COLOSSUS

Recurring cyberdefence from a GitHub repository

Hackathon DefAcc 2026 | Track 3: Automated cyberdefence | COLOSSUS team (four contributors); lead: Daniel

Repository and run instructions: github.com/YamidGT/COLOSSUS

Abstract

AI can accelerate vulnerability discovery, but an autonomous defender can also ship a broken patch or follow hostile instructions. COLOSSUS accepts a GitHub URL, investigates an isolated copy, generates repairs with GLM, and gates lab deployment on recorded tests and applicable proofs. A fresh DSVW run completed repair, lab promotion and mirror refresh in 330.7 seconds, including five minutes of health observation.

1. The threat

Faster vulnerability discovery is useful only if defenders can safely remediate what they find. An automated patch may break legitimate behavior; a repository comment or tool response may redirect the agent. Developers need an auditable path from evidence to a running repair, with independent checks and recovery. COLOSSUS treats repository content and model proposals as untrusted inputs, consistent with agent-security research [1].

2. What we built

An operator enrolls a repository through a Typer CLI or React/Tailwind action center. The controller pins source, builds a supported Python application in gVisor, and records readiness and dependencies. Scheduled reviews and OSV/CISA intelligence feed bounded investigations. GLM, accessed through Mistral, proposes patches; deterministic code controls edits, budgets and deployment. Every repair requires protected security and legitimate-behavior tests. For the registered policy fixture, GLM proposes implementations inside a frozen Dafny template, receives verifier diagnostics and retries within a fixed budget [2]. Unknown or unsupported proof falls back to mandatory testing under the tested policy; required proofs still block deployment. Compiled components require artifact and wrapper checks. Accepted artifacts undergo lab observation, promotion and mirror refresh. A retained previous image supports recovery. Credentials remain outside target containers.

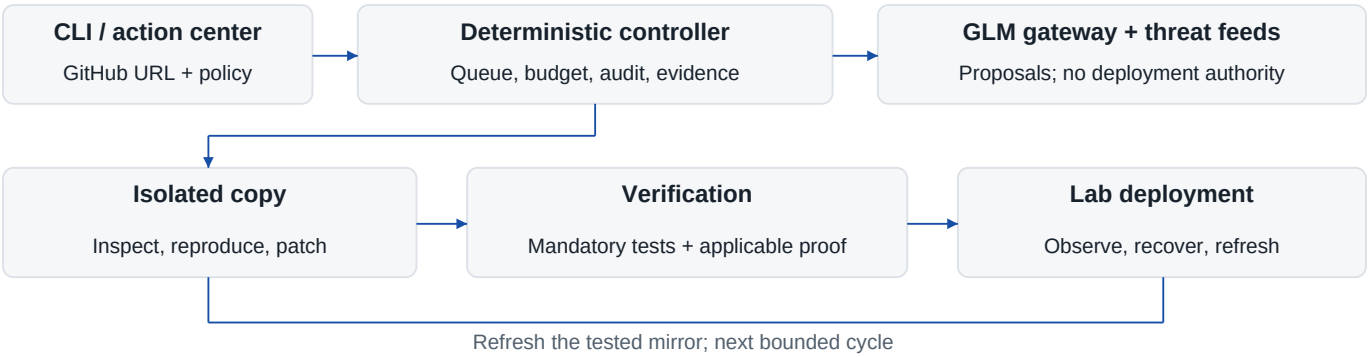


Figure 1. Models propose repairs and proof candidates; deterministic gates control isolated execution and lab deployment.

Evaluation and results

3. How we evaluated

We evaluated two fresh runs of one DSVW SQL-query repair through protected tests, deployment, recurrence and rollback. An owned four-method policy fixture exercises proof, compilation and wrong implementations. A 24-repository compatibility pilot accepted 20 archives; five additional runtime attempts yielded two assisted starts and three failures. GLM triaged Python snippets in 11 repositories; judgments remain unconfirmed. Metrics are scope, completion, latency and cost, not security accuracy. CTIBench, CyberSecEval and DafnyBench remain unrun [3-5].

4. Results

Measurement	Scope / baseline	Observed result
DSVW repair checks	One original vs patched query	7 required gates passed; formal unsupported
Fresh full cycle	1 new call; 300 s observation	330.7 s; $0.001287
Fresh repetition	2 calls; first candidate blocked	341.0 s; $0.002657
GLM formal component	Owned: 4 methods; 39 wrapper checks	5.50 s; 1 call; $0.000078
Incorrect controls	4 policy mutants + wrapper bypass	All rejected; substituted files rejected
GLM signal triage	11 repos; unconfirmed judgments	57 / 59 returned; $0.004014
Application smoke tests	VAmPI modified pin; PyGoat sub-lab	6/6 + 7/7 HTTP checks; no repair
Repository campaign	24 requested; mixed coverage	20 / 24 archives admitted

Table 1. Development and component-calibration results; no overall security score or benchmark certification.

Two fresh DSVW cycles completed deployment and mirror refresh. In the repetition, the security gate blocked the first generated patch; GLM used failure feedback and the second candidate passed. Both next cycles returned NO_CHANGE. Component proofs remain separate; earlier provider errors and narrow scope prevent general reliability claims.

5. Limits and dual-use risk

This is a constrained lab prototype, not a production security guarantee. Generic repair coverage and stack compatibility remain limited. Proofs cover named components under fixed contracts and toolchain assumptions; they do not prove an entire Python application secure. Public training repositories and small calibration samples limit generalization. Detection and patch generation are dual-use: execution is confined to enrolled local copies with quotas, restricted networking and no model-held deployment credentials. No live public target or cloud attack scenario was exercised.

References

[1] Debenedetti et al. (2024). AgentDojo. NeurIPS. Source

[2] Dafny contributors. Dafny Reference Manual. Source

[3] Alam et al. (2024). CTIBench. NeurIPS. Source

[4] Bhatt et al. (2023). Purple Llama CyberSecEval. Source

[5] Loughridge et al. (2024). DafnyBench. Source