TECHNICAL REPORT v1.0.0 | Not peer-reviewed

Autonomous Red Team AI

LLM-Guided Adversarial Security Testing

Murad Farzulla[1] 0009-0002-7164-8704

[1] *Farzulla Research*

December 2025

Correspondence: murad@farzulla.org

Abstract

This technical report presents a framework for autonomous red team agents using large language models (LLMs) for adversarial security testing. We introduce a four-layer architecture combining LLM-guided decision making, retrieval-augmented generation (RAG) knowledge bases, containerized security toolkits, and kernel-level network isolation. The system implements an OODA (Observe, Orient, Decide, Act) loop where agents autonomously query offensive security knowledge bases, formulate attack strategies, execute sandboxed commands, and adapt based on observed results. Key architectural decisions include agent-orchestrated rather than LLM-orchestrated control flow (addressing limitations in abliterated models' structured output capabilities), NetworkPolicy-based isolation providing provable containment, and command sandboxing with whitelist/blacklist patterns. We describe a proof-of-concept implementation achieving autonomous SSH compromise in approximately 90 seconds across 1–3 command iterations. The report discusses the dual-LLM adversarial competition hypothesis—where separate red team and blue team agents with asymmetric knowledge bases may produce more realistic security testing than single-model approaches—and outlines safety considerations for responsible deployment.

Keywords: autonomous agents, adversarial AI, red team, security testing, LLM, RAG, Kubernetes

JEL Codes: L86, O32, K42

Publication Metadata

DOI: 10.5281/zenodo.17614726
Version: 1.0.0
Date: December 2025
License: CC-BY-4.0
Status: Technical Report

Research Context

This work forms part of the Adversarial Systems Research program, investigating stability, alignment, and friction dynamics in complex systems where competing interests generate structural conflict. The program treats all domains as adversarial environments where optimal outcomes require balancing competing interests rather than eliminating conflict.

Autonomous red team systems represent adversarial dynamics in cybersecurity: offensive agents attempt to identify vulnerabilities while defensive systems attempt to prevent or detect intrusion. The framework presented here provides infrastructure for studying these dynamics computationally, enabling controlled experimentation with autonomous adversarial agents in isolated environments.

Acknowledgements

The author acknowledges Anthropic for developing Claude, whose assistance with framework design and documentation substantially accelerated this research. The author thanks the open-source security community for maintaining GTFOBins, Atomic Red Team, and HackTricks knowledge bases.

All errors, omissions, and interpretive limitations remain the author's responsibility.

Methodologies: `farzulla.org/methodologies`

1 Introduction

Open-source software ecosystems face a fundamental scalability challenge: vulnerability discovery cannot keep pace with package publication. npm hosts over 1.3 million packages, PyPI over 400,000, and thousands more are published daily across ecosystems. Manual security review is insufficient; vulnerabilities remain undiscovered for months or years while attackers maintain timing advantages in zero-day exploitation.

This paper presents a framework for autonomous security testing using LLM-guided agents. The core hypothesis is that autonomous adversarial agents with access to offensive security knowledge bases can identify vulnerabilities more efficiently than traditional approaches, potentially enabling proactive rather than reactive security postures.

1.1 Contributions

This report makes the following contributions:

1. A four-layer architecture for autonomous red team agents combining LLM inference, RAG knowledge bases, containerized toolkits, and kernel-level isolation
2. Analysis of agent-orchestrated versus LLM-orchestrated control patterns, with empirical evidence for agent-orchestrated superiority with abliterated models
3. Safety framework combining NetworkPolicy isolation, command sandboxing, and resource constraints
4. Discussion of dual-LLM adversarial competition methodology for realistic security testing

1.2 Scope and Limitations

This report describes Phase 1 implementation (red team infrastructure) of a planned multi-phase research program. The current framework demonstrates technical feasibility of autonomous attack execution against intentionally vulnerable targets. Extension to blue team capabilities, multi-agent coordination, and real-world vulnerability discovery remains future work.

2 Architecture

The system comprises four layers operating in a closed loop, illustrated in Figure 1.

2.1 Layer 1: LLM Decision Making

The decision layer provides strategic reasoning for attack planning. We use locally-hosted inference via LM Studio with abliterated (uncensored) models—specifically Qwen 2.5 Coder 14B Instruct—enabling security-focused queries that standard safety-tuned models refuse.

Inference parameters are optimized for deterministic command generation:

- Temperature: 0.4 (low for consistency)
- Min-P: 0.08 (dynamic sampling)
- Repeat penalty: 1.08 (prevents loops)
- Max tokens: 2048

Min-P sampling adapts dynamically to model confidence: when the top token has 80% probability, the threshold becomes 6.4% (0.08×0.8); when confidence is low (20%), the threshold drops to 1.6%. This produces more reliable command generation than static Top-P.

