Verification and Validation Assignment

Team 6
Apostolos Cavounis
Yu-Kuang Chen (Eric)
Si-Ci Chou (Simon)

Objectives

1. Verify End-to-End Scenario Execution Stability (FR-1 / CAP-1)
2. Verify LLM Attack Plan Generation Quality (FR-2 / CAP-1)
3. Verify RAG Retrieval Latency and Relevance (FR-3 / CAP-2)
4. Verify Dashboard Report Completeness & Export Quality (FR-4 / CAP-3)
5. Verify Dashboard Performance Under Load (NFR-1 / CAP-3)

Prioritization

Product Capability	**Priority (High, Medium, Low)**	**Brief Justification**
CAP-1: LLM-Driven Scenario Generation & Execution (FR-1, FR-2)	High	Core function of the system. Required for MVP demonstrations and validates the fundamental goal of automated red-team scenario generation. Both FR-1 and FR-2 are marked *Must Have* in RVTM.
CAP-3: Dashboard Reporting & Visualization (FR-4)	High	Essential for communicating simulation results and providing PoC steps, remediation, and severity. Marked *Must Have* in RVTM, and critical for evaluator understanding.
CAP-4: Human Approval & Safety Controls (FR-5, NFR-2)	High	Prevents unsafe actions, ensures legal/ethical compliance, and protects remote systems. Both FR-5 and NFR-2 are flagged *Must Have* in RVTM.
CAP-2: RAG Ingestion & Retrieval (FR-3)	Medium	Improves LLM output quality but is marked *Should Have*. Important to accuracy, but not required for MVP to function.

CAP-3 Performance Requirement (NFR-1)	Medium	Dashboard performance impacts user experience, but minor delays do not prevent system operation. Marked *Should Have* in RVTM.

Full RVTM

Requirements				Test			
Project Capability	Req. ID	Requirement (Shall Statement)	Supporting Context	Test Case ID	Test Description	Impacted Components	Additional Comments
CAP-1 LLM-driven scenario generation & execution	FR-1	Creating and executing an authorized red-team scenario against a user-supplied target.	Core mission: reliable, affordable, safe scenario execution; human approval prevents unsafe actions.	TC-01	Run 10 scenarios across prepared testbeds; verify timing ≤30 min for medium complexity; confirm reports and approval gates.	LLM Engine, Scenario Executor, Dashboard	Priority: Must Have
CAP-1 LLM-driven scenario generation & execution	FR-2	Support generating red-team attack plans using a locally hosted LLM based on user-supplied target and context.	LLM hosted via Ollama; inputs include target URL, known vulnerabilities.	TC-02	Run 10 simulations; verify actionable steps (≥3) and alignment with OWASP Top 10 in ≥80% cases.	LLM Engine, RAG Pipeline	Priority: Must Have
CAP-2 RAG ingestion & contextual retrieval	FR-3	The system shall ingest user-supplied documents into a vector index and retrieve relevant context for LLM queries within sub-second latency.	Uses FAISS or similar; supports PDF, TXT, Markdown.	TC-03	Upload 20 docs; measure retrieval latency (<1s) and relevance (≥90%).	RAG Pipeline, Vector DB	Priority: Should Have
CAP-3 Dashboard reporting	FR-4	The system shall present scan results and attack simulations in a web-based dashboard, including vulnerability details, severity ratings, and reproducible PoC steps.	Dashboard supports interactive filtering and export (PDF/JSON).	TC-04	Run 5 simulations; verify report includes vulnerability name, CVSS severity, PoC steps, remediation guidance; confirm export works.	Dashboard UI, Backend API	Priority: Must Have
CAP-3 Dashboard reporting	NFR-1	The system shall return dashboard query results in under 2 seconds under normal load conditions.	Normal load = 10 concurrent users.	TC-05	Simulate 50 queries; measure response time; ≥95% complete within 2s.	Dashboard UI, Backend API	Priority: Should Have

