27 Despliegue de Resultados Generales en la UI Web (Paolo Ortiz, 2025).

En realidad, esta página al estar bien configurada no tiene posibles vulneraciones, y esto se debe en que en realidad al ser un sistema de banner que maneja información de estudiantes, y de vida académica (calificaciones, asistencias, etc.), no debería ser posible filtrarlo o ser vulnerable a cualquier tipo de ataque, por lo que es una página institucional bien configurada y con controles estrictos que hacen que sea incluso actualizado frente a componentes como cabeceras, archivos, etc.

Cabe destacar que, al ser una página ya completa con muchas más opciones la herramienta se toma seis minutos con cincuenta, esto ya que es un sitio que maneja mucha información, que el prototipo debe evaluar haciendo las pruebas correspondientes.

5.5. Logs del Servidor:

Conforme se hicieron las pruebas para evaluar el prototipo, se puede observar como el servidor muestra información de logs, esto puede ser información como los métodos

que se ejecutan, las conexiones remotas en la misma red privada, e incluso código o mensajes de error en caso de que los haya, como se puede observar en la figura 28.

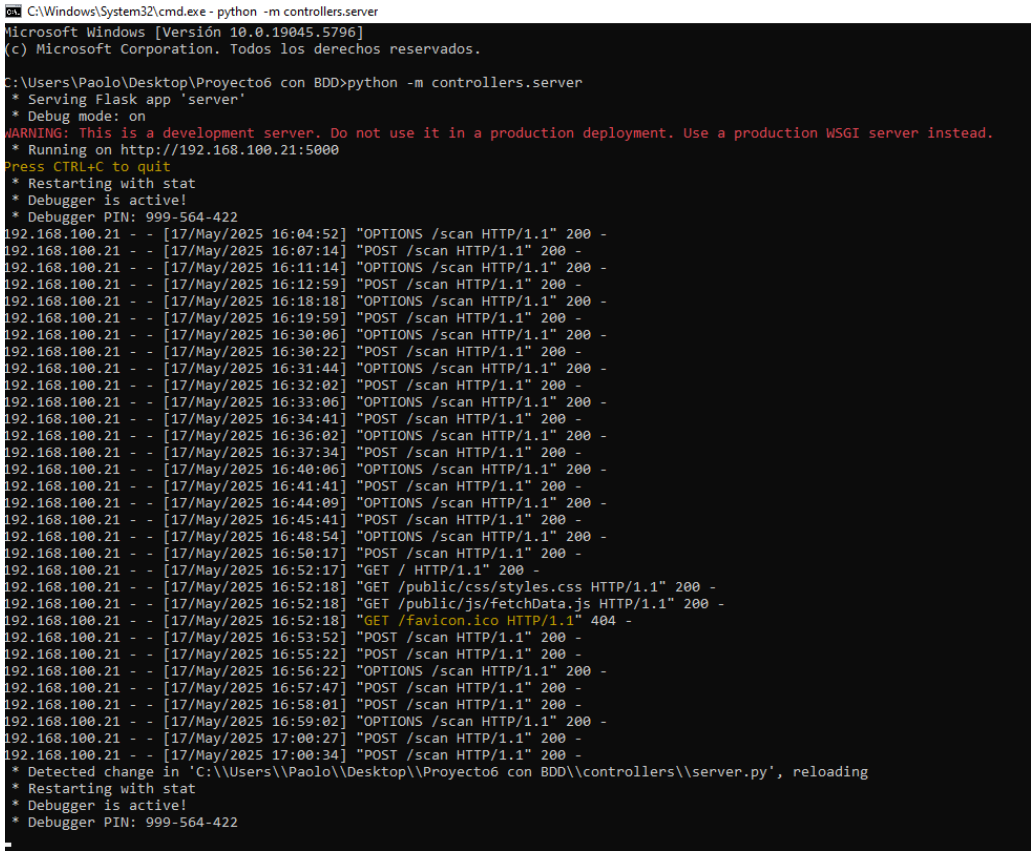

Figura 28 Logs de Consola del Servidor FLASK del prototipo (Paolo Ortiz, 2025).