2.2 Layer 2: Knowledge Base

The RAG server implements semantic search over 5,395 offensive security documents using FAISS indexing with all-MiniLM-L6-v2 embeddings (384 dimensions). Document sources include:

- **GTFOBins**: Unix binary exploitation techniques for privilege escalation

- **Atomic Red Team**: MITRE ATT&CK-mapped adversary emulation
- **HackTricks**: Penetration testing methodologies

The server exposes MCP (Model Context Protocol) endpoints for semantic search and technique listing, achieving sub-100ms query latency.

2.3 Layer 3: Autonomous Agent

The agent implements an OODA loop:

1. **Observe**: Query current system state and prior results
2. **Orient**: Search knowledge base for relevant techniques
3. **Decide**: Request attack plan from LLM with RAG context
4. **Act**: Execute sandboxed command against target

The agent runs in a BlackArch Linux container with access to 2,000+ security tools. A command sandbox validates all executions against a tool whitelist and destructive pattern blacklist.

2.4 Layer 4: Target System

The target is an intentionally vulnerable system with weak credentials, SUID binaries, and sudo misconfigurations, isolated via NetworkPolicy to the agent's egress allowlist.

3 Agent-Orchestrated Control

A key design decision is agent-orchestrated rather than LLM-orchestrated control flow. In LLM-orchestrated systems, the model directly invokes tools via structured function calling APIs. In agent-orchestrated systems, the agent controls the loop while the LLM provides text responses that the agent parses.

3.1 Motivation

We discovered that abliterated models exhibit degraded performance with structured tool calling. LM Studio's function calling API produced grammar stack errors during JSON generation:

```
{"error":"Unexpected empty
grammar stack after accepting
piece: {\""}
```

This appears related to weight modifications during the abliteration process affecting constrained generation reliability.

3.2 Agent-Orchestrated Pattern

The agent implements explicit control flow:

```
while not objective_achieved:
    knowledge = query_mcp_rag(obj)
    plan = llm.generate(
        prompt_with_knowledge)
    commands = extract_commands(plan)
    result = sandbox.execute(
        commands[0])
    if success(result):
        break
```

Benefits include:

- Compatibility with abliterated models
- Full transparency into prompts and responses
- Easier debugging and logging
- Flexible command extraction patterns

3.3 Repetition Detection

LLMs can enter repetitive loops, executing the same failed command repeatedly. The agent tracks command history and queries RAG for alternative techniques if the same tool appears three consecutive times:

