AI Security Agent for EO Projects

Automating Security at Every Stage of the Earth Observation Software Lifecycle

Federico Fornari
ECMWF Computing Department, Bologna
ISGC 2026 · 18 March 2026

ECMWF

Context & Motivation

The Projects' Platforms

- EU-funded research platforms with distributed cloud infrastructures
- Kubernetes clusters on ECMWF CCI cloud infrastructure
- GitLab CI/CD orchestrated by Dagger and FluxCD
- Fast-moving, multi-partner development environments

GitLab Dagger FluxCD

The Security Challenge

- Container images accumulate CVEs silently between builds
- Manual remediation backlogs grow faster than teams can clear
- Test coverage stagnates — edge cases never get written
- Static analysis can't tell exploitable from theoretical CVEs

What if AI could fix all of this automatically, inline with the pipeline, before any human reviews the output?

What is the AI Security Agent?

A fully automated, LLM-powered security platform integrated into GitLab CI/CD

Automation First

Every security action that can be automated, is. Zero human intervention required.

No External Dependencies

All LLM inference runs on-premise via Ollama. Source code and CVE data never leave the organisation.

Iterative Improvement

Fixes are verified and retried. Failed attempts are blacklisted to prevent repetition.

1 Container Hardening

Trivy + LLM

2 Source Code Remediation

SonarQube + LLM

3 Test Enhancement

LLM (opt-in)

4 Penetration Testing

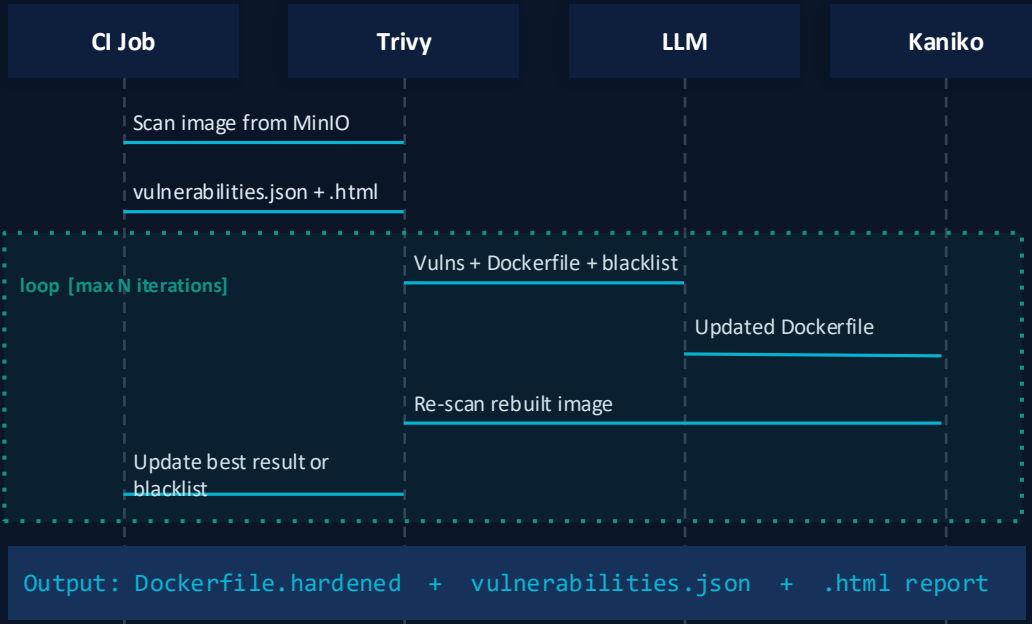
K3s + PentestGPT

Languages: .NET · Python · Java · Node.js *+ any language via YAML*

The CI/CD Pipeline in Practice

`detect_language`	Identify app language, set path variables	
`sonar_scan`	SonarQube static analysis	SonarQube
`build_image`	Build & store container image in MinIO	Kaniko, MinIO
`scan_and_fix`	Trivy scans CVEs — LLM hardens iteratively (up to 5×)	Trivy, Ollama
`fix_source_code`	LLM fixes SonarQube BLOCKER/CRITICAL issues	SonarQube, Ollama
`enhance_tests`	Generate unit tests — opt-in via commit message	Ollama
`integration_test`	Deploy to ephemeral K3s cluster, run PentestGPT	k3s, PentestGPT
`push_image`	Push hardened image to GitLab registry	GitLab
`cleanup`	Tear down K3s cluster, remove temp data	

