

Secure Scope AI: LLM-Assisted, Offline-First Vulnerability Triage for the Global South

Elvin Vidal Mollinedo Mencia — Cybersecurity Architect

Anette Alejandra Cruz Rojas — Backend / API

Jazid Andrey Ovando Durán — DevSecOps & Pipeline

Miguel Ángel Vega Moreno — Frontend Design

Hub Santa Cruz, Bolivia · Escuela Militar de Ingeniería (EMI). With: Apart Research · Y AIS · Cursor Bolivia · Schmidt Sciences.

Regional track: Latin America · Sub-track: AI Security (pipeline security and use of LLMs in DevSecOps).

Research conducted at The Global South AI Safety Hackathon, June 2026.

Abstract

Scanners such as Trivy and OWASP Dependency-Check produce large volumes of context-free findings that teams must review, prioritize, and fix manually; as a result, many vulnerabilities remain unremediated. We present Secure Scope AI, an agent that integrates natively into DevSecOps pipelines and works with any AI model—in the cloud or locally via Ollama—, preserving data sovereignty. Unlike traditional scanners, the agent does not detect vulnerabilities itself: a deterministic correlation engine consolidates, normalizes, and deduplicates findings before any inference, and only then does the model contextualize, prioritize, and propose remediations, safe versions, or compensating controls. We validated the platform with three reproducible pipelines over OWASP NodeGoat, which correlated 366 vulnerabilities per run (108 critical and 528 high accumulated) in ~3–4 minutes, with AI analysis in seconds; the bottleneck is SCA, not the AI. The system explicitly separates deterministic evidence from AI interpretation and degrades gracefully when no provider is configured, showing that it is feasible to integrate AI into DevSecOps in an auditable, offline-first way, especially in Global South settings with limited resources.

1. Introduction

Development teams rely on tools such as Trivy and OWASP Dependency-Check to detect vulnerabilities in source code, dependencies, and container images. However, these tools produce numerous findings that must be reviewed, prioritized, and fixed manually, which is time-consuming and requires specialized staff; as a result, many vulnerabilities go unremediated or are addressed late, increasing the attack surface.

We present **Secure Scope AI**, an AI-powered cybersecurity agent that integrates natively into DevSecOps pipelines and can connect to any AI model, in the cloud or run locally, preserving data sovereignty. The agent runs the analysis, correlates findings, prioritizes by risk, and generates actionable recommendations: it proposes dependency updates and simulates their sanitization without modifying the repository, and when a direct fix is not feasible, it recommends compensating controls and specific mitigations.

Our main contributions are:

1. A provider-agnostic, AI-based security agent that integrates with commercial and local models inside DevSecOps pipelines.
2. A deterministic correlation and prioritization engine for vulnerabilities that reduces manual effort and the risk of LLM hallucinations.
3. A remediation system that proposes safe updates, simulates sanitization, and generates compensating controls when a direct fix is not possible.

2. Related Work

Software Composition Analysis (SCA) and container-scanning tools such as Trivy and OWASP Dependency-Check are the state of the art for identifying vulnerabilities in dependencies and images, backed by CVE and GHSA identifiers and CVSS metrics. However, they are limited to detecting and reporting: correlating results, removing duplicates, prioritizing, and defining remediation remain the team's responsibility. LLMs are already used to assist with these tasks, but work on prompt injection, multilingual jailbreaks, and the OWASP Top 10 for LLM Applications shows that they can hallucinate or be manipulated through untrusted data; moreover, most commercial solutions depend on the cloud, which limits adoption where sovereignty or connectivity constraints exist.

Secure Scope AI closes this gap by combining consolidated scanners with an agent able to operate with multiple AI providers or local models. It does not replace the detection engines; rather, it turns their results into actions: it correlates findings from several sources, prioritizes by project context, recommends safe updates, simulates sanitization, and generates compensating controls when there is no direct fix. An organization would choose it over traditional solutions to move from a model of “vulnerability detection” to one of “AI-assisted remediation,” with contextual prioritization, verifiable recommendations, and the ability to operate entirely in local environments.

