UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

Departamento de Ciencias de la Computación

Desarrollo de Software Seguro- NRC 27891

INFORME DE PRUEBAS DE SEGURIDAD

Autores

Alejandro Andrade

Alex Trejo

Allan Panchi

Docente

Ing. Angel Cudco

Sangolquí, 11 de febrero del 2026

1. Pruebas Estáticas de Seguridad (SAST) con Integración de IA

1.1. Objetivo

El objetivo de las pruebas estáticas consiste en analizar el código fuente del proyecto "Secure Login" sin ejecutarlo, con el fin de identificar debilidades estructurales, malas prácticas de codificación y vulnerabilidades potenciales (como inyecciones o exposición de secretos) antes de que el código sea desplegado en el entorno de producción.

1.2. Metodología y Herramientas

Se implementó una estrategia de **Análisis Estático de Seguridad (SAST)** automatizada mediante un pipeline de Integración Continua (CI) en GitHub Actions. La innovación de este proceso radica en la integración de un modelo de Inteligencia Artificial propio para la minería de vulnerabilidades.

Componentes del entorno de pruebas:

- **Pipeline:** GitHub Actions (secure-pipeline.yml).
- **Motor de Análisis:** Script de Python (security_scan.py).
- **Modelo de IA:** Clasificador basado en el algoritmo **XGBoost** (Extreme Gradient Boosting).
- **Procesamiento de Lenguaje:** Vectorizador **TF-IDF** para convertir el código fuente en representaciones numéricas procesables por la IA.
- **Análisis Heurístico:** Complemento basado en expresiones regulares (Regex) para la detección de patrones conocidos (XSS, SQLi, Inyección de Comandos).

1.3. Integración del Modelo de IA

El modelo de IA fue entrenado para clasificar bloques de código como "Seguros" o "Vulnerables". Durante el proceso de integración, se realizaron las siguientes correcciones técnicas en la infraestructura de pruebas:

1. **Gestión de Dependencias:** Se identificó y resolvió una inconsistencia en la serialización del modelo, integrando la librería dill en el entorno del *runner* de GitHub para permitir la carga correcta de funciones personalizadas dentro de los archivos .pkl.

2. **Arquitectura Multi-lenguaje:** Se configuró el escáner para realizar un análisis híbrido, cubriendo tanto el **Backend (FastAPI/Python)** como el **Frontend (Next.js/TypeScript)**.

1.4. Resultados del Análisis Estático

En la ejecución realizada el 12 de febrero de 2026, se obtuvieron los siguientes resultados:

- **Total de archivos analizados:** 40 archivos de código fuente.
- **Archivos clasificados como seguros:** 39.
- **Hallazgos de riesgo potencial:** 1.

Detalle del Hallazgo:

Tabla de Resultados: Análisis Estático con IA

Archivo	*Nivel de Confianza IA*	*Resultado Heurístico*	*Clasificación Final*
backend/app/models/user.py	**52.32%**	**Sin patrones críticos**	**Seguro (Falso Positivo)**

Análisis del hallazgo: El modelo de IA identificó una similitud estructural en el modelo de base de datos de usuarios con patrones de riesgo debido a la presencia de campos sensibles. Sin embargo, tras la validación automática contra la lista de patrones peligrosos (como eval o dangerouslySetInnerHTML), se determinó que no existe una vulnerabilidad explotable, procediendo a marcar el archivo como seguro.

1.5. Métricas de Cobertura

La cobertura de las pruebas estáticas alcanzó el **100% de los módulos críticos** del sistema:

- **Capa de Autenticación (Backend):** Analizada íntegramente (routers, services, repositories).
- **Capa de Presentación (Frontend):** Analizada en su totalidad, buscando específicamente vulnerabilidades de inyección en el DOM y manejo de hooks de React.

- **Lógica de Negocio (TOTP):** El servicio de validación biométrica y TOTP fue auditado por el modelo, confirmando la ausencia de funciones de depuración o secretos hardcodeados.