Capability 1 — Container Hardening



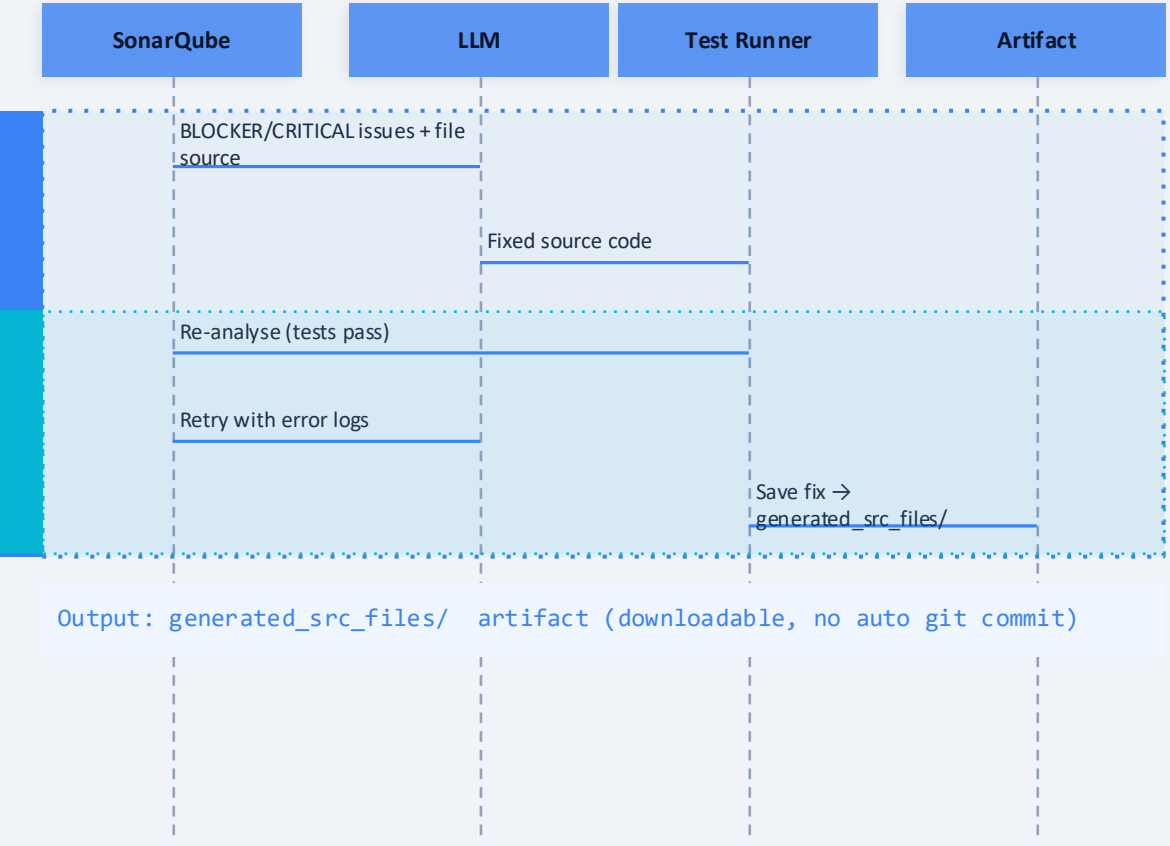
420	12	97%
Before	**After**	**Saved**
19 CRIT + 401 HIGH	iteration 5 / 5	no human input

LLM Dockerfile strategies:

- Upgrade base image to patched version
- Pin / upgrade vulnerable system libraries
- Add USER directive — non-root execution
- Remove unnecessary tools & privileges

Tools: Kaniko, Trivy, Ollama

Capability 2 — Source Code Remediation



What the LLM fixes:

Security Vulnerabilities

SQL injection, XSS, insecure deserialisation, hardcoded credentials

Bugs

Null dereference, resource leaks, index out-of-bounds

Code Smells

Excessive complexity, unused imports, duplicated logic

```
SQL Injection — python:S3649
BEFORE: f"SELECT…WHERE id={user_id}"
AFTER:  execute("…id=?", (uid,))
```

14 BLOCKERs → 0

SonarQube

Capability 3 — Test Enhancement

Bridging the coverage gap — automatically

Opt-in trigger: commit message contains 'enhance unit tests'

Targets: happy path gaps, edge cases, error conditions

Safety guarantee: generated tests must compile and pass before accepted

Retry loop: failures feed error logs back to the LLM for a retry

Output: generated_test_files/ artifact — no automatic git commit

Supported Frameworks

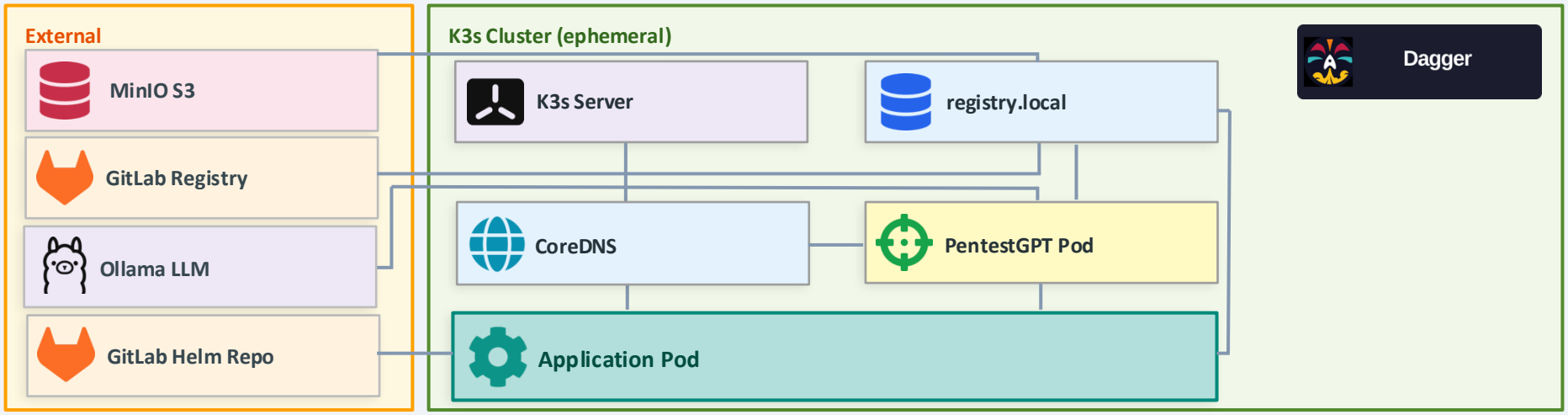
xUnit, NUnit, MSTest	.NET
pytest	Python
JUnit 5	Java
Jest, Mocha	Node.js