Analizando la figura 28, se menciona que el propósito de la herramienta es apoyar al usuario a encontrar vulnerabilidades que generalmente pueden estar en los sitios que usan cotidianamente, pero también es siempre necesario llevar una auditoría de lo que se hace, por eso mismo es el propósito de estos logs, por si en algún momento se adaptará a une entorno empresarial y observar desde que dispositivos se están realizando posibles acciones y se puede visualizar que método se ejecuta del código.

5.6. Análisis General:

Al poseer un análisis por cada subcapítulo de este capítulo, se puede mencionar que es necesario reflexionar generalmente sobre la herramienta, sacando posibles conclusiones que nos permitan llevar a cabo un análisis de cuándo la herramienta es necesaria para entornos grandes y cuándo es solo de uso más personal.

Se considera que la herramienta podría ser utilizada en entornos empresariales cuando se lleven a cabo pruebas de nuevos sitios web propios de la empresa, donde se tengan configuraciones ya avanzadas de páginas o sitios web, para que la herramienta haga pruebas y los desarrolladores puedan realizar posibles cambios para mejorar estas vulnerabilidades, además para adaptar páginas de terceros, donde siempre se necesitará la seguridad empresarial, ya que se incluyen datos sensibles y protegidos, donde las estimaciones de tiempo de pruebas con el prototipo llegarían a ser en un rango de un minuto a tres minutos como máximo, para así poder encontrar posibles configuraciones a mejorar.

Y por último después de analizar el uso de la herramienta, se recomienda para un uso personal en cuanto a información para el usuario, que necesita o desea conocer que está expuesto en cuanto a datos, información, privacidad, etc. En los sitios que usa cotidianamente y que se podría mejorar, con el manejo de recomendaciones bastante fáciles de entender para cualquier clase de usuario, esto al ser páginas más completas o con configuraciones mejor realizadas, van a tener una duración estimada de dos minutos a seis minutos como máximo.

Como conclusión del análisis se puede afirmar que el prototipo cumple en cuánto a funcionalidad y facilidad de uso, además de permitir al usuario cumplir de forma rápida y ágil pruebas para detectar vulnerabilidades y genera un sistema de recomendaciones simples de entender para que se pueda realizar posibles acciones ante esas vulnerabilidades, o cuidados preventivos, además se estima que la herramienta tarda de uno a seis minutos como máximo en cada análisis, esto dependiendo de factores como configuración de la página, cantidad de datos, o cantidad de procesos de la misma.

Caso de Prueba	Prototipo Desarrollado
Caso 1: Página sin vulnerabilidades	Detectó solo alertas tipo INFO, sin falsos positivos. Tiempo de escaneo: 1:17 min.
Caso 2: Formulario con SQL Injection	Detectó vulnerabilidades de tipo MEDIUM (redirección), INFO y comprometimiento de datos. Tiempo de escaneo: 1:34 min.
Caso 3: Token Repetido (CSRF)	No identificó vulnerabilidad como crítica, pero emitió advertencias sobre reutilización de tokens y riesgo de filtración. Tiempo: 1:18 min.
Caso 4: Intranet Institucional	No detectó vulnerabilidades. Tiempo total: 6:50 min (por la complejidad del sitio).
Visibilidad de Resultados y UX	Visualizaciones por pasos (datos, vulnerabilidades, recomendaciones), enfoque pedagógico y adaptable a usuario sin conocimientos técnicos.

Tabla 3 Tabla Comparativa de los Resultados de las Pruebas Realizadas

CAPÍTULO VI: CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

6.1.1. Conclusiones a partir de los objetivos específicos

La fase de estudio y selección de herramientas para el entorno de escaneo de vulnerabilidades arrojó un conjunto de componentes coherentes y de alto rendimiento. La adopción de Nuclei como motor principal de escaneo, combinada con *Flask* para el conjunto de peticiones y PostgreSQL como almacén de mapeos en JSONB, garantiza una integración nativa, escalabilidad horizontal y una capacidad buena de extensión. Cada criterio, flexibilidad de plantillas, madurez del escáner asociado, licencia libre, facilidad de integración y rendimiento, se puede decir que facilitó decisiones de diseño que maximizan la modularidad y reducen el tiempo de incorporación de nuevas reglas de detección.