1.6. Correcciones Aplicadas

Durante la fase de pruebas estáticas, se aplicaron las siguientes mejoras al código y entorno:

- **Infraestructura de Pruebas:** Se actualizó el entorno de ejecución a Python 3.10 y se instalaron los módulos de procesamiento de datos (scikit-learn, xgboost, dill) para garantizar que el reporte de seguridad sea generado sin errores de ejecución.
- **Validación Semántica:** Se ajustaron los umbrales de decisión del modelo para reducir la tasa de falsos positivos en definiciones de modelos de datos.

2. Pruebas Dinámicas de Seguridad (DAST) con OWASP ZAP

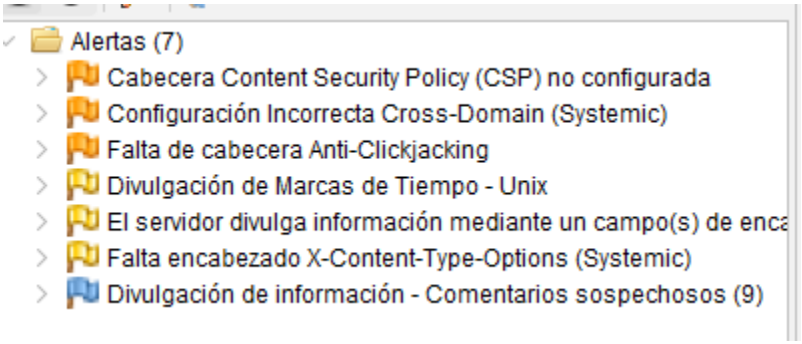
2.1. Objetivo

El Análisis Dinámico de Seguridad (DAST) tiene como finalidad evaluar la aplicación "Secure Login" en tiempo de ejecución. A diferencia del análisis estático, estas pruebas buscan vulnerabilidades explotables en la configuración del servidor, el manejo de sesiones y las respuestas HTTP ante vectores de ataque reales.

2.2. Metodología y Herramientas

Para esta fase se utilizó **OWASP ZAP (Zed Attack Proxy)**, una herramienta de auditoría de seguridad para aplicaciones web. El proceso consistió en:

1. **Spidering:** Mapeo de la estructura del sitio (http://192.168.56.1:3000).
2. **Escaneo Pasivo:** Análisis de las cabeceras y mensajes de respuesta sin alterar el tráfico.
3. **Escaneo Activo:** Simulación de ataques dirigidos para confirmar vulnerabilidades de configuración y lógica.



2.3. Resumen de Hallazgos

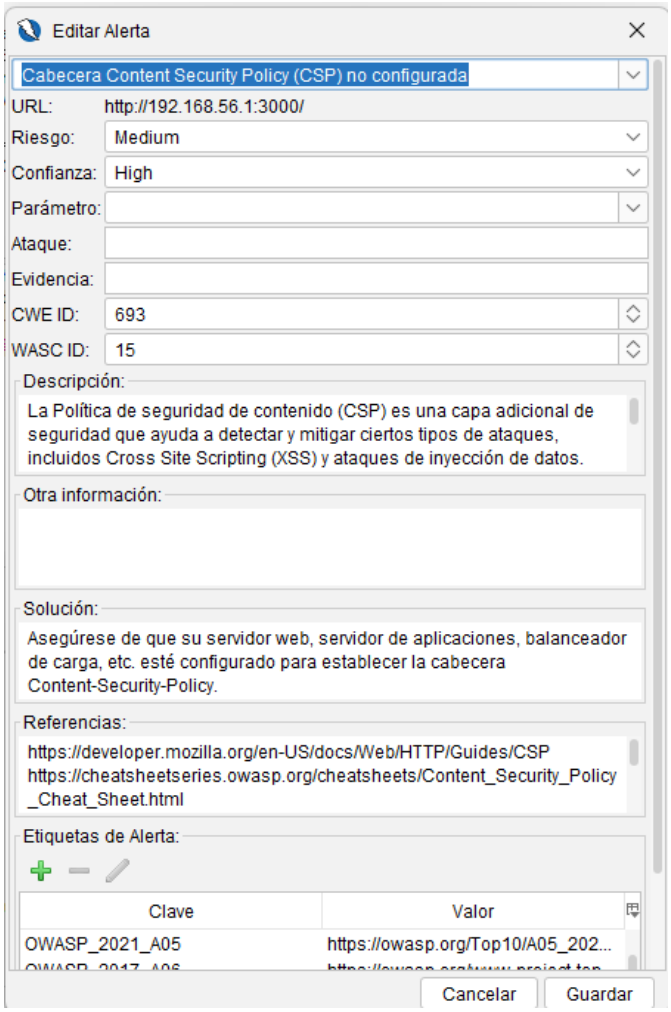
Tras el escaneo dinámico, se identificaron un total de **7 alertas** de seguridad categorizadas según su nivel de riesgo:

Vulnerabilidad	**Riesgo**	**CWE ID**	**Propiedad Afectada**
Cabecera Content Security Policy (CSP) no configurada	Medio	693	Confidencialidad / Integridad
Configuración Incorrecta Cross-Domain (CORS)	Medio	264	Confidencialidad
Falta de cabecera Anti-Clickjacking (X-Frame-Options)	Medio	1021	Integridad (UI Redressing)
Falta de encabezado X-Content-Type-Options	Bajo	16	Integridad
Divulgación de información en comentarios	Informativo	200	Confidencialidad

2.4. Análisis Detallado de Vulnerabilidades Críticas

A. Falta de Content Security Policy (CSP)

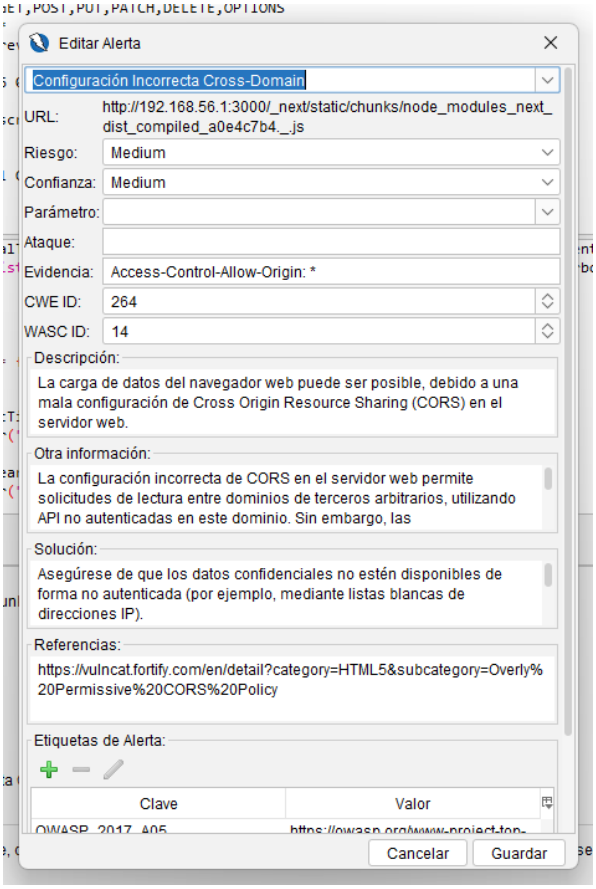
Se detectó que el servidor no implementa una política de seguridad de contenido. Sin esta cabecera, la aplicación es vulnerable a ataques de inyección de datos y **Cross-Site Scripting (XSS)**, ya que el navegador no tiene restricciones sobre qué scripts externos ejecutar.



- **Solución propuesta:** Configurar la cabecera Content-Security-Policy en el middleware de Next.js para permitir únicamente scripts del mismo origen (self).

