

AuraRemed: Autonomous Security Engineering Report

A Two-Tier Hybrid Architecture for Cognitive Code Self-Remediation and Operational Verification

Author: Santiago Rodríguez Heras

Target Core: Local Batch Run Matrix

Date: May 24, 2026

1. Executive Summary

Modern Continuous Integration and Continuous Deployment (CI/CD) workflows are constantly bottlenecked by manual triage of static security findings. Traditional Static Application Security Testing (SAST) utilities excel at deterministic pattern recognition but lack structural synthesis capabilities to safely mitigate findings without operator intervention.

AuraRemed addresses this paradigm by introducing an automated, multi-loop architectural pipeline that joins rule-based static validation with advanced localized language modeling. The platform acts as an autonomous feedback loop, analyzing software vulnerabilities, generating surgical in-memory patches, enforcing rigorous code documentation practices, and self-validating changes against the original security baseline.

 **Key Achievement:** Completed full mitigation testing against a multi-vector threat matrix. 100% of discovered high-criticality vulnerabilities were remediated autonomously, achieving absolute compliance and clean validation exits (Code 0).

2. System Architecture & The Remediation Loop

The framework minimizes reliance on external clouds or third-party APIs to preserve complete privacy over intellectual property and prevent source code exfiltration risks. The design operates on a dual-tier control mechanism:

- Tier 2 (Deterministic Verification):** Utilizes an optimized local instance of Semgrep. This engine enforces compliance against extensive semantic patterns to guarantee syntactic and stylistic determinism.
- Tier 1 (Cognitive Remediation):** Leverages a locally orchestrated qwen2.5-coder large language model via the Ollama runtime layer. This tier assesses the abstract syntax tree and structural context surrounding defects.

The Feedback Synchronization Formula

The iterative lifecycle operates as a stateful machine bounded by an execution limit constraint to mitigate stasis and infinity limits. Let the target file state at iteration i be denoted as S_i , and the evaluation parsing engine output be $V(S_i)$. The next transition state is defined by the following expression:

$$S_{i+1} = \Psi(S_i, V(S_i))$$

Where Ψ represents the surgical patch injection function orchestrated by the cognitive layer. If at any state $V(S_i) = \emptyset$ (representing zero flagged vulnerabilities), the environment declares stability, stops the sequence, and exits safely with execution value 0.

3. Threat Test Bench Breakdown

To evaluate operational capabilities, AuraRemed was stress-tested against a repository containing five distinct high-risk vulnerabilities mapped to the OWASP Top 10 classifications:

Target Script Module	Vulnerability Vector Detected	Exploitation & Business Risk Impact	Remediation Mechanism Applied
vuln_sql_injection.py	SQL Injection (CWE-89)	Arbitrary database read/write, authentication bypass, data extraction.	Parameterized inputs utilizing secure tuple object binding definitions.
vuln_command_injection.py	OS Command Injection (CWE-78)	Remote Code Execution (RCE) via shell concatenation vectors.	Refactored runtime execution to pass arguments explicitly via isolated lists.
vuln_path_traversal.py	Path Traversal (CWE-22)	Unauthorized access to sensitive server infrastructure layout/files.	Sanitized inputs using strict basename filtering and path normalization rules.
vuln_insecure_pickle.py	Insecure Deserialization (CWE-502)	Arbitrary payload execution during binary serialization routines.	Migrated object handling infrastructure to standard deterministic JSON parsing layers.
vuln_hardcoded_jwt.py	Exposed Cryptographic Token (CWE-798)	Global authentication spoofing and complete signature compromise.	Decoupled parameters to fetch signing keys from secure <code>os.environ</code> spaces.

4. Code Hardening & Auto-Documentation Quality

A unique aspect of the AuraRemed orchestrator is its strict formatting wrapper constraint. Instead of merely patching the logical flaw, the cognitive tier is programmed to inject an immutable **Security Mitigation Report** directly inside the source code as a standardized Python docstring.

This guarantees that software developers receive complete contextual visibility regarding what mitigation occurred without having to leave their integrated development environment (IDE). Below is a verified output block demonstrating the automated injection behavior:

```
def query_user_profile(user_input_id):  
    """  
    [AuraRemed Security Mitigation Report]  
    - Vulnerability Detected: SQL Injection  
    - Exploit Risk: Raw string interpolation enables severe data exfiltration or structural  
    bypass.  
    - Remediation Applied: Replaced unsafe string formatting with parameterized SQL query  
    binding.  
    """  
    conn = sqlite3.connect('users.db')  
    query = "SELECT * FROM profiles WHERE id = ?"  
    return conn.execute(query, (user_input_id,)).fetchall()
```

5. Empirical Performance Analytics

The batch processing execution run yielded definitive results across all test targets. For every file, the initial vulnerability metrics captured during Loop 1 triggered the cognitive remediation pipeline. By Loop 2, subsequent validation scans confirmed that 100% of targeted modules achieved complete compliance.

The convergence rate demonstrates optimal efficiency, proving that the localized model is highly deterministic when guided by granular context strings and specialized system personas.

6. Fellowship Strategic Conclusion

AuraRemed successfully bridges the critical disconnect between automated defensive detection and active remediation. By proving that a local, lightweight cognitive model can surgically harden multi-vector flaws—while preserving original software dependencies, runtime business code logic, and interface frameworks intact—this research sets a baseline blueprint for fully self-healing software supply chains. The empirical metrics support its immediate viability for production-level pre-commit pipelines and secure DevSecOps agent orchestration.