Asimismo, la evaluación de herramientas que complementan al prototipo (*JavaScript Fetch* API y *Tailwind* CSS en el *frontend*, o Psycopg2 en la capa de persistencia) confirmó que la combinación tecnológica elegida ofrece un equilibrio bueno entre facilidad de uso, mantenimiento y robustez. La combinación de Python con Flask en el servidor, SQL con JSONB en la base de datos y JavaScript ligero en el cliente reduce al mínimo las dependencias externas y hace más sencillo rastrear cada paso del escaneo. De este modo, cuando se deba actualizar o migrar el sistema, evita que algo se vea perjudicado o que baje el rendimiento.

El armado y puesta en marcha de este prototipo de escaneo de vulnerabilidades revela un esquema totalmente desacoplado y orientado a servicios. Al separar el motor de detección (Nuclei), la capa de lógica de negocio (Flask) y la persistencia (PostgreSQL), se consigue un flujo de datos muy ágil y rápido, precargar los mapeos de vulnerabilidades al inicio reduce en una forma muy grande, la espera en cada consulta y mantiene tiempos de respuesta constantes, incluso con cargas variables (cargas pequeñas y grandes).

La interfaz, armada con *HTML5*, *Fetch* API y *Tailwind* CSS, se lleva de la mejor manera con la API *RESTful* de Flask. Al darle “Iniciar escaneo”, el cronómetro interno comienza a contar sin pausas y, en cuanto aparecen los hallazgos o resultados, van mostrándose

uno tras otro: vulnerabilidades, datos expuestos, soluciones y recomendaciones. Todo fluye ligero, sin bloqueos, y permite al usuario ver paso a paso qué está pasando. Este enfoque, basado en un diseño API First y en principios de responsabilidad única, hace que el prototipo sea fácil de mantener y escalable, además permite añadir nuevos filtros o vistas, siendo que apenas se debe conectar el módulo correcto, sin reescribir mucho código.

A la hora de probarlo, se montaron cuatro escenarios muy reales: una página estática sin scripts, un formulario con huecos para inyección SQL y XSS, un panel con token CSRF fijo y la intranet de la universidad en su sede Santo Domingo. En todos los casos, la herramienta encontró lo que debía, ordenó los hallazgos por severidad y los emparejó con explicaciones claras sin lanzar falsos positivos molestos. Y lo mejor: incluso cuando había muchos datos, cada escaneo duró menos de seis minutos, un tiempo razonable que demuestra que el motor soporta cargas grandes y respeta los límites de rendimiento previstos.

Además, al medir y comparar de cerca el tiempo de respuesta, el uso de CPU y memoria, y lo rápido que la base de datos contesta cada consulta, se confirmó que el prototipo se comporta de manera muy estable, incluso cuando varios escaneos corren al mismo tiempo. Los registros que genera quedan guardados paso a paso, desde que arranca Nuclei hasta que llegan los resultados al navegador, lo que ayuda a revisar con detalle cada etapa y a encontrar mejoras. Con esta arquitectura completa, queda claro que el prototipo puede meterse en entornos de producción sin ningún problema o miedo, ofreciendo la confiabilidad y la calidad que cualquier equipo u organización necesita.

6.1.2. **Conclusiones Generales del Trabajo**

El prototipo respondió de manera ágil y constante al detectar las fallas más comunes (SQLi, XSS, CSRF), completando cada análisis en menos de seis minutos incluso cuando había demasiados datos o configuraciones. Eso demuestra que suma bien al flujo de trabajo continuo sin frenar a nadie ni armar cuellos de botella.

Al juntar Nuclei con una carga previa de patrones en memoria, se acortó la espera en cada consulta y se agregó contexto local a los resultados, con explicaciones y consejos listos para usar. De paso, se dejó atrás la salida en inglés de Nuclei, validando que la arquitectura diseñada en los capítulos III y IV escala bien sin importar cuánta información

haya.

En la parte visual, el combo de HTML5 y JavaScript se acopló de la mejor manera con el *backend* en Flask, entregando una experiencia intuitiva y sin trabas que hagan que el usuario lo considere “no útil”. Con ingresar la URL y dar clic en el botón para escanear, más el cronómetro en vivo, se hizo más fácil el entendimiento a todo tipo de usuario y se bajó el margen de error humano.

La base de datos en PostgreSQL resultó muy flexible para guardar explicaciones, datos expuestos, soluciones y sugerencias dentro de columnas JSONB. Esto permitió añadir nuevos patrones sin tocar la estructura de tablas, de modo que el prototipo sigue activo siempre y con opciones de crecer sin inactividad.