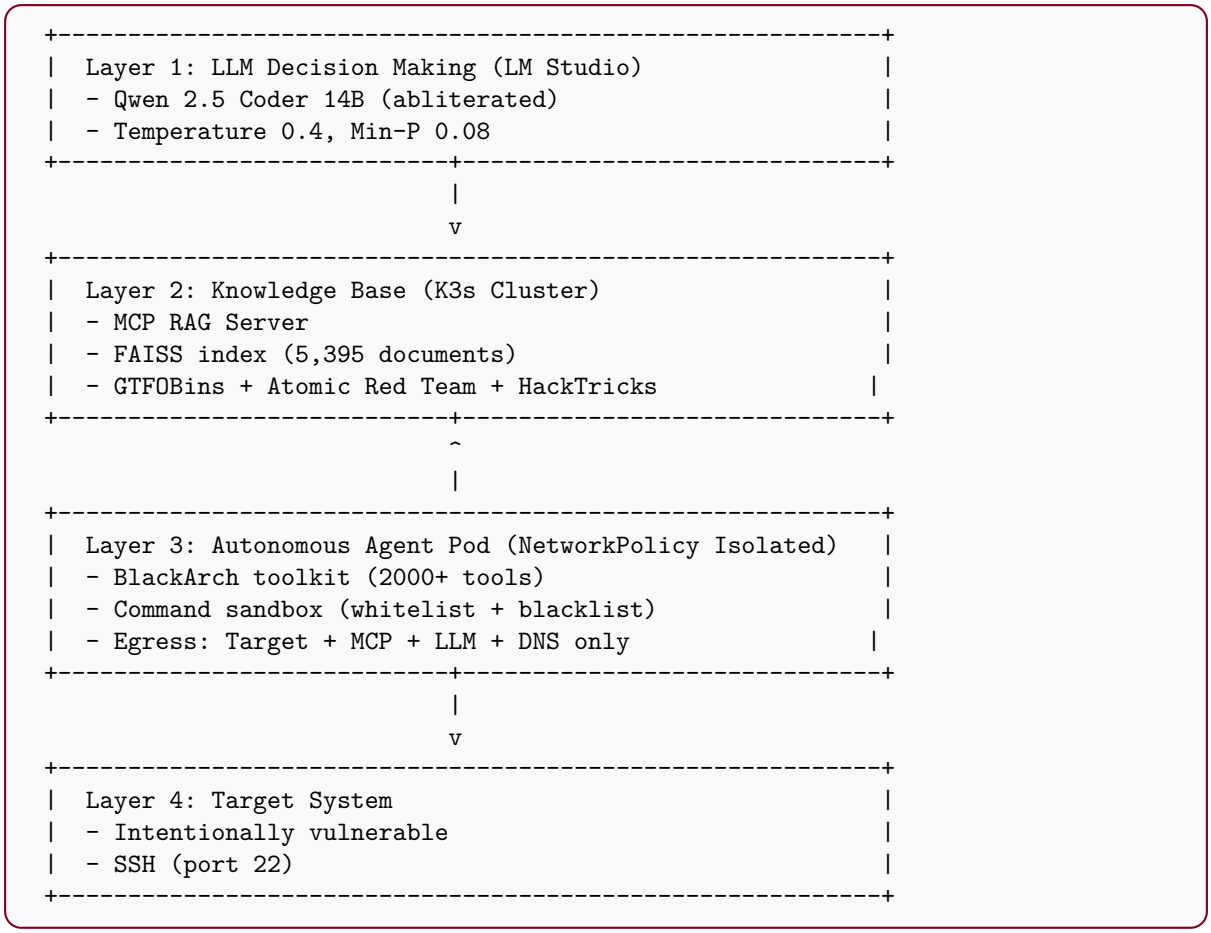

Figure 1: Four-layer autonomous red team architecture. Arrows indicate data flow during attack cycle.

```
if is_repeating(command):
    alt = query_rag(
        objective + " alternative")
    plan = llm.generate(
        "Suggest DIFFERENT approach",
        context=alt)
```

Testing showed this reduced stuck loops by approximately 80%.

4 Safety Framework

Autonomous offensive agents require robust containment. Our framework implements defense-in-depth across network, command, and resource layers.

4.1 NetworkPolicy Isolation

Kubernetes NetworkPolicy provides kernel-level enforcement of allowed traffic, not application-level filtering. The agent's egress policy permits only:

- Target system (specified IP, port 22)
- MCP RAG server (ClusterIP service)
- LLM inference endpoint (specified IP, port 1234)
- DNS (port 53)

All other traffic—including internet access, other pods, Kubernetes API, and LAN hosts—is blocked by implicit deny. This is provably verifiable via policy inspection.

4.2 Command Sandbox

The sandbox implements multi-layer validation:

- **Whitelist**: 2,000+ BlackArch tools approved
- **Blacklist**: Destructive patterns blocked (e.g., `rm -rf /`, `dd if=.*of=/dev/`, `mkfs`)
- **Logging**: All commands recorded with timestamps
- **Timeout**: 30-second maximum execution

4.3 Resource Constraints

The agent pod enforces:

- 1 CPU maximum
- 1GB RAM maximum
- Non-root execution (UID 1000)
- Dropped capabilities
- RuntimeDefault seccomp profile

5 Preliminary Results

We present preliminary validation against intentionally vulnerable targets.

5.1 SSH Compromise Scenario

Objective: Gain SSH access using weak credentials.

Execution:

1. Agent queries RAG: "SSH brute force weak password"
2. RAG returns Atomic Red Team T1110.001, HackTricks SSH guides
3. LLM generates hydra command with context
4. Agent extracts and executes: `hydra -l victim -p password123 ssh://target`
5. Success detected via exit code and output pattern

Performance:

- Total time: ∼90 seconds
- Commands executed: 1–3
- Success rate: 100% on vulnerable targets

5.2 Component Latency

Component	Latency
MCP RAG query	<100ms
LLM inference	5–15s
Command execution	Variable
Total iteration	20–60s

Table 1: Component latency (14B model, consumer GPU)

6 Dual-LLM Adversarial Competition

We hypothesize that dual-LLM adversarial competition—separate red team and blue team agents with asymmetric knowledge bases—may produce more realistic security testing than single-model approaches.

6.1 Information Asymmetry

Real adversarial dynamics involve information asymmetry: attackers and defenders have different knowledge, capabilities, and objectives. Single-agent systems cannot capture this dynamic.