3. Methods

Secure Scope AI is a reproducible DevSecOps platform. The backend (Django, DRF, Celery, PostgreSQL, Redis) orchestrates the pipeline; the frontend (React) provides real-time monitoring and visualization; and an Nginx gateway centralizes access over HTTPS and isolates the internal services. Detection uses Trivy (container images) and OWASP Dependency-Check (SCA), executed in an isolated security-tools container. The core is a seven-stage pipeline —cloning, dependency build, Docker build, container run, dependency analysis, Trivy scan, and intelligent analysis—; in the last stage the agent correlates, deduplicates, prioritizes, and recommends. The agent is decoupled from the AI provider (commercial models or local via Ollama), and each stage produces auditable logs and telemetry, enabling reproducibility and traceability (Figure 1, Appendix).

3.1 Deterministic correlation and reliability

A central decision is that the AI does not detect vulnerabilities. Before any inference, a deterministic engine consolidates the findings from Trivy and Dependency-Check: it unifies identifiers (CVE, GHSA), normalizes severities, extracts CVSS scores, and removes duplicates; when both tools detect the same vulnerability, it is correlated and the highest CVSS is kept. The result is a single, validated list that acts as the source of truth for the rest of the pipeline, preventing the model from inventing or altering findings and reducing the risk of hallucinations, while improving traceability and auditability.

3.2 Multi-AI engine, sanitization, and security

The Multi-AI Engine supports DeepSeek (default), OpenAI, Gemini, Azure OpenAI, and local models via Ollama (Llama 3.2, Mistral, Code Llama), plus custom endpoints; in our runs it operated in live mode with DeepSeek (Figure 2, Appendix). To minimize hallucinations, the AI only analyzes already-validated vulnerabilities and uses structured responses, low randomness, and post-hoc validation, complementing each finding with references to OWASP, CWE, and NIST. A sanitization module assesses which libraries could be fixed and simulates the outcome (“Fixable simulated”), generating a patch only as a local ZIP without touching Git or the database (Figure 4, Appendix). The platform includes industry-aligned controls —JWT with rotation, Argon2, 2FA (TOTP), role-based access control, auditing, and API hardening—; by offline-first design, it can be deployed locally with Docker Compose, preloaded images, and vulnerability databases, using OWASP NodeGoat as the reference application.

4. Results

We ran real pipelines over OWASP NodeGoat (commit c5cb68a7) on the hub LAN, with the Multi-AI Engine in live mode using DeepSeek (Figure 2, Appendix). Table 1 summarizes the runs and Table 2

the severity breakdown; each pipeline consolidated 366 vulnerabilities, accumulating 1,107 across the three.

Table 1. Pipeline runs over NodeGoat.

Pipeline	Status	Duration	Vulnerabilities
PIPE-2026-000003	Completed	3m 47s	366
PIPE-2026-000004	Completed	2m 46s	366
PIPE-2026-000005	Completed	3m 48s	366

Table 2. Severity breakdown (per NodeGoat scan and accumulated across the 3 pipelines).

Severity	NodeGoat (per scan)	Accumulated (3 pipelines)
CRITICAL	36	108
HIGH	176	528
MEDIUM	—	377
LOW	—	94

Per-stage timing. In a complete live pipeline (PIPE-2026-000001, ~2m 26s): BUILDING 43 s, DOCKER_BUILD 2 s, CONTAINER_RUNNING 8 s, OWASP Dependency-Check 1m 14s, TRIVY 1 s, and Multi-AI with DeepSeek 17 s (Figure 1, Appendix). The bottleneck is Dependency-Check; the rest, including the AI analysis, runs in seconds, which validates operational feasibility.