CAP-4 Human approval & safety controls	FR-5	The system shall require explicit user approval before executing any high-impact command that could modify remote state or affect system availability.	High-impact = data modification, DoS tests, etc.	TC-06	Run 10 simulations with high-risk commands; verify system blocks execution and prompts approval in 100% cases.	Approval Workflow, Audit Log	Priority: Must Have
CAP-4 Human approval & safety controls	NFR-2	The system shall retain audit logs for at least 90 days and prevent unauthorized tampering or deletion.	Logs stored append-only with cryptographic checks.	TC-07	Perform 20 actions; verify logs recorded; attempt unauthorized modification and confirm block.	Audit Log, Security Layer	Priority: Must Have

Test Approach

Category	Description	Manual or Automated
Unit Testing	Test individual components such as LLM prompt templates, safety-rule functions, RAG retrieval functions, and dashboard API endpoints. Ensures each module behaves correctly in isolation.	Mostly Automated
Integration Testing	Validate interactions between LLM Planner to RAG, Backend to Dashboard, confirms connected modules work together.	Mixed
System Testing	End-to-end testing of full workflow. Checks correctness, timing, and safety enforcement.	Manual
Non-Functional Testing	Test performance (RAG <1s, dashboard <2s), audit-log integrity, safety enforcement, and scalability under load.	Mixed
Regression Testing	Re-run automated unit and integration tests after each major change, plus a weekly manual full-pipeline scenario to ensure no regressions.	Mixed

Validation Plan

Our validation strategy focuses on confirming that the AI RedTeam Assistant effectively aids security teams or small businesses in identifying vulnerabilities and provides actionable remediation steps that developers can understand. We will utilize a combination of User Acceptance Testing (UAT) and Effectiveness Benchmarking to make sure that the system solves the core problem.

1. User Acceptance Testing (UAT): To validate the system in realistic conditions, we will conduct a "Blind Test." A team member acting as the security analyst will be tasked with identifying critical vulnerabilities on an isolated sandbox using only the AI RedTeam dashboard, with zero prior knowledge of the target.

- Success Metrics: The analyst must exploit at least 3 vulnerabilities within a 1-hour session and rate the "Human-in-the-loop" approval workflow > 4/5 on a usability scale.

2. Usability & Performance Evaluation: We will validate the "Remediation Loop" by handing a PDF report generated by the system to a developer who did not participate in the attack. They must attempt to patch the vulnerability using only the report's guidance. Additionally, we will benchmark the Hallucination Rate by running the tool against a hardened server to ensure it does not suggest impossible attacks.

- Success Metrics: The developer successfully patches the issue without external help, and the system maintains a False Positive Rate < 20% on hardened targets.

Tools, Environments & Test Data Sets

We will utilize the following resources to ensure verification and validation while maintaining safety and reproducibility.

Testing Tools

- Pytest: Configured for automated unit and integration testing of the Python backend, specifically for the RAG retrieval logic and safety-check functions.
- Playwright: Used for end-to-end testing of the Dashboard (CAP-3) to automatically verify report generation and UI responsiveness.
- Postman: Used for manual verification of the API endpoints during the development of the Scenario Executor.

Environments

- Simulation Sandbox (Docker): To validate attack capabilities safely, we will deploy vulnerable applications within isolated Docker containers. This ensures network separation between the "attack" targets and the host network.
- Local LLM Host (Ollama): We will emulate the AI generation environment using a local instance of Ollama. This allows us to test "jailbreak" attempts and attack planning without API costs, or leaking sensitive prompt data.

Test Data Sets

- RAG Knowledge Base: A curated set of OWASP Top 10 documentation and CVE summaries used to validate that the RAG pipeline retrieves relevant context.
- Control Targets: A set of "hardened" applications used to measure the false positive rate during effectiveness benchmarking.

Configuration & Management

- GitHub Actions: We have found that we can configure our CI/CD pipeline to trigger the Pytest suite on every pull request.
- Archiving: Test results, including generated vulnerability reports and execution logs, will be archived as in GitHub.