

NEOps

DevSecOps for the AI Era

<https://github.com/hthecreator/broad-bros>

Team Broad Bros

Hackathon Submission

Spyridon Georgakopoulos – georgako@outlook.com

Hans A. Gunnoo – hans.gunnoo@faculty.ai

Callum Pender – callum.pender@faculty.ai

November 23, 2025

Abstract

NEOps is a Command Line Interface (CLI) tool informed by guidance from the OWASP Top 10 LLM Vulnerabilities and AI Labs that embeds AI safety into your product lifecycle from day one. While traditional security tools catch SQL injections, they miss unique AI threats like unsafe model usage, deprecated dependencies, and improper LLM output handling. NEOps fills this gap by combining static analysis with **AI Agent reasoning (OpenAI o3)** to scan code, architecture, and documentation, integrated directly into CI/CD pipelines via `pyproject.toml`.

1 Problem Statement: The AI Safety Gap

In modern software development, cybersecurity checks are routine. However, **AI safety often comes later—or not at all**. As engineering teams rush to integrate Large Language Models (LLMs), they introduce new risks:

- **Improper Output Handling:** Executing or rendering raw LLM outputs without sanitization, leading to injection attacks.
- **Unsafe Dependencies:** Using models that have been deprecated or flagged as dangerous by providers.
- **Missing Guardrails:** Failing to include safety identifiers in API requests.

Currently, there is no “DevSecOps” standard for these AI-specific problems. NEOps fills that gap.

2 Key Differentiators (Judging Criteria)

Defensibility & Acceleration (AI Risk): NEOps strengthens the product shield by embedding automated AI safety checks directly into developer workflows. It shifts safety “left,” allowing teams to catch risks like **OWASP-001** (Unsanitized Execution) before deployment.

Functionality (Does It Work?): It is a fully functional CLI tool that hooks into CI/CD pipelines. Unlike simple regex linters, NEOps uses **AI Agents** to understand context, reducing false positives and identifying complex vulnerabilities.

Scalability: With language-agnostic configuration, extendable rule sets, and Docker support, NEOps scales across teams and tech stacks. It generates machine-readable (JSON) and human-readable (Markdown) reports for seamless integration.

3 Installation & Usage

NEOps is designed to live where developers live: in the terminal.

3.1 Installation

```
# Install via pip
pip install neops
```

3.2 Running a Scan

NEOps automatically detects git-tracked files. To scan a repository for AI safety violations:

```
# Basic scan (defaults to current directory)
neops scan

# Scan with custom config path
neops -p /path/to/pyproject.toml scan
```

The tool will output a summary of errors, warnings, and info logs, saving detailed results to timestamped JSON and Markdown files.

4 Detection Capabilities

NEOps rules are curated from trusted sources, including **OWASP**, **OpenAI**, and **Anthropic**.

Severity	Code	Description
ERROR	OWASP-001	LLM output executed without sanitization. Detects <code>exec()</code> , <code>eval()</code> , or <code>system</code> calls on raw AI output.
ERROR	OWASP-003	Injection Risk. Detects LLM output used in DB queries or file paths without parameterization.
ERROR	NEOps-002	Dangerous Provider. Flags usage of model providers classified as dangerous.
WARN	OWASP-002	Unsafe Rendering. Detects LLM output rendered to HTML without encoding.
WARN	NEOps-004	Legacy Model. Detects models scheduled for deprecation (e.g., <code>claude-3-5-haiku-20241022</code>).

5 Configuration-as-Code

NEOps requires no proprietary UI setup. Configuration is managed via standard files, allowing teams to enable/disable rules or change severity levels to fit their risk appetite.

Example Configuration (pyproject.toml)

```
[tool.neops.rules]
# Override severity for specific rules
"OWASP-001" = { severity = "warning" }
```

```
# Disable rules that don't apply to your project
"OWASP-002" = { enabled = false }

# Enforce stricter checks on legacy models
"NEOps-003" = { severity = "error" }
```

6 Technical Architecture

The scanning engine operates on two levels:

1. **Static Analysis:** Fast scanning of configuration files and code structure to identify deprecated models and missing safety headers.
2. **AI Agent Analysis:** Utilizing **OpenAI o3**, NEOps analyzes code snippets in context to determine if data flows are actually dangerous or if sanitization logic exists nearby.

7 Conclusion

NEOps is not just a linter; it is an enabler for safe AI innovation. By treating AI safety with the same rigor as traditional DevOps, we allow engineers to build faster, knowing that the guardrails are automated, rigorous, and always on.