Correlation. The engine marked as Correlated the findings present in both sources, keeping the highest CVSS, and consolidated 366 vulnerabilities per run (Figure 1, Appendix); e.g., NSWG-ECO-393 and GHSA-xc7v-wxcw-j472 (Memory Exposure in tunnel-agent@0.4.3, MEDIUM), CVE-2021-41182 (jquery-ui), and Arbitrary Code Execution in Handlebars (HIGH). The analyst additionally derives a per-CVE compensating-controls matrix (Figure 5, Appendix).

Separating evidence from interpretation. We distinguish two layers: (1) reproducible deterministic observations —the 366 vulnerabilities, their severity, and the per-CVE correlation come from Trivy + Dependency-Check, independently of the model—; and (2) the interpretive layer, generated live by DeepSeek, which produces the executive summary (Figure 3, Appendix) and the prioritized remediation plan (Figure 6, Appendix). The system maintains this separation explicitly and, if no provider is configured, degrades gracefully to a deterministic CVSS-based fallback without halting the pipeline. As a robustness check, the repository's QA suite reports 104 tests passed and 2 skipped (API contract, authentication, permissions, errors, and integration).

5. Discussion and Limitations

Secure Scope AI shows that AI can be integrated into DevSecOps without granting it absolute authority: LLMs are most useful as an analysis and recommendation layer over data already validated by deterministic tools, not as primary detectors. From this we derive three principles for AI systems in security: use verifiable sources of truth before inference, validate and structure the model's responses, and maintain controlled degradation when the AI is unavailable. The offline-first approach further broadens access for organizations with connectivity, budget, or sovereignty constraints. We observed that the main challenge is not inference but the volume of findings: correlation and risk-based prioritization were effective at reducing noise.

5.1 Limitations

The evaluation focused on OWASP NodeGoat, so results could vary on larger applications or other languages; although the runs already execute in live mode with DeepSeek, we did not quantitatively compare against human analysts nor formally measure the precision of the LLM's recommendations. Although we incorporated controls, we do not fully resolve threats such as prompt injection or incorrect recommendations, and we assume Trivy and Dependency-Check are a reliable baseline: their false positives or negatives would propagate to later stages. Finally, running untrusted repositories inside the pipeline requires more robust isolation for production.

5.2 Future Work

We plan to evaluate with multiple real applications, incorporate metrics for precision, usefulness, and hallucination rate, and compare against human experts. We will also explore defenses against prompt injection, automatic validation of recommendations with NVD and OSV, remediation via controlled patches, and stronger isolation plus support for new languages and tools.

6. Conclusion

Secure Scope AI offers a practical architecture for integrating AI into DevSecOps pipelines in a secure, reproducible, and auditable way. Instead of replacing traditional tools, it combines deterministic scanning, finding correlation, and AI-assisted reasoning to turn technical results into remediation actions. By relying on verifiable sources, structured validation, and local or cloud deployments, it constitutes a replicable model of secure AI, adaptable to organizations with diverse capabilities and especially useful in Global South settings.

Code and Data

- Code repository (app + how to run it): <https://github.com/elvinmollinedo/SecureScopeAI> — includes README.md with quick start: `cp .env.example .env && docker compose up -d --build` (access at <https://localhost>).
- Demo video: <https://drive.google.com/file/d/16SFqPBI6R6o8gfhg-DBqKoTR2msJ5ZI/view?usp=sharing>
- Test target: OWASP NodeGoat — <https://github.com/OWASP/NodeGoat>
- Tools: Trivy (Aqua Security), OWASP Dependency-Check 10.0.4, Multi-AI Engine (DeepSeek by default; OpenAI, Gemini, Azure OpenAI, Ollama, and compatible endpoints; deterministic fallback when no provider is set).
- Data: scan reports (Trivy / Dependency-Check JSON) and generated PDFs are produced locally by the pipeline; no external datasets are distributed.