Gracias a índices correctamente creados en las columnas de soluciones y recomendaciones, el acceso a la información fue casi instantáneo, aunque hubiera cientos de registros. Eso confirma que PostgreSQL soporta bien este tipo de “prototipos” donde las listas pueden variar en tamaño.

Separar el escáner (Nuclei) de la base de datos limpió el código y bajó el conflicto de uso entre módulos. Cada parte hace lo suyo de forma independiente, lo que hace más fácil migrar componentes sin romper nada y suma puntos para el trabajo en equipo.

Al probar con páginas intencionalmente vulnerables, el prototipo encuentra vulnerabilidades relacionadas a SQLi, XSS y CSRF sin alardes de falsos positivos. El orden por severidad y las breves explicaciones dejaron claro qué hacer y fortalecieron la confiabilidad tanto para desarrolladores como para auditores.

Con sitios muy sencillos, que solo tenían HTML sin *scripts*, el sistema solo lanzó alertas de tipo INFO por cabeceras faltantes, justo lo que se esperaba. Esa claridad en diferenciar casos base de situaciones de riesgo real es clave para no hacer que al usuario se le muestre datos sin importancia.

Se pusieron a prueba dos formatos de interfaz: una página web minimalista con un solo campo de URL y un botón, lo cual explica su facilidad de uso. Además, una extensión de navegador que permite escanear con la misma velocidad. La primera mantiene las cosas simples, la segunda da un extra de agilidad.

La extensión brilló al librar al usuario de copiar y pegar direcciones, escaneando la pestaña activa con un clic. Eso acelera los chequeos en entornos reales y hace más cómodo el día a día del desarrollador.

Al tener ambos modos disponibles, el prototipo sirve para tareas puntuales y también para inspecciones rápidas en producción. Así se asegura de que sea usable tanto a quienes apenas están aprendiendo como a los expertos de seguridad.

En las pruebas de carga, ni el CPU ni la memoria sufrieron valores atípicos, y los tiempos se mantuvieron estables. Eso quiere decir que el sistema puede correr en máquinas sencillas sin pedir hardware de última generación.

Los registros detallados en consola de Flask dejaron un rastro completo de cada paso: conexiones, ejecución de Nuclei y operaciones propias que permiten al usuario y al auditor entender que se hace en cada paso. Esto no solo ayuda a depurar, sino que facilita conectar todo con sistemas de monitoreo de cualquier empresa.

La arquitectura elegida (Flask, PostgreSQL, Nuclei y Fetch API) permitió concentrarse en la lógica de detección y en la experiencia de usuario, sin perder tiempo en configurar infraestructuras complejas. Fue una decisión que nos permitió combinar herramientas sólidas y bien documentadas con la solución propuesta.

Presentar recomendaciones en un tono claro, pero sin perder el rigor técnico hizo que el prototipo no solo avise de fallos, sino que enseñe cómo arreglarlos. Ese extra convierte a la herramienta en un aliado tanto para aprender como para mantener la seguridad de la mejor manera.

En definitiva, esta herramienta cumple su misión de ofrecer un análisis de seguridad rápido, modular y fácil de usar. Deja una base firme sobre la cual sumar funciones, y demuestra su valor tanto en desarrollo como en operación continua.

6.2. Recomendaciones

6.2.1. Recomendaciones para el trabajo actual (para el usuario)

Se recomienda tener y colocar el prototipo como etapa inicial en el ciclo de desarrollo, de modo que al evaluar cada versión en entornos de prueba se ejecute un escaneo

automático de SQLi, XSS y CSRF. Esto permite encontrar y corregir vulnerabilidades menores de forma oportuna, evitando que se sumen fallos de seguridad y reduciendo el riesgo de hallazgos que sean complicados de corregir tarde en auditorías.

Conviene aprovechar el cronómetro interno para registrar el rendimiento en cada ejecución. Al comparar los tiempos de escaneo tras cambios de código o cambios en la configuración del servidor, el equipo obtiene un estimado de la eficacia de sus mejoras y puede ajustar recursos según se requiera.

