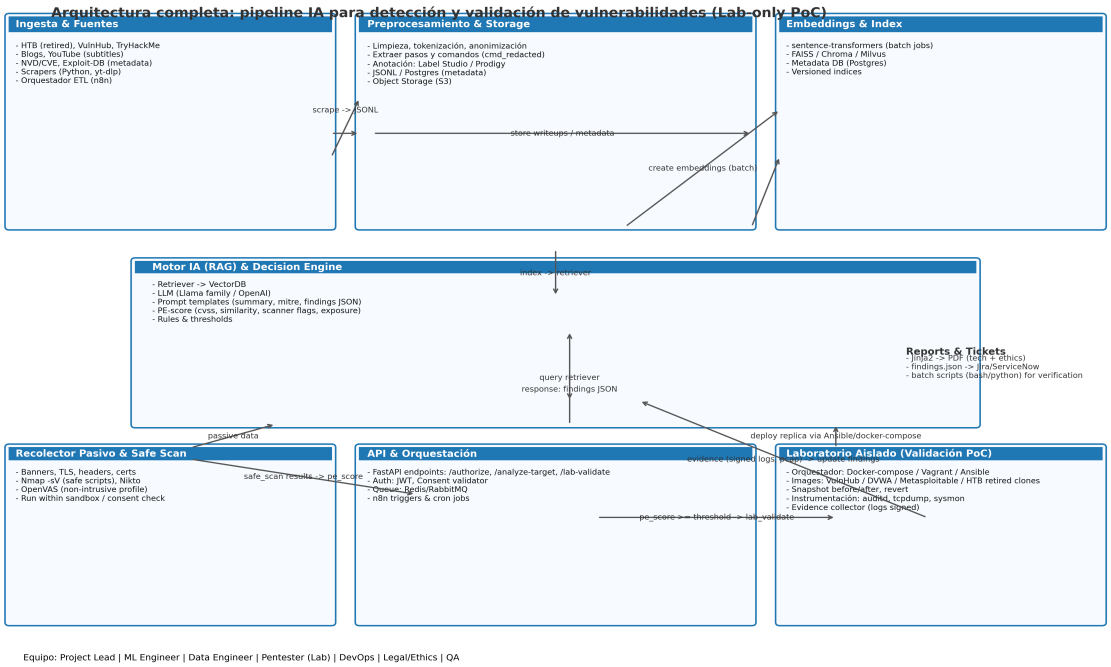
Plan de Tesis - Paso a Paso Completo

IA para detección, priorización y validación de vulnerabilidades (Lab-only PoC)



Portada

Plan de Tesis: IA para detección, priorización y validación de vulnerabilidades (Proceso paso a paso) Autor: Alexander Fecha: 2025-10-07 Versión: 1.1

Resumen ejecutivo

Este documento describe, paso a paso, cómo implementar el sistema propuesto: ingestión de write-ups públicos, indexación (RAG), módulo de análisis de objetivos, laboratorio aislado para validación de explotabilidad (PoC) y generación automatizada de informes y planes de remediación. Todas las acciones destructivas o de explotación se realizarán exclusivamente en entornos controlados y autorizados.

Pre-requisitos de infraestructura

Hardware mínimo: 16GB RAM, 4 vCPU, 200GB disco (más GPU para fine-tuning opcional). Software: Docker, docker-compose, Python 3.10+, Git, VirtualBox (si usa Vagrant), Ansible, PostgreSQL, Redis, FAISS/Chroma dependencies. Accesos: cuenta GitHub, S3 (u object store), servidor con capacidad para ejecutar containers y VMs. Roles requeridos: Project Lead, ML Engineer, Data Engineer, Pentester (Lab), DevOps/SRE, Legal/Ethics, QA/Reviewer.

Estructura del repositorio (esqueleto)

Recomendado: - /data: JSONL writeups, embeddings - /notebooks: experiments, RAG demo - /src: - /ingest: scrapers, n8n flows - /preproc: normalization, redact - /embed: embedding jobs - /api: FastAPI - /lab: docker-compose, ansible playbooks - /reports: jinja2 templates - /infra: terraform/vagrant - README.md, requirements.txt, Dockerfile, .github/workflows

Paso 1 — Preparación legal y ética

1) Redactar formato de consentimiento (PDF/JWT) que incluya: target (IP/CIDR), actividades permitidas, ventana temporal, firma del owner. 2) Aprobación por comité/asesor legal. 3) Definir política de acceso a PoC-lab (quién puede ejecutar PoC y cómo se almacenan).

Paso 2 — Ingesta y dataset (acciones concretas)

1) Crear lista de fuentes públicas: HTB retired, VulnHub, TryHackMe writeups, blogs, NVD/CVE. 2) Implementar scrapers (Python): - requests + BeautifulSoup para HTML - yt-dlp para subtítulos de YouTube 3) Guardar raw HTML/subs y transformar a JSONL con campos: id, fuente, fecha, titulo, so, servicios, cves, mitre, pasos_raw, pasos_struct (cmd_redacted), resumen. 4) Ejecutar jobs de n8n para orquestar scrapers (cron schedules).