Author Contributions

- Elvin Vidal Mollinedo Mencia — Cybersecurity architecture, threat model, design of the defense-in-depth methodology.
- Anette Alejandra Cruz Rojas — Backend / API (Django, models, scanning and correlation services).
- Jazid Andrey Ovando Durán — DevSecOps and pipeline (7-stage orchestration, tool integration, offline deployment).
- Miguel Ángel Vega Moreno — Frontend design and development (dashboards, live console, UX).

All authors contributed to writing and reviewed the manuscript.

References

1. Aqua Security. (2024). Trivy: Comprehensive Vulnerability Scanner. <https://github.com/aquasecurity/trivy>

2. OWASP. (2024). Dependency-Check. <https://owasp.org/www-project-dependency-check/>
3. OWASP. (2021). OWASP Top 10. <https://owasp.org/Top10/>
4. OWASP. (2023). OWASP Top 10 for Large Language Model Applications. <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
5. Greshake, K., et al. (2023). Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. arXiv:2302.12173. <https://arxiv.org/abs/2302.12173>
6. Deng, Y., et al. (2024). Multilingual Jailbreak Challenges in Large Language Models. ICLR 2024. <https://arxiv.org/abs/2310.06474>
7. Levy Yeyati, E. (2025). Latam-GPT and the Search for AI Sovereignty. Brookings Institution, November 25. <https://www.brookings.edu/articles/latam-gpt-and-the-search-for-ai-sovereignty/>
8. Lehdonvirta, V., Wú, B., & Hawkins, Z. (2024). Compute North vs. Compute South: The Uneven Possibilities of Compute-based AI Governance Around the Globe. Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, 7(1), 828–838. <https://doi.org/10.1609/aies.v7i1.31683>
9. OWASP. NodeGoat Project. <https://github.com/OWASP/NodeGoat>
10. FIRST. (2019). Common Vulnerability Scoring System (CVSS) v3.1. <https://www.first.org/cvss/>

Appendix

Figures are consolidated in this appendix; the body of the report references them by number.

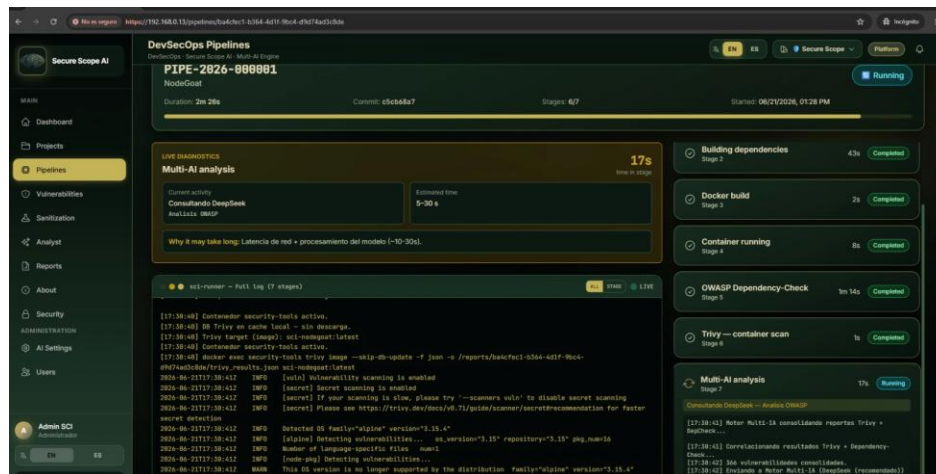


Figure 1. Seven-stage DevSecOps pipeline over NodeGoat (PIPE-2026-000001, commit c5cb68a7) with per-stage timing. During the analysis stage the Multi-AI Engine queries DeepSeek live; the log records the Trivy + Dependency-Check correlation and “366 consolidated vulnerabilities.”