Es aconsejable validar primero en páginas estáticas y en entornos de prueba controlados antes de aplicar el escaneo a entornos reales. De esta manera se reducen los falsos positivos y se van mejorando los valores necesarios de las plantillas de Nuclei para mostrar mejores resultados y coherentes cuando se analicen aplicaciones de producción.

Para mantener un documento histórico de hallazgos, es bueno crear un repositorio o carpeta de reportes en donde se manejen versiones (por ejemplo, /reportes). Así se documenta la evolución de la seguridad a lo largo del tiempo y se puede comparar si una vulnerabilidad previamente corregida, vuelve a aparecer después de realizar cambios.

Se aconseja tener 10–15 minutos en reuniones periódicas para revisar el informe que genera el prototipo, teniendo énfasis en las vulnerabilidades de severidad media y alta. Esta revisión mantiene al equipo alineado, y ayuda a la toma de decisiones rápida y evita que los riesgos críticos queden desatendidos o que se sigan mostrando.

Para facilitar la comprensión a usuarios no muy técnicos, conviene clasificar las vulnerabilidades en categorías claras: “Exposición de datos”, “Configuración insegura” e “Inyección de código”. Este orden hace más simple la interpretación y permite asignar responsables de forma más rápida y directa.

También resulta valioso, que se varíen los datos de entrada en los formularios (texto normal, cadenas con caracteres especiales y cadenas largas) para verificar la funcionalidad ante distintos tamaños y configuraciones de ataques. Esto amplía la cobertura de pruebas sin necesidad de integrar herramientas adicionales.

Se puede programar los escaneos en horarios de baja actividad (por ejemplo, al inicio de la jornada) y revisar los resultados a primera hora. De esta forma, el equipo inicia el

día con un informe claro de la seguridad de la aplicación y puede organizar las correcciones en las tareas diarias.

Para el seguimiento de las acciones correctivas que se generan como recomendaciones al usuario, se sugiere exportar los resultados a formatos (CSV o JSON) y, además, integrarlos en documentos donde se evidencie la gestión de incidencias. Cada hallazgo se convierte en un *ticket* con prioridad y fecha de detección, lo que hace mucho más fácil el control del ciclo de vida de la vulnerabilidad.

Al cerrar el *ticket* que se une a un reporte de una vulnerabilidad, debe registrarse la fecha de resolución y que se realizó. Esto construye un historial que permite medir el tiempo medio de corrección y hallar áreas que requieren mayor atención o capacitación.

Finalmente, agrupar los *tickets* por tipo de vulnerabilidad ofrece información valiosa sobre las categorías donde más se observa vulnerabilidades comúnmente. Con estos indicadores, se pueden hacer sesiones de formación específicas y orientar las mejoras hacia los riesgos más frecuentes en el proyecto.

6.2.2. Recomendaciones de mejora para el futuro

Para ampliar el alcance del prototipo, resulta bueno documentar y exponer de la mejor manera sus limitaciones actuales, tales como la cobertura limitada y de poca complejidad a vulnerabilidades OWASP Top 10 y la ausencia de integración nativa con pipelines de CI/CD. Incluir un apartado de “Limitaciones y Alcances” en la documentación permitirá a futuros usuarios y evaluadores entender de la mejor manera los escenarios en los que la herramienta entrega resultados beunos y aquellos que requieren desarrollo adicional.

Se sugiere incorporar un motor de análisis de código (SAST) y otro de pruebas avanzadas (DAST) que sumen a las plantillas de Nuclei. De este modo, el prototipo no solo escanearía firmas y patrones conocidos, sino que también identificaría errores lógicos y vulnerabilidades de ejecución que no están en las reglas basadas en firmas, haciendo mejor la cobertura y manejo de la evaluación.

Es conveniente ampliar la librería de plantillas para cubrir vectores emergentes, por ejemplo, SSRF, XXE o API GraphQL, adaptando reglas YAML específicas para cada caso. Esta extensión garantizará que el prototipo evolucione conforme crecen las amenazas, manteniendo su relevancia en entornos que implementen tecnologías

modernas y microservicios.

Para facilitar la actualización de los mapeos, podría realizarse un módulo administrativo web que permita crear, editar o desactivar reglas y explicaciones directamente desde la interfaz administrativa. Un CRUD sencillo con roles de administrador y auditor proveería control de versiones y seguimiento de cambios, reduciendo la necesidad de intervenciones manuales en la base de datos.

