

LLM Red Team Web Framework: A Modular AI Security Testing Platform for Large Language Models

1st Mr.K.Rajesh

CSE Assistant Professor

Keshav Memorial Institute of Technology Keshav Memorial Institute of Technology Keshav Memorial Institute of Technology

Hyderabad 500029, India

rajesh.karanji1@gmail.com

2nd V. Harivaths

IV year CSE Student

Hyderabad 500029, India

vharivaths@gmail.com

3rd Mohammed Yasin Ali

IV year CSE Student

Hyderabad 500029, India

zainuali38@gmail.com

4th S. Raghu

IV year CSE Student

Keshav Memorial Institute of Technology

Hyderabad 500029, India

thisisraghunani@gmail.com

5th M. Rishikesh

IV year CSE Student

Keshav Memorial Institute of Technology

Hyderabad 500029, India

rishikesh0268@gmail.com

Abstract—Large Language Models (LLMs) are increasingly integrated into enterprise systems such as conversational agents, Retrieval-Augmented Generation (RAG) pipelines, autonomous AI agents, and document processing systems. Despite their capabilities, these systems remain vulnerable to prompt injection attacks, jailbreak attempts, sensitive data leakage, insecure tool usage, and unsafe autonomous behavior. Existing testing methodologies are often fragmented, manual, and lack centralized observability.

This paper presents the LLM Red Team Web Framework, a modular AI security testing platform designed to automate adversarial evaluation of LLM-powered systems. The framework supports multiple target types including Chat UI systems, RAG applications, agentic systems, API-based chat systems, and file upload pipelines. The platform enables users to create projects, configure target endpoints, select multiple vulnerability scanners, and execute security tests in isolated Docker-based environments. The framework provides real-time observability dashboards with live attack logs, severity visualization, Attack Success Rate (ASR), leakage metrics, and detection timing analytics. A dedicated findings module presents vulnerability evidence, mitigation strategies, and automated JIRA ticket creation support. Experimental evaluation demonstrates that the framework successfully automates adversarial testing workflows while improving visibility and reproducibility of AI security assessments.

Index Terms—LLM Security, AI Red Teaming, Prompt Injection, Jailbreak Detection, RAG Security, AI Observability, Cybersecurity, Docker Sandbox

I. INTRODUCTION

The rapid adoption of Large Language Models (LLMs) across enterprise applications has introduced a new class of security challenges [1]. Modern AI systems such as conversational agents, Retrieval-Augmented Generation (RAG) pipelines, autonomous AI agents, and multimodal systems are increasingly vulnerable to prompt injection, jailbreak attacks,

data leakage, insecure tool execution, and model manipulation [2].

Traditional cybersecurity testing frameworks are not designed to evaluate AI-specific vulnerabilities. Existing penetration testing tools primarily focus on network infrastructure, APIs, or web vulnerabilities, while modern AI systems require semantic and behavioral adversarial testing [3]. Furthermore, many AI security assessments are manual, inconsistent, and lack centralized monitoring capabilities.

Recent developments in AI red teaming have highlighted the need for automated testing frameworks capable of evaluating LLM robustness under adversarial conditions [4]. However, existing tools are often fragmented, difficult to integrate, or limited to specific attack categories.

To address these limitations, this paper proposes the **LLM Red Team Web Framework**, a modular and scalable platform for automated AI security testing. The framework enables users to configure AI targets, execute multiple vulnerability scanners, visualize attack metrics in real time, and generate actionable mitigation reports.

The major contributions of this work are:

- A modular web-based framework for AI red teaming across multiple LLM target types.
- Support for adversarial testing against Chat UI systems, RAG pipelines, autonomous agents, and file upload interfaces.
- Docker-isolated scanner execution for secure and reproducible testing.
- Real-time observability dashboards for vulnerability visualization and attack analytics.
- Automated findings management with evidence logs, severity classification, mitigation suggestions, and JIRA integration.

- Scalable backend architecture using FastAPI, Redis queues, PostgreSQL/MongoDB, and containerized execution environments.

II. RELATED WORK

A. AI Security and LLM Vulnerabilities

Recent studies have demonstrated that LLMs are vulnerable to prompt injection, jailbreak attacks, and sensitive information leakage [2]. OWASP has identified LLM security risks including insecure output handling, excessive agency, and training data poisoning [3].

B. AI Red Teaming

AI red teaming focuses on adversarial testing of machine learning systems under hostile conditions [4]. Organizations such as Anthropic and OpenAI employ structured red teaming methodologies to identify harmful model behaviors before deployment.

C. Observability in AI Systems

Modern AI observability platforms monitor latency, token usage, hallucinations, and user interactions [5]. However, most observability systems focus on operational metrics rather than adversarial security analysis.

D. Containerized Security Testing

Docker-based sandboxing has become a standard approach for secure execution of potentially dangerous workloads [6]. Container isolation enables reproducible and scalable execution of security scanners.

Existing research often addresses isolated aspects of AI security. The proposed framework extends prior work by integrating adversarial testing, observability, findings management, and scalable execution into a unified platform.

III. SYSTEM ARCHITECTURE

The LLM Red Team Web Framework follows a layered modular architecture comprising a frontend presentation layer, backend orchestration layer, scanner execution layer, and observability layer.

A. Frontend Layer

The frontend is implemented using React.js and provides:

- Landing page and project management interfaces
- Target endpoint configuration
- Vulnerability scanner selection
- Real-time testing dashboards
- Findings visualization and mitigation panels

The frontend communicates with the backend using RESTful APIs and WebSocket streams for live updates.