36% → 42% (+6 pts)

Languages: .NET · Python · Java · Node.js

Capability 4 — Automated Penetration Testing

Knowing a CVE exists ≠ knowing it's exploitable in your deployment. PentestGPT finds out.



Stack: k3s | Helm | Ollama | GitLab | MinIO | PentestGPT

12 CVEs Assessed — Real Production Application

2 Class A 17% Exploitable	7 Class B — 58% — Present, not exploitable in this config	3 Class C 25% N/A

Team focused on 2 issues not 12 — other 10 adequately mitigated and documented.

Technology Stack

Dagger

Pipeline as Code

Python CI/CD framework. Identical on laptops and GitLab. Content-addressable caching skips unchanged stages.

dagger.io

Ollama

On-Premise LLM Inference

Self-hosted on ECMWF CCI GPU (NVIDIA A100). qwen3-coder:30b for patches, gpt-oss:20b for analysis. No data leaves the org.

ollama.com

MinIO

Container Registry

S3-compatible on-premise object store. No Docker Hub rate limits. Full retention control. Images stay inside the network.

min.io

k3s

Ephemeral Kubernetes

Lightweight cluster for integration testing and pentesting. Fresh per pipeline run, torn down after.

k3s.io

Vault

Secret Management

All credentials retrieved at runtime via AppRole. Nothing stored in GitLab CI/CD variables or committed to the repo.

vaultproject.io

PentestGPT

AI Penetration Testing

Combines LLM reasoning with real pentest tools to classify CVEs by actual exploitability in the deployed app.

github.com/GreyDGL/PentestGPT

Multi-Language by Design

Fully automatic language detection

The agent inspects the repository for language markers and selects the appropriate toolchain. Set TARGET_LANGUAGE to override.

.NET/C#	`.csproj / .sln`
Java	`pom.xml / build.gradle`
Node.js	`package.json`
Python	`*.py files`

Adding a new language — one YAML block, no code changes:

```
go: { markers: ["go.mod"],
      sdk_image: "golang:1.21-alpine",
      test_command: "go test ./..." }
```

Per language, the agent configures:

- **SDK / base image**
  e.g. mcr.microsoft.com/dotnet/aspnet:8.0
- **Build command**
  dotnet build / mvn package / npm install
- **Test runner & command**
  dotnet test / pytest / mvn test / jest
- **SonarQube scanner**
  dotnet-sonarscanner / sonar-scanner
- **Dockerfile strategies**
  Language-specific hardening approaches

Real-World Results

Real-world .NET 6 Application

Container CVEs	SonarQube BLOCKERs	Test Coverage	Remediation Time
420	14	36%	3-5 Days
→ 12	→ 0	→ 42%	→ 360 min
97% reduction	100% resolved	+6 pts	~12-20× faster

PentestGPT — 12 CVEs Assessed

2 Class A 17% Exploitable	7 Class B — 58% — Present, not exploitable	3 Class C 25% N/A

Focused on 2 issues not 12 — evidence the other 10 were adequately mitigated.

Summary & What's Next

Four Capabilities — One Automated Pipeline

1 Container Hardening

Trivy + LLM → 97% CVE reduction

2 Source Code Remediation

SonarQube + LLM → 100% BLOCKERs resolved

3 Test Enhancement

LLM test generation → +6% coverage

4 Penetration Testing

K3s + PentestGPT → exploitability classification

Future Directions

Extend language coverage

Add Go, Rust, and other languages via the YAML config — no code changes

Policy-driven hardening

Auto-enforce CIS Benchmarks and OWASP Top 10 as mandatory quality gates

Runtime threat detection

Extend from static analysis to live log monitoring and anomaly detection

Feedback loop to developers

Auto-open GitLab issues with LLM-generated explanations for each finding

Thank You

Questions & Discussion

Federico Fornari
ECMWF Computing Department, Bologna
federico.fornari@ecmwf.int

ECMWF

Team

- Mohanad Albughdadi
- Marica Antonacci
- Vasileios Baousis
- Federico Fornari
- Tolga Kaprol
- Claudio Pisa

EFP Cloud System Team
ECMWF Computing Department