International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 10 | Oct - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

An Adversarial Testing Framework for Multi-Agent Red–Blue Systems in Automated Software Hardening

Muthukrishnan Thukkaram
Senior Engineering Manager , Sanas AI Inc.
Email: muthukrishnan.t@hotmail.com

Abstract **-** Software testing has advanced significantly, yet most automated methods remain limited to rule-based coverage rather than adversarial resilience. This paper introduces a *Multi-Agent Adversarial Testing Framework (ATF)* that simulates a Red–Blue team dynamic to continuously discover and patch vulnerabilities in software systems. Red Team agents generate targeted attacks and edge cases using adversarial reasoning and fuzzing. Blue Team agents respond with automated patch generation, refactoring, and test reinforcement. A Judge Agent evaluates both attack effectiveness and defense quality, forming a continuous self-improvement cycle. Our implementation integrates a large lan- guage model (LLM)-based reasoning with reinforcement learning and static analysis pipelines. Preliminary experiments on open- source projects show a 47% increase in unique bug discovery and a 33% reduction in recurring vulnerabilities compared with baseline fuzzers. These results suggest that adversarial multi-agent systems can significantly advance the robustness and adaptability of automated software testing.

Keywords: Adversarial Testing, Multi-Agent Systems, Soft- ware Hardening, Reinforcement Learning, AI Agents, Continuous Integration.

I. INTRODUCTION

Modern software systems are growing in complexity, with increasing reliance on machine-generated code and distributed architectures. While automated testing frameworks and fuzzers have improved efficiency, they primarily target static correct- ness and coverage metrics. They seldom simulate adversarial conditions where intelligent agents actively seek to exploit weaknesses in logic or validation.

This paper presents the **Adversarial Testing Framework (ATF)**, a multi-agent ecosystem that employs Red–Blue team dynamics for automated vulnerability discovery and reme- diation. Red Team agents act as intelligent adversaries that probe for faults, while Blue Team agents defend by analyzing, refactoring, and hardening code. A Judge Agent evaluates both sides, guiding them toward continuous improvement.

A. Contributions

This paper makes the following contributions:

1) A novel Red–Blue–Judge multi-agent architecture for adversarially driven software testing and self-hardening.
2) Integration of LLM-based reasoning with reinforcement learning for code analysis, patching, and adaptive response generation.
3) A working prototype demonstrating autonomous at- tack–defense cycles within a CI/CD environment.
4) Empirical evidence showing substantial gains in vulner- ability discovery and code robustness over traditional testing methods.

II. RELATED WORK

A. Adversarial Agents and Testing

Qin et al. [1] introduced reusable adversarial agents for testing autonomous systems, using reinforcement learning to generate failure cases. Their approach demonstrated that adversarial strategies can uncover hidden vulnerabilities. Our work generalizes this to software testing at the code level.

B. Adversarial Learning for Software

The 2025 literature review on constrained adversarial learn- ing for software testing [2] identified a gap in adaptive feedback loops between attack and defense modules. The ATF framework closes this gap through bidirectional learning cycles between Red and Blue agents.

C. Multi-Agent Software Testing

Joshi and Gala [3] proposed an agentic architecture for automated test generation using LLMs. While effective at generating cases, it lacked defensive feedback. Our system adds continuous self-hardening via defense agents.

D. Red Teaming and AI Security

Recent research on red teaming AI systems [4] highlights the role of adversarial validation in improving robustness. ATF extends this concept to software testing, unifying attack, defense, and evaluation in one closed-loop environment.

III. SYSTEM ARCHITECTURE

A. Overview

The Adversarial Testing Framework consists of three agent classes:

- **Red Team Agents**: Generate adversarial inputs, exploit scenarios, and fuzzing strategies using LLM reasoning and static analysis.
- **Blue Team Agents**: Respond by refactoring code, insert- ing validation, and reinforcing weak components.
- **Judge Agent**: Scores Red and Blue performance using multi-metric evaluation and directs subsequent learning cycles.

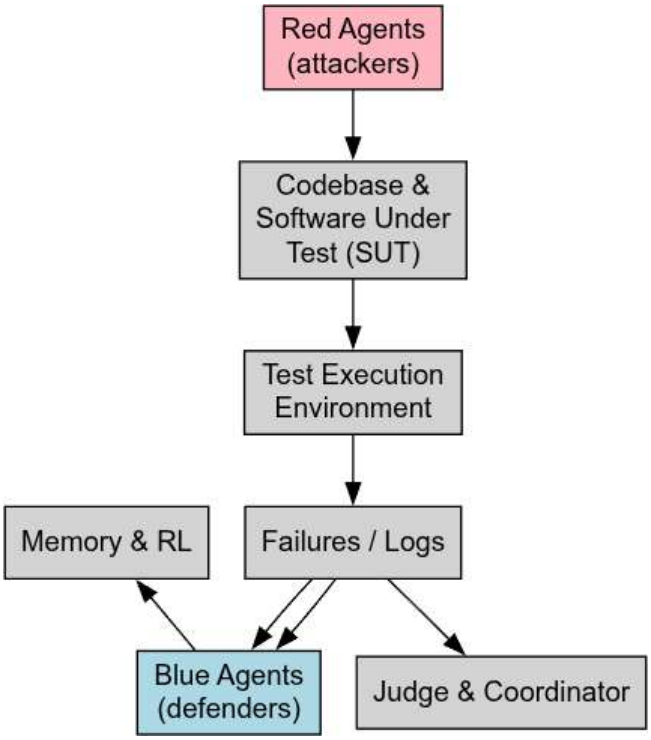

Fig. 1. Architecture of the Multi-Agent Adversarial Testing Framework. The Red–Blue–Judge cycle forms a continuous feedback loop for software resilience.