Paso 3 — Preprocesamiento y anotación

1) Limpieza: normalizar unicode, eliminar headers/footers. 2) Anonimización: reemplazar IPs/credenciales por tokens <IP>, <CRED>. 3) Extraer bloques de código y comandos; guardar versiones redactadas para publicación y versiones internas completas SOLO en lab-protected storage. 4) Anotación con Label Studio: etiquetas MITRE, CVE, exploitability_label (confirmed/likely/unconfirmed).

Paso 4 — Embeddings & Indexación

1) Batch jobs con sentence-transformers (ej: all-MiniLM-L6-v2) para generar embeddings. 2) Indexar en FAISS/Chroma/Milvus y almacenar metadata en Postgres. 3) Versionado de índices y backups (S3).

Paso 5 — Desarrollo del Motor RAG

1) Configurar retriever que consulte Vector DB y recupere top-k fragments. 2) Definir prompts/templates para outputs: summary, mitre list, findings JSON. 3) Integrar LLM (OpenAI o Llama locally) via LangChain. 4) Implementar módulo PE-score (Python) que combine cvss, similarity, scanner flags, config exposure. Ajuste de pesos mediante validación.

Paso 6 — API y orquestación

1) FastAPI endpoints: - POST /authorize -> subir consentimiento (validate JWT) - POST /analyze-target -> body: {target, auth_token, scope} - POST /lab-validate -> dispara validación en lab (solo si target replicado) 2) Queue: Redis/RabbitMQ para tareas largas (indexing, lab jobs). 3) n8n flows para pipelines ETL y triggers nocturnos.

Paso 7 — Recolector pasivo y safe scans

1) Implementar passive collector: gather banners, certs, headers (requests, python-sslyze, custom parsers). 2) Safe scans (solo con consent): nmap -sV --script=banner --max-retries 1; OpenVAS configured non-intrusive; Nikto for config checks. 3) Guardar outputs raw en storage y hashes en Postgres for traceability.

Paso 8 — Orquestador LAB (validación PoC reproducible)

1) Diseñar docker-compose y/o Vagrant files para desplegar images VulnHub/DVWA/Metasploitable. 2) Playbooks Ansible (deploy -> snapshot -> instrument -> run tests -> collect evidence -> revert). 3) Instrumentación: instalar auditd/syslog collector, tcpdump, agente que sube logs firmados. 4) Scripts PoC INTERNAL (NO publicar), se ejecutan dentro del lab y guardan outputs firmados (hash).

Paso 9 — Generación de informes y remediación

1) findings.json: array de hallazgos con fields: id, service, port, evidence, possible_cves, priority, recommended_actions, verification_playbook, status. 2) Jinja2 templates -> generate PDF (technical + ethical). Include consent hash and evidence hashes. 3) Integración con Jira/ServiceNow: crear tickets via API with findings payload.

Paso 10 — Evaluación y validación

1) Dataset split: train/val/test for classification tasks (exploitability label). 2) Metrics: precision/recall/F1, ROC-AUC for PE-score, recall@k for retrieval. 3) Human eval: panel of pentesters reviews N=200 reports. 4) Calibration: tune PE-score weights to maximize ROC-AUC and minimize false positives.

Paso 11 — Despliegue y CI/CD

1) Dockerize services: api, retriever, worker, embeddings job. 2) GitHub Actions: test, lint, build images, deploy to staging. 3) Backups: periodic snapshot of vector DB, Postgres, S3. 4) Monitoring: Prometheus + Grafana + ELK for logs.

Paso 12 — Runbook operativo (checklist detallado)

Pre-run checks: - Confirm consent JWT valid and scope - Ensure target is in allowed CIDR - Ensure lab resources available Execution: - Trigger /analyze-target - Wait for passive collector and RAG output - If pe_score >= threshold and target republished to lab -> trigger /lab-validate - After lab run, retrieve evidence and generate reports Post-run: - Revert snapshots - Store evidence hashes - Create tickets and assign owners - Schedule verification task (post-remediation)

Seguridad y control de acceso

1) Access control: RBAC for repo and lab environment. 2) Logs: append-only storage, sign logs with system key (OpenSSL). 3) Secrets: use HashiCorp Vault or env secrets for keys. 4) Data retention policy: define retention and secure deletion for PoC data.

Cronograma detallado (sugerido)

Sem 1-2: Legal & design Sem 3-6: Ingest & dataset Sem 7-9: Anotación & embeddings Sem 10-12: RAG & API Sem 13-15: Lab infra & playbooks Sem 16-18: Lab validations (>=20) Sem 19-20: Evaluation & panel Sem 21-24: Write-up & defense

Anexos: ejemplos de comandos seguros y templates

Ejemplos (no-exploitativos): - nmap safe scan: nmap -sV --script=banner --max-retries 1 -oX output.xml <target> - Generate embedding job (python snippet): from sentence_transformers import SentenceTransformer - PE-score function skeleton: (see repo/snippets) Consent template fields: owner, target, scope, allowed_actions, valid_from, valid_to, signature

Contacto y próximos pasos

Si quieres, genero ahora: - docker-compose.yml base para lab - n8n workflow export JSON - Esqueleto FastAPI con endpoints - Notebook RAG demo Dime cuál quieres y lo preparo en el siguiente mensaje.