Se recomienda desarrollar paneles de control de monitoreo en tiempo real, para poder ver métricas clave como, tasa de escaneos, tiempos medios de ejecución, número de errores o vulnerabildades por severidad y configurar alertas automáticas que avisen de picos de carga o de la aparición de vulnerabilidades críticas en producción.

Para mejorar la experiencia global, conviene estandarizar la interfaz y las explicaciones de las vulnerabilidades, de manera que tenga un alcance más internacional, soportando múltiples idiomas mediante ficheros de traducción. Esto agranda el público objetivo del prototipo y facilitará su adopción en organizaciones con equipos multiculturales.

Finalmente, sería bueno implementar pruebas de usabilidad y accesibilidad (WCAG 2.1), garantizando que la herramienta sea manojeada y operada por usuarios con discapacidad visual o motora. Integrar atajos de teclado, y un contraste adecuado hará que el prototipo cumpla con estándares de accesibilidad y mejore la satisfacción de todos los usuarios.

BIBLIOGRAFÍA

Acunetix. (s.f.). Escáner de vulnerabilidades. Acunetix. Recuperado el 20 de abril de 2025, de https://www.acunetix.com/vulnerability-scanner/

Amazon Web Services. (2021). State of Cloud Security Report (2021). Recuperado de https://fidelissecurity.com/resource/report/aws-cloud-security-report/

Astra Security. (s.f.). Informe de escaneo de vulnerabilidades. GetAstra. Recuperado el 20 de abril de 2025, de https://www.getastra.com/blog/security-audit/vulnerability-scanning-report/

Barth, A., Jackson, C., & Mitchell, J. C. (2008). Robust defenses for cross-site request forgery. En Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08). https://doi.org/10.1145/1455770.1455784

Committee on National Security Systems. (2010). National Information Assurance Glossary (CNSSI No. 4009). Washington, DC: Committee on National Security Systems.

Cycognito. (s.f.). Escaneo de vulnerabilidades. Cycognito. Recuperado el 20 de abril de 2025, de https://www.cycognito.com/learn/vulnerability-assessment/vulnerability-scanning.php

EveryTI. (s.f.). 5 pasos para interpretar y actuar sobre los resultados de tu análisis de vulnerabilidades. EveryTI. Recuperado el 20 de abril de 2025, de https://everyti.com/5-pasos-para-interpretar-y-actuar-sobre-los-resultados-de-tu-analisis-de-vulnerabilidades/

Forrester Research. (2020). The State of Application Security. Forrester Research.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. En Proceedings of the IEEE International Symposium on Secure Software Engineering. Recuperado de https://faculty.cc.gatech.edu/~orso/papers/halfond.viegas.orso.ISSSE06.pdf

Indusface. (s.f.). ¿Qué es un escaneo de seguridad autenticado? Indusface. Recuperado el 20 de abril de 2025, de https://www.indusface.com/blog/what-is-an-authenticated-security-scan/

International Organization for Standardization. (2013). ISO/IEC 27001:2013 Information technology — Security techniques — Information security management systems — Requirements. Geneva, Switzerland: ISO.

International Organization for Standardization. (2018). ISO/IEC 27000:2018 Information technology — Security techniques — Information security management systems — Overview and vocabulary. Geneva, Switzerland: ISO.

Kiuwan. (s.f.). Escaneo de vulnerabilidades vs. pruebas de penetración: diferencias clave. Kiuwan. Recuperado el 20 de abril de 2025, de https://www.kiuwan.com/blog/vulnerability-scanning-vs-penetration-testing-key-differences/

Mitre Corporation. (2020). Common Weakness Enumeration (CWE). Recuperado de https://cwe.mitre.org

National Institute of Standards and Technology. (2007). Guidelines on Securing Public Web Servers (Special Publication 800-44 Rev. 2). Gaithersburg, MD: NIST. Recuperado de https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-44ver2.pdf

National Institute of Standards and Technology. (2008). Technical Guide to Information Security Testing and Assessment (NIST Special Publication 800-115). Gaithersburg, MD: NIST.

National Institute of Standards and Technology. (2017). Digital Identity Guidelines: Authentication and Lifecycle Management (NIST Special Publication 800-63B). Gaithersburg, MD: NIST.