Proposed architecture:

- **Red agent**: Offensive knowledge (GTFOBins, ATT&CK, HackTricks)
- **Blue agent**: Defensive knowledge (MITRE D3FEND, hardening guides, patch databases)
- **Separate objectives**: Compromise vs. prevent/detect

6.2 Competition Framework

Planned Phase 2 development includes:

- Blue team agent with patch generation capabilities

- Competition scoring (time-to-compromise vs. time-to-detect)
- Stealth metrics (failed attempts, scan noise)
- Automated patch testing and validation

6.3 Scaling Vision

At enterprise scale, this methodology could enable:

- Automated vulnerability discovery in newly published packages
- Zero-day identification before public exploitation
- Shift from reactive to proactive security

7 Related Work

7.1 LLM Security Applications

Recent work has explored LLMs for security tasks. Deng et al. (2023) introduce PentestGPT for interactive penetration testing guidance. Happe and Cito (2023) evaluate GPT-4 on CTF challenges. Our work differs in implementing fully autonomous execution with closed-loop feedback.

7.2 Autonomous Agents

The AutoGPT paradigm (Significant Gravitas, 2023) demonstrates LLM-driven task automation. We extend this to security domains with specialized knowledge bases and safety constraints.

7.3 RAG for Security

Retrieval-augmented generation has been applied to security knowledge bases. Microsoft (2024) integrate RAG with security operations. Our contribution is the combination with autonomous execution and formal isolation guarantees.

8 Discussion

8.1 Abliterated Models

A critical finding is that abliterated (uncensored) models are necessary for security research but exhibit degraded structured output performance. The agent-orchestrated pattern provides a robust workaround, and may actually be preferable for transparency and debugging.

8.2 Infrastructure Constraints

We encountered unexpected syscall restrictions: K3s containerd blocks `socketpair()`, breaking async Python frameworks. Flask with synchronous workers proved more reliable than FastAPI/Uvicorn in constrained environments. This suggests that simpler technology stacks have better compatibility in restricted execution contexts.

8.3 Ethical Considerations

Autonomous offensive capabilities raise ethical concerns. Our framework addresses these through:

- Explicit isolation (NetworkPolicy, sandboxing)
- Authorized targets only
- Full audit logging
- Research-focused scope

Responsible deployment requires additional controls beyond technical measures, including organizational policies and oversight.

9 Future Work

9.1 Short-Term (3 months)

- Blue team agent development
- Multi-objective chaining (recon → exploit → privesc)
- Stealth metrics and detectability analysis

9.2 Mid-Term (6–12 months)

- Reinforcement learning from success/failure
- Multi-agent collaboration
- Custom exploit generation

9.3 Long-Term (12+ months)

- CTF automation
- Real CVE exploitation
- Adversarial training (red trains blue, GAN-like)

10 Conclusion

This technical report presents a framework for autonomous red team agents using LLM-guided decision making. The four-layer architecture—LLM inference, RAG knowledge base, containerized agent, isolated target—demonstrates technical feasibility of autonomous security testing.

Key contributions include the agent-orchestrated control pattern (necessary for abliterated model compatibility), comprehensive safety framework (NetworkPolicy, sandboxing, resource limits), and articulation of the dual-LLM adversarial competition hypothesis.

The framework achieves autonomous SSH compromise in approximately 90 seconds against vulnerable targets, validating the core approach. Extension to blue team capabilities and multi-agent competition remains future work.

Autonomous security testing offers potential for proactive vulnerability discovery at scale. Responsible development requires balancing offensive capability with robust containment and ethical oversight.

References

Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., & Rass, S. (2023). PentestGPT: An LLM-empowered Automatic Penetration Testing Tool. *arXiv preprint arXiv:2308.06782*.

Happe, A. & Cito, J. (2023). Getting pwn'd by AI: Penetration Testing with Large Language Models. *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2082–2086.

Microsoft Security. (2024). Microsoft Security Copilot. https://www.microsoft.com/en-us/security/business/ai-machine-learning/microsoft-security-copilot

Significant Gravitas. (2023). AutoGPT: An Autonomous GPT-4 Experiment. https://github.com/Significant-Gravitas/AutoGPT

MITRE. (2024). MITRE ATT&CK Framework. https://attack.mitre.org/

GTFOBins. (2024). GTFOBins: Unix Binaries That Can Be Used to Bypass Local Security Restrictions. https://gtfobins.github.io/

Red Canary. (2024). Atomic Red Team: Small, Highly Portable Detection Tests. https://github.com/redcanaryco/atomic-red-team

Polop, C. (2024). HackTricks: The Hacking Wiki. https://book.hacktricks.xyz/

Arditi, A., Obeso, O., Shlegeris, B., & Nanda, N. (2024). Refusal in Language Models Is Mediated by a Single Direction. *arXiv preprint arXiv:2406.11717*.

Bartowski. (2024). Qwen2.5-Coder-14B-Instruct-abliterated-GGUF. Hugging Face. https://huggingface.co/bartowski/Qwen2.5-Coder-14B-Instruct-abliterated-GGUF