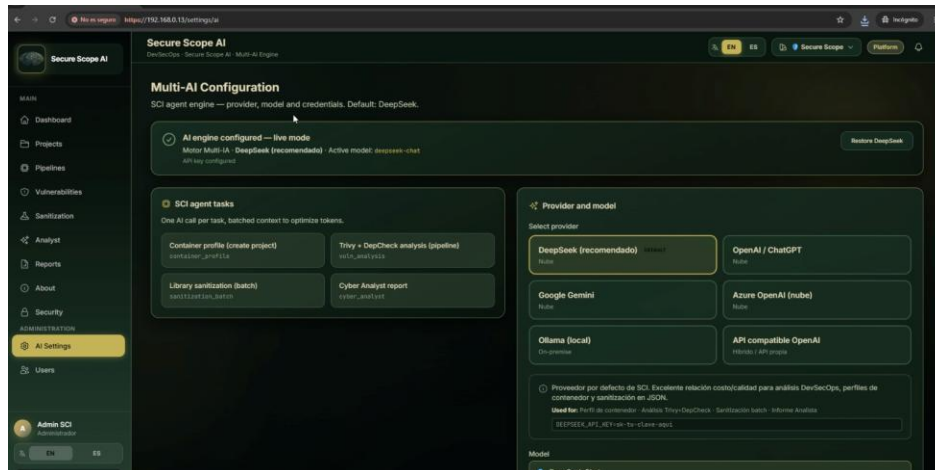


Figure 2. Multi-AI configuration in live mode with DeepSeek (deepseek-chat model, API key configured) and supported providers: OpenAI, Gemini, Azure OpenAI, Ollama (on-premise), and OpenAI-compatible endpoints. Each agent task uses one call per task.

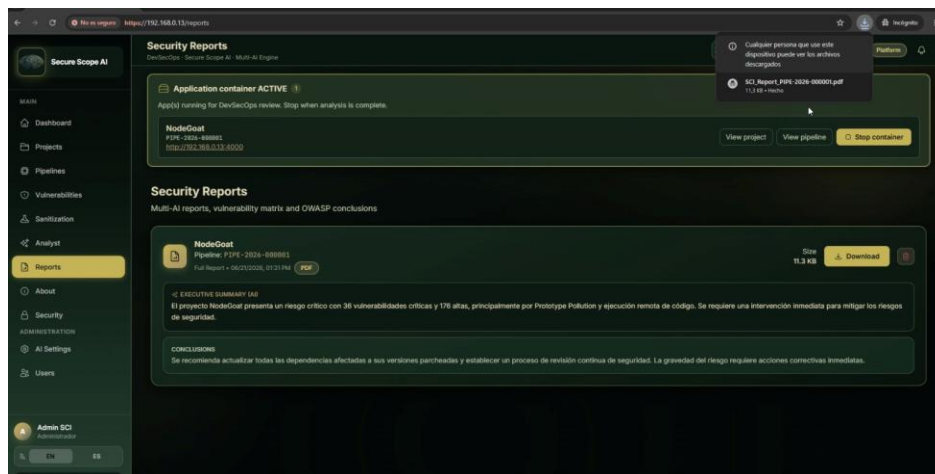


Figure 3. Security PDF report generated by the Multi-AI Engine live: the executive summary identifies 36 critical and 176 high vulnerabilities in NodeGoat, dominated by Prototype Pollution and remote code execution.