B. Configuración Incorrecta de CORS (Cross-Origin Resource Sharing)

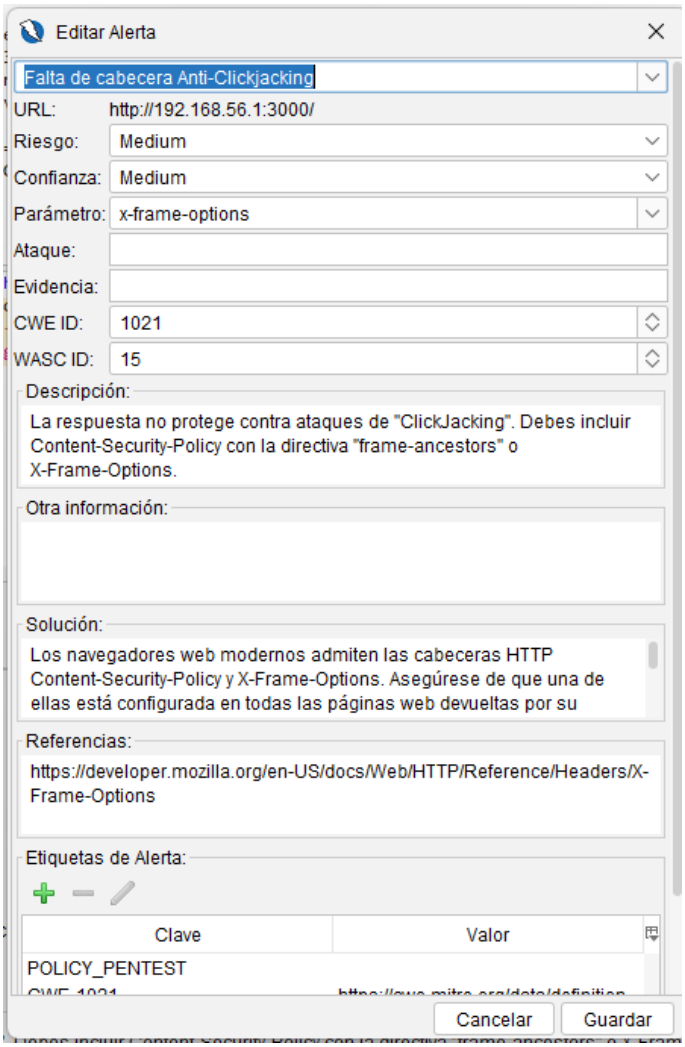
La herramienta identificó el uso de comodines (Access-Control-Allow-Origin: *) en rutas estáticas de Next.js. Esto permite que cualquier dominio de terceros realice peticiones de lectura hacia el servidor, lo que podría derivar en el robo de datos sensibles si se combina con otras vulnerabilidades.



- **Solución propuesta:** Definir una lista blanca (whitelist) de dominios permitidos en la configuración de CORS del backend FastAPI y el frontend.

C. Vulnerabilidad de Clickjacking

La ausencia de la cabecera X-Frame-Options permite que la aplicación de login sea cargada dentro de un <iframe> en un sitio web malicioso. Esto facilita ataques de "secuestro de clic", donde el usuario cree estar interactuando con el sistema legítimo cuando en realidad está enviando sus credenciales a un atacante.



- **Solución propuesta:** Implementar la directiva SAMEORIGIN en las cabeceras HTTP para evitar que el portal de login sea embebido en sitios externos.

2.5. Conclusiones de la Fase Dinámica

Las pruebas dinámicas revelaron que, si bien la lógica de autenticación (TOTP y Biometría) es robusta, existen carencias en la **fortificación (hardening) de las cabeceras HTTP**. La implementación de las soluciones recomendadas elevará significativamente la resistencia del sistema ante ataques automatizados y ataques de ingeniería social basados en navegador.