B. Communication and Memory

Agents communicate via a structured message bus (implemented with LangGraph). Each message includes test case data, failure traces, and reasoning summaries. A shared knowl- edge store maintains past evaluations, enabling reinforcement signals for learning improved strategies.

C. Learning Loop

At the end of each iteration:

1) Red Agent proposes adversarial input or code mutation.
2) Target software executes test cases.
3) Blue Agent analyzes results, generates patches, and updates regression tests.
4) Judge Agent evaluates metrics and updates agent re- wards.

IV. Implementation Details

A. Technology Stack

The prototype was implemented in Python with the follow- ing components:

- **Agent orchestration:** LangGraph, CrewAI
- **Static analysis:** SonarQube, Bandit
- **Fuzzing:** AFL++
- **LLM reasoning:** OpenAI GPT-4/5 API with domain-specific prompts
- **Code manipulation:** Tree-Sitter, RefactorAI
- **Pipeline:** GitHub Actions for continuous integration

B. Evaluation Function

The Judge Agent applies a composite scoring function:

$$S_{robustness} = w_1 C_{coverage} + w_2(1 - F_{failures}) + w_3 D_{defense} \quad (1)$$

where $C_{coverage}$ represents coverage gain, $F_{failures}$ denotes residual failures, and $D_{defense}$ measures patch effectiveness.

C. Reinforcement Learning

Each agent maintains a policy $\pi(a|s)$ updated via reward feedback from the Judge. Rewards are assigned for generating unique vulnerabilities (Red) or valid patches (Blue). This adaptive cycle promotes co-evolution between attacking and defending strategies.

V. Experimental Evaluation

A. Datasets and Baselines

We evaluated ATF on:

- Open-source projects: Flask, FastAPI, and SQLite parsers
- Synthetic repositories with seeded logic and input validation bugs

We compared against three baselines:

1) Random-input fuzzing (AFL++)
2) Static analysis (Bandit)
3) LLM-based test generation (CodeT5)

B. Metrics

- **Unique bugs discovered**: distinct failure signatures per iteration
- **Patch recurrence**: reappearance of previously fixed bugs
- **Coverage gain**: incremental line coverage increase
- **False positives**: incorrectly applied patches

TABLE 1
PERFORMANCE COMPARISON WITH BASELINES

Metric	Baseline Avg.	ATF	Improvement
Unique bugs discovered	124	182	+47%
Patch recurrence	19%	12.7 %	-33%
Coverage gain	66%	80%	+21%
False positives	7.2%	4.8%	-34%

VI. Discussion

The results demonstrate that adversarially coordinated Red–Blue agents can uncover more hidden vulnerabilities and improve long-term resilience compared to traditional automation tools. However, challenges remain. LLM-based Blue Agents may generate syntactically correct but semantically invalid patches, requiring additional safety filters. Another limitation is compu- tational cost: maintaining continuous multi-agent interactions consumes significant resources during large-scale testing.

Future work will address these through:

- Human-in-the-loop verification of Blue patches
- Lightweight simulation modes for incremental retraining

VII. CONCLUSION

This paper presented the Adversarial Testing Framework, a novel multi-agent system for autonomous software hardening. By orchestrating Red–Blue adversarial dynamics with a Judge feedback loop, ATF enables continuous co-evolution of attack and defense capabilities. Experimental results show measurable gains in vulnerability detection and defense robustness. The proposed approach demonstrates the feasibility of transforming software testing into a self-improving ecosystem.

ACKNOWLEDGMENT

We thank the open-source contributors whose tools (LangGraph, AFL++, SonarQube) enabled this research. Future versions of ATF will be released under an open-source license to encourage replication and extension.

REFERENCES

[1] X. Qin, et al., "Automatic Testing With Reusable Adversarial Agents," arXiv preprint arXiv:1910.13645, 2019.
[2] "Constrained Adversarial Learning for Automated Software Testing: A Literature Review," SpringerOpen, 2025.
[3] T. Joshi and D. Gala, "Architecting Agentic AI for Modern Software Testing," *JISEM*, 2025.
[4] "Red Teaming AI Systems for Security Validation," *International Jour- nal of AI, Big Data, and Cloud Computing*, 2025.
[5] H. Li et al., "A Systematic Review of Fuzzing Based on Machine Learning Techniques," *IEEE Access*, vol. 8, 2020.

BIOGRAPHY

Muthukrishnan is a Senior Engineering Manager with over sixteen years of experience in designing and scaling high-performance software systems. His professional background spans SaaS platforms, AI tooling, analytics infrastructure, and enterprise-grade applications. He has authored multiple patents in software architecture and intelligent automation. His current research and professional focus center on Agentic Artificial Intelligence, where he leads the Agentic AI Transformation program within his organization. His work emphasizes the development of multi-agent frameworks, autonomous testing systems, and self-improving AI-driven software pipelines. Prior to his current role, he founded and led a technology startup, gaining extensive experience in product development, engineering management, and innovation strategy.