National Institute of Standards and Technology. (2018). Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy (NIST Special Publication 800-37 Rev. 2). Gaithersburg, MD: NIST.

National Institute of Standards and Technology. (2020). Security and Privacy Controls for Information Systems and Organizations (NIST Special Publication 800-53 Rev. 5). Gaithersburg, MD: NIST.

OWASP Foundation. (2019). OWASP Application Security Verification Standard – ASVS 4.0.1. Recuperado de https://owasp.org/www-project-application-security-verification-standard/

OWASP Foundation. (2021). OWASP Top Ten Project. Recuperado de https://owasp.org/Top10/

OWASP Foundation. (2021). OWASP Top Ten Web Application Security Risks. Recuperado de https://owasp.org/www-project-top-ten/

Palo Alto Networks. (s.f.). Escaneo de vulnerabilidades. Palo Alto Networks. Recuperado el 20 de abril de 2025, de https://www.paloaltonetworks.com/cyberpedia/vulnerability-scanning

Stuttard, D., & Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. Wiley.

Wiz. (s.f.). Escaneo de vulnerabilidades. Wiz. Recuperado el 20 de abril de 2025, de https://www.wiz.io/es-es/academy/vulnerability-scanning

ANEXOS

1. Instalación de las herramientas necesarias

Instalación y configuración de herramientas seleccionadas

A continuación, se detallan los pasos para preparar el entorno de trabajo en Windows 10 y facilitar la reproducibilidad

Python y entorno virtual

Instalar Python 3.10+ y pip desde python.org

python -m venv venv

venv\Scripts\activate

pip install --upgrade pip

pip install flask flask-cors psycopg2-binary

Nuclei

Descargar el ejecutable para Windows desde:

https://github.com/projectdiscovery/nuclei/releases

Descomprimir en C:\Program Files\Nuclei\ y añadir la carpeta al PATH del sistema.

Verificar:

nuclei -version

PostgreSQL 17

Instalar PostgreSQL y pgAdmin desde https://www.postgresql.org/download/

Durante la instalación, configurar el usuario postgres con contraseña Prototipo123.

Crear la base de datos PROYECTO y ejecutar el script de tablas (ver Anexo 4).

Comprobar conexión:

psql -U postgres -d PROYECTO -h localhost

Variables de entorno

Añadir en el sistema (Panel de Control → Sistema → Variables de entorno):

NUCLEI_PATH="C:\Program Files\Nuclei\nuclei.exe"

DB_HOST="localhost"

DB_NAME="PROYECTO"

DB_USER="postgres"

DB_PASS=" Prototipo123"

Configurar el servidor Flask

Clonar el repositorio dentro de controllers/.

En server.py, leer las variables de entorno con os.getenv().

Ejecutar:

set FLASK_APP=controllers/server.py

set FLASK_ENV=development

flask run --host=0.0.0.0 --port=5000

Interfaz web y extensión

Colocar home.html, css/styles.css y js/fetchData.js dentro de public/.

Para la extensión, incluir manifest.json, popup.html, popup.css, background.js y cargarla en

Chrome/Edge/Opera en modo desarrollador con “Load unpacked”.

Pruebas iniciales

Abrir en el navegador: http://192.168.100.21:5000/.

Verificar que al hacer clic en “Iniciar Escaneo” salgan resultados y cronómetro.

Probar llamadas directas con Postman a POST http://<IP>:5000/scan con JSON {“url":

"https://ejemplo.com”}.

Con esto, todas las herramientas quedan instaladas, configuradas y listas para su uso en el

proyecto, garantizando un entorno coherente y reproducible para desarrollos futuros.

2. **Enlace al repositorio del proyecto (Prototipo Final):**

https://drive.google.com/file/d/1tvEworkdLZGwAgZXUonV4NtAaNCXNcng/view?usp=drive_link

3. **Enlace al repositorio para las páginas de prueba (Fase de Pruebas y Análisis de Resultados):**

https://drive.google.com/file/d/18KS7gIYL6KRdQeAzsZq9McJTCZf7J1LV/view?usp=drive_link

4. **Enlace al repositorio para el script de creación de tablas de la base de datos e inserción de datos necesarios para la funcionalidad del escaneo:**

https://drive.google.com/file/d/1zEc4m6HpyOVm_dJa7YfjEFOfs3NpvDDq/view?usp=sharing