Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Software

Desarrollo de Software Seguro

Proyecto Integrador Parcial II

Profesor: Geovanny Cudco

28 de noviembre de 2025

1. Tema

Desarrollo e Implementación de un Pipeline CI/CD Seguro con integración de IA para la Detección Automática de Vulnerabilidades en código fuente mediante un Modelo de Minería de Datos.

2. Tipo de actividad

Proyecto práctico individual o en equipo (máximo 3 personas).

3. Objetivo

Diseñar, implementar y demostrar un pipeline CI/CD completamente automatizado y seguro que integre un modelo de inteligencia artificial basado en técnicas de minería de datos capaz de clasificar código fuente como seguro o vulnerable, permitiendo que únicamente el código considerado seguro llegue a producción, garantizando así la aplicación de los principios de *Secure DevOps* y *Shift-Left Security*.

IMPORTANTE: Está estrictamente prohibido la incorporación de tecnologías de *Large Language Models* (LLM) como GPT, Claude, Llama, CodeLlama, etc. El modelo de IA debe ser obligatoriamente un clasificador de minería de datos tradicional (scikit-learn, XGBoost, Random Forest, SVM, etc.) entrenado por Ud, usando datasets públicos (o propios) de código vulnerable/seguro.

4. Descripción

El proyecto consiste en crear una infraestructura CI/CD segura y automatizada que procese código fuente presentado por un usuario en una rama de testing de un repositorio Git (usando GitHub o GitLab). El flujo debe ser el siguiente:

4.1. Flujo de trabajo requerido

4.1.1. Ramas obligatorias

- **dev** → rama de desarrollo (donde el desarrollador hace *push*)
- **test** → rama de *staging*/pruebas
- **main** → rama de producción

4.1.2. Trigger

- El pipeline se activa automáticamente al crear un *Pull Request* de **dev** → **test**.

4.1.3. Etapas del Pipeline (todas obligatorias y automatizadas)

Etapa 1: Revisión de Seguridad con Modelo de Minería de Datos

- Se ejecuta un job que descarga el diff del PR.
- Se procesa el código modificado (extrayendo features como tokens, AST simplificado, patrones de llamadas a funciones peligrosas, uso de sanitización, etc.).
- Se clasifica el código como SEGURO o VULNERABLE utilizando exclusivamente un modelo de machine learning clásico (scikit-learn, XGBoost, etc.).
- Si el modelo devuelve **"VULNERABLE"**:
  - El PR se marca automáticamente como rejected.º se bloquea el merge.
  - Se crea un comentario detallado en el PR con la probabilidad y tipo de vulnerabilidad detectada.
  - Se envía notificación inmediata vía Telegram al desarrollador con el detalle.
  - Se aplica la etiqueta "fixing-requiredz se crea una issue automática vinculada.
- Si el modelo devuelve "SEGURO"→ continúa el pipeline.

Etapa 2: Merge Automático a rama test + Pruebas

- Merge automático a test.
- Ejecución de pruebas unitarias e integración (pytest, Jest, JUnit, etc.).
- Si alguna prueba falla → bloqueo y notificación Telegram + etiqueta "tests-failed".

Etapa 3: Merge a main y Despliegue en Producción

- Solo si todo lo anterior pasó → merge automático a main.
- Build de imagen Docker y despliegue automático en un proveedor gratuito:
  - Opciones permitidas: Render, Railway, Fly.io, Vercel (para frontend), Northflank, o Docker Hub + Play with Docker (para demo).
  - También permitido: Heroku (si aún tiene plan gratuito disponible).
  - Otro que considere el estudiante.
- Notificación final de éxito vía Telegram y/o email.

4.1.4. Notificaciones obligatorias en todas las fases:

Deben enviarse mensajes vía Telegram (bot propio) o correo electrónico en los siguientes eventos:

- Inicio de revisión de seguridad
- Resultado de la clasificación del modelo (seguro/vulnerable + probabilidad)
- Merge a test realizado
- Resultado de pruebas
- Despliegue en producción exitoso o fallido
- Rechazo por vulnerabilidad (con detalle)

5. Requisitos

a. Modelo de minería de datos entrenado por el estudiante (deben entregar el .pkl o .joblib).

b. Dataset utilizado debe ser público (recomendados: kaggle, Big-Vul, DiverseVul, CVEFixes, o synthetic con Juliet Test Suite).

c. Features mínimas: tokens, AST depth, llamadas a funciones peligrosas (eval, exec, subprocess, SQL raw, etc.), presencia de sanitización/escapes.

d. Accuracy mínima demostrada: 82 % en validación cruzada (deben mostrarlo en el README).

e. Telegram Bot propio (token en GitHub Secrets).

f. Despliegue real y funcional (debe estar online y accesible).

g. Branch protection rules activadas en test y main (requerir revisión de seguridad aprobada).

6. Formato de entrega

a. Repositorio GitHub o GitLab público o con acceso otorgado al profesor.

b. README.md completo con::

- Instrucciones de setup del pipeline
- Cómo entrenaron el modelo (notebook incluido)
- Capturas y enlace al bot de Telegram
- Enlace al despliegue en producción

c. Informe técnico en **Latex** se adjunta el formato del informe.

d. Exposición de 8-12 minutos mostrando:

- Código vulnerable → rechazo automático
- Código seguro → flujo completo hasta producción

7. Fecha de entrega

Fecha de Entrega: 17 de diciembre de 2025, 23:59 horas.

Nota importante:
Bajo ningún concepto se recibirán actividades fuera del plazo establecido. No se otorgarán prórrogas individuales ni se aceptarán entregas tardías por ningún medio. Es responsabilidad del estudiante gestionar oportunamente su trabajo y asegurar el cumplimiento del cronograma.

8. Criterios de Evaluación

a. Funcionalidad completa del pipeline (automatización total): **6 puntos**

b. Modelo de minería de datos propio y efectivo (prohibido LLM): **6 puntos**

c. Notificaciones Telegram/correo electrónico en todas las fases + issues automáticas: **3 puntos**

d. Despliegue automático en proveedor gratuito y funcional: **3 puntos**

e. Calidad del informe y documentación (README + notebook): **2 puntos**

Penalizaciones

a. Uso de LLM (incluso parcial): 20 puntos (nota 0 automático)

b. Pipeline no completamente automático: 4 a 6 puntos

c. Sin despliegue real: 3 puntos

d. Otras.