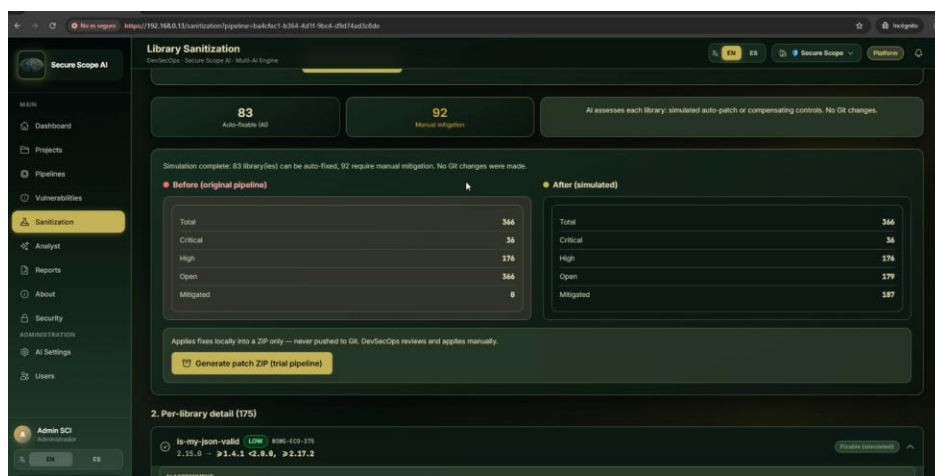


Figure 4. Sanitization module: before/after simulation over 366 vulnerabilities (36 critical, 176 high). The AI assesses 83 auto-fixable libraries and 92 requiring manual mitigation; the patch is generated only as a local ZIP, without modifying Git. This explicit separation between simulation and real application is consistent with AI-safety best practices.

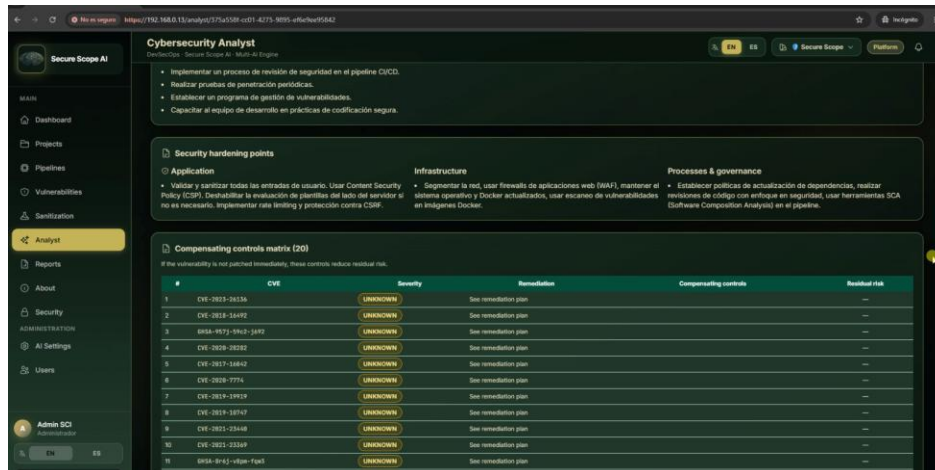


Figure 5. Cybersecurity Analyst: hardening points (application, infrastructure, processes and governance) and a per-CVE compensating-controls matrix with residual risk.

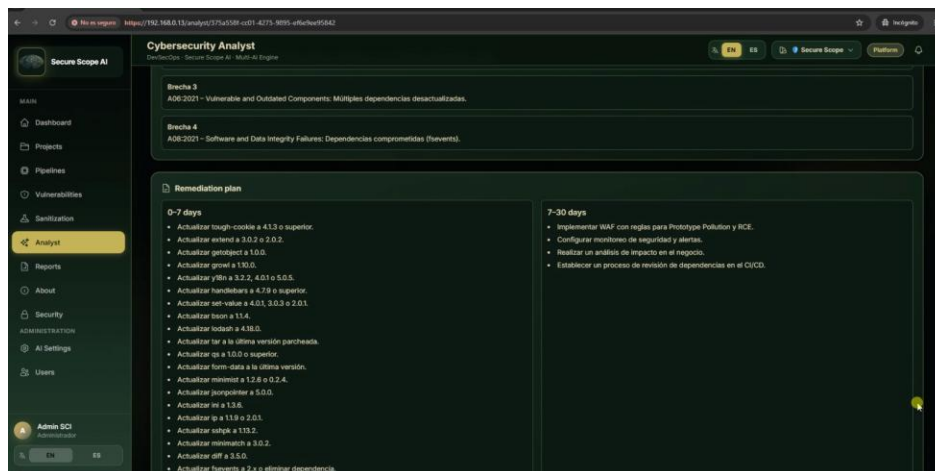


Figure 6. AI-generated remediation plan, prioritized by time window (0–7 days and 7–30 days), with target versions per dependency and infrastructure-level controls (e.g., WAF for Prototype Pollution and RCE).