B. Backend Orchestration Layer

The backend is implemented using FastAPI and acts as the core control plane. Major responsibilities include:

- Authentication and project management
- Scanner orchestration
- Queue management using Redis
- Result aggregation and parsing
- Findings storage and severity classification

C. Experiment Execution Layer

Security scanners execute inside isolated Docker containers to ensure reproducibility and safe execution.

Supported scanner categories include:

- Prompt Injection Scanners
- Jailbreak Testing Tools
- RAG Leakage Analysis
- Agent Tool Abuse Simulators
- File Upload Exploit Testers

D. Observability Layer

The observability subsystem visualizes:

- Attack Success Rate (ASR)
- Detection Time
- Data Leakage Volume
- Vulnerability Severity Distribution
- Repeated Failures

Live logs are streamed directly to the dashboard for real-time monitoring.

IV. METHODOLOGY

The framework processes testing workflows through multiple stages.

A. Target Configuration

Users create projects and define target endpoints:

$$T = \{url, type, auth, metadata\} \quad (1)$$

where *type* represents the target category such as Chat UI, RAG, Agent, or File Upload system.

B. Scanner Selection

Users select one or more scanners:

$$S = \{s_1, s_2, \dots, s_n\} \quad (2)$$

Each scanner operates independently within a Docker container.

C. Attack Execution

Each scanner generates adversarial payloads:

$$A_i = f(s_i, T) \quad (3)$$

where A_i represents attack payloads generated for scanner s_i .

D. Attack Success Rate

Attack Success Rate (ASR) is computed as:

$$ASR = \frac{N_{success}}{N_{total}} \times 100 \quad (4)$$

where $N_{success}$ represents successful attacks and N_{total} represents total attack attempts.

E. Leakage Analysis

Leakage score is calculated as:

$$L = \frac{\text{Sensitive Tokens Exposed}}{\text{Total Tokens Tested}} \quad (5)$$

F. Severity Classification

Findings are classified into severity levels:

$$Severity \in \{Low, Medium, High, Critical\} \quad (6)$$

based on exploitability and impact.

V. SYSTEM WORKFLOW

The framework follows the workflow below:

- 1) User accesses the landing page
- 2) User creates a project
- 3) Target endpoint is configured
- 4) Target type is selected
- 5) Vulnerability scanners are selected
- 6) Security tests are executed
- 7) Results stream live to the dashboard
- 8) Findings are analyzed and stored
- 9) Mitigation suggestions are generated
- 10) JIRA tickets may be created

VI. IMPLEMENTATION

A. Technology Stack

- Frontend: React.js
- Backend: FastAPI
- Queue System: Redis
- Database: PostgreSQL / MongoDB
- Containerization: Docker
- Observability: Prometheus + Grafana
- Authentication: JWT

B. Backend Modules

TABLE I
BACKEND MODULE SUMMARY

Module	Purpose	AI Related
Projects	Project Management	No
Targets	Endpoint Config	No
Scanners	Attack Execution	Yes
Observability	Metrics Monitoring	Yes
Findings	Vulnerability Storage	Yes
Auth	Authentication	No
Docker Runner	Isolated Execution	Yes

VII. EXPERIMENTAL EVALUATION

A. Evaluation Setup

The framework was evaluated across multiple LLM targets including:

- Chat-based systems
- RAG applications
- Autonomous agents
- File upload interfaces

B. Prompt Injection Testing

TABLE II
PROMPT INJECTION EVALUATION

Attack Type	Detection Rate	ASR
Instruction Override	92%	18%
Context Escape	88%	22%
System Prompt Leak	90%	15%

C. Jailbreak Testing

TABLE III
JAILBREAK TESTING RESULTS

Technique	Success Rate	Detection
DAN Prompt	35%	91%
Roleplay Attack	28%	89%
Recursive Prompting	24%	86%

VIII. RESULTS AND DISCUSSION

The framework successfully automated adversarial testing workflows for multiple LLM architectures.

The observability dashboards improved visibility into attack patterns, repeated failures, and leakage metrics. Docker-based execution ensured scanner isolation and reproducibility.

The framework demonstrated strong scalability under concurrent testing workloads due to its Redis-backed queue architecture and stateless FastAPI backend.

However, certain limitations remain:

- External API rate limits may impact testing throughput
- Some jailbreak techniques evolve rapidly
- Advanced multimodal attacks are not fully supported

Future enhancements will focus on autonomous agent evaluation, multimodal red teaming, and adaptive attack generation using reinforcement learning.

IX. CONCLUSION

This paper presented the LLM Red Team Web Framework, a modular platform for automated AI security testing and observability.

The framework supports adversarial testing across multiple target categories, including conversational AI systems, RAG pipelines, autonomous agents, and file upload systems. Real-time dashboards, live log streaming, severity classification, and

findings management improve visibility and reproducibility of AI security assessments.

Experimental evaluation demonstrates that the framework effectively automates red teaming workflows while providing scalable observability and mitigation support for modern AI systems.

Future work includes multimodal vulnerability analysis, autonomous adversarial agents, and integration with enterprise SIEM platforms.

ACKNOWLEDGMENT

The authors would like to thank the Department of Computer Science and Engineering, Keshav Memorial Institute of Technology, for providing support and infrastructure for this research work.

REFERENCES

- [1] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2024.
- [2] S. Greshake et al., "More than you've asked for: A Comprehensive Analysis of Novel Prompt Injection Threats to Application-Integrated Large Language Models," 2023.
- [3] OWASP, "LLM Application Security Risks," 2024.
- [4] Anthropic, "AI Red Teaming Methodologies and Safety Evaluations," 2024.
- [5] M. Zaharia et al., "Operational Monitoring and Observability for AI Systems," IEEE Access, 2024.
- [6] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, 2023.