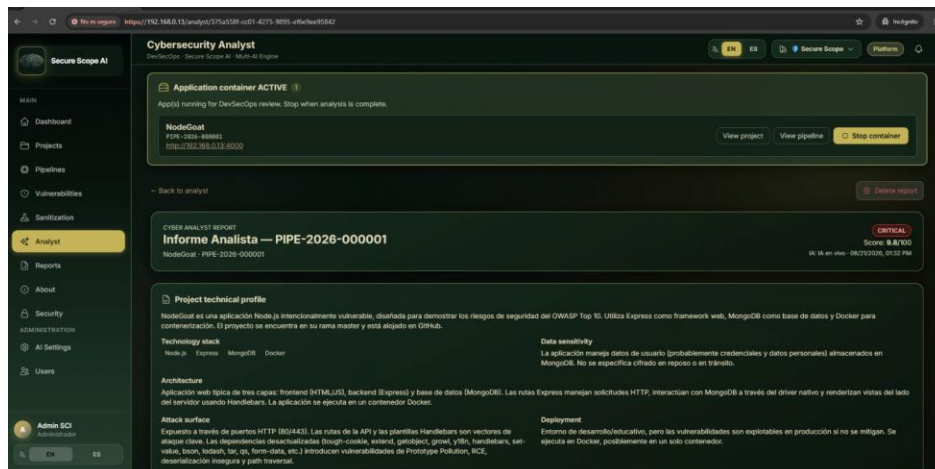


Figure 7. Analyst report (live AI) for PIPE-2026-000001: project technical profile, technology stack, three-tier architecture, and NodeGoat's attack surface.

Stack and versions (excerpt): Django 5.1.5, DRF 3.15.2, simplejwt 5.4.0, PostgreSQL 16, Redis 7, Celery 5.4.0, Argon2, React 19, Vite 6, Tailwind 3.4, Nginx 1.27, Trivy + OWASP Dependency-Check 10.0.4, WeasyPrint / xhtml2pdf for PDF. Approximate source size: ~15,000 lines.

Key files (for reproducibility): backend/apps/pipelines/tasks.py (orchestration),
backend/apps/scanning/services/correlation_engine.py (correlation),
backend/apps/scanning/services/deepseek_service.py (Multi-AI analysis),
backend/apps/scanning/services/library_sanitization_service.py (sanitization simulation),
backend/apps/analyst/services/cyber_analyst_service.py (CISO report),
backend/apps/ai/provider_registry.py (provider abstraction), backend/common/middleware.py (API
hardening). The repository also includes an alternative FastAPI orchestrator in MVP state
(fastapi_orchestrator/), independent of the main Django + Celery flow.

LLM Usage Statement

During the development of Secure Scope AI, several LLM-based assistants were used to support specific project tasks. ChatGPT was used for overall architecture planning, component definition, and structuring the development flow. Antigravity supported backend implementation, task automation, and the generation of supporting code. DeepSeek was used to design and implement the intelligent agent's API and some components of the analysis engine. Cursor served as a programming assistant for code review, technical documentation, refactoring, and validation of architecture decisions.

The AI models were used solely as development-assistance tools. All design, integration, implementation, testing, and final validation decisions were made by the authors. Likewise, all results, metrics, security analyses, and claims presented in this work were manually reviewed and verified by the team before being included in the project and in this report.