Dissensus AI Working Paper Series

DAI-2513

# Autonomous Red Team and Blue Team AI

*LLM-Guided Adversarial Security Competition*

Murad Farzulla[1,2,*] Andrew Maksakov[1]

[1]Dissensus AI, London, UK [2]King's College London, London, UK

[*]Correspondence: research@dissensus.ai

January 2026

**Abstract**

This technical report presents a framework for autonomous adversarial security competition using large language models (LLMs). We introduce a dual-agent architecture where autonomous red team and blue team agents compete in isolated environments: the red team attempts to compromise target systems while the blue team defends, detects, and remediates in real time. Phase 1 established the red team infrastructure—a four-layer architecture combining LLM-guided decision making, retrieval-augmented generation (RAG) over offensive knowledge bases, containerized security toolkits, and kernel-level network isolation. Phase 2, presented in this updated report, introduces the blue team agent with a five-phase defensive methodology (Audit, Detect, Analyze, Remediate, Harden), an LLM-assisted patch generation framework with rollback support, and a competition scoring engine that evaluates red vs. blue performance across weighted security dimensions. Key architectural decisions include agent-orchestrated control flow (addressing limitations in abliterated models' structured output capabilities), NetworkPolicy-based isolation, command sandboxing with defensive tool whitelisting, and MITRE D3FEND integration for defensive knowledge retrieval. The red team agent achieves autonomous SSH compromise in approximately 90 seconds; the blue team agent implements a DefenseSandbox restricting operations to whitelisted defensive tools (auditd, fail2ban, iptables, lynis, rkhunter, chkrootkit, aide, ossec). The competition scoring framework evaluates time-to-compromise vs. time-to-detect, patch effectiveness, and stealth metrics. We describe the full implementation and discuss implications for autonomous security testing at scale.

**Keywords:** autonomous agents, adversarial AI, red team, blue team, LLM, RAG, Kubernetes, patch generation, scoring framework

**JEL Codes:** L86, O32, K42

**DOI:** `10.5281/zenodo.17614726` | **Version:** v2.0.0 | **License:** CC-BY-4.0

## Research Context

This work forms part of the Adversarial Systems Research programme at Dissensus AI, investigating stability, alignment, and friction dynamics in complex systems where competing interests generate structural conflict. The formal foundations for this programme are developed in the Axiom of Consent (Farzulla, 2025a), which models friction as a coordination primitive in multi-agent systems—the dual-agent adversarial architecture presented here instantiates those dynamics in a cybersecurity domain where offensive and defensive agents generate friction through competing objectives under information asymmetry.

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. This paper also draws on our investigation of abliterated language models (Farzulla, 2025b), which examines how safety-alignment removal affects reasoning capabilities—a finding that directly motivated the agent-orchestrated control pattern described in Section 3.

## Acknowledgements

This paper benefited from extended collaboration with Claude (Anthropic), whose contributions to framework design, documentation, and iterative refinement were substantive. The authors gratefully acknowledge this assistance while taking full responsibility for all claims, errors, and interpretive choices.

The authors thank the open-source security community for maintaining GTFOBins, Atomic Red Team, and HackTricks knowledge bases.

This work is part of the Adversarial Systems Research program at Dissensus AI, a broader investigation into stability, alignment, and friction dynamics across political, financial, cognitive, and multi-agent systems. Related papers in the series are available through the Adversarial Systems & Complexity Research Initiative (ASCRI; systems.ac).

The authors welcome feedback, criticism, and collaboration. Correspondence should be directed to research@dissensus.ai.

## 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.

Recent work has demonstrated that large language models can autonomously exploit real-world vulnerabilities: Fang et al. (2024a) show that GPT-4-based agents exploit one-day CVEs with an 87% success rate, while Fang et al. (2024b) demonstrate autonomous web application compromise. These results suggest that autonomous offensive capabilities are no longer hypothetical—they are technically feasible with current foundation models. At the same time, the red-teaming literature has matured from manual human evaluation (Ganguli et al., 2022) through semi-automated frameworks to fully autonomous systems (Zhou et al., 2025), and Meta's CyberSecEval benchmark suite (Bhatt et al., 2023) has established standardised evaluation methodology for LLM cybersecurity risks. What remains underdeveloped is the defensive counterpart: most autonomous security systems focus exclusively on offence, leaving the question of how autonomous defenders co-evolve with autonomous attackers largely unaddressed.

This paper presents a framework for autonomous security testing using LLM-guided agents that addresses this gap. The core hypothesis is that autonomous adversarial agents with access to offensive security knowledge bases can identify vulnerabilities more efficiently than traditional approaches, and that pairing them with autonomous defensive agents under a competition scoring framework produces more realistic security testing than single-agent approaches. The dual-agent architecture enforces information asymmetry—separate knowledge bases, separate toolkits, separate objectives—that mirrors the structure of real adversarial engagements.

### 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 (Phase 1)
2. Analysis of agent-orchestrated versus LLM-orchestrated control patterns, with empirical evidence for agent-orchestrated superiority with abliterated models (Phase 1)
3. Safety framework combining NetworkPolicy isolation, command sandboxing, and resource constraints (Phase 1)
4. An autonomous blue team agent implementing a five-phase defensive methodology with DefenseSandbox and MITRE D3FEND knowledge base integration (Phase 2)
5. An LLM-assisted patch generation framework with known-pattern matching, LLM fallback, validation, and automatic rollback (Phase 2)
6. A competition scoring engine with weighted evaluation across vulnerability remediation, service availability, detection speed, and hardening effectiveness (Phase 2)
7. Full dual-LLM adversarial competition implementation with Kubernetes deployment manifests and namespace isolation (Phase 2)

### 1.2 Scope and Limitations

This report describes Phase 1 (red team infrastructure) and Phase 2 (blue team development) of a multi-phase research program. Phase 1 demonstrates technical feasibility of autonomous attack execution against intentionally vulnerable targets. Phase 2 extends the framework with autonomous defensive capabilities, patch generation, and competition scoring. Extension to multi-agent coordination, reinforcement learning, 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% ($0.08 \times 0.8$); 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—a security-domain variant of the ReAct paradigm (Yao et al., 2023) that interleaves reasoning and acting, augmented with retrieval-augmented generation (Lewis et al., 2020) for domain-specific knowledge:

1. **Observe**: Query current system state and prior results
2. **Orient**: Search knowledge base for relevant techniques

```
+----------------------------+------------------------------+
|  Layer 1: LLM Decision Making (LM Studio)                 |
|  - Qwen 2.5 Coder 14B (abliterated)                       |
|  - Temperature 0.4, Min-P 0.08                            |
+----------------------------+------------------------------+
                             |
                             v
+-----------------------------------------------------------+
|  Layer 2: Knowledge Base (K3s Cluster)                    |
|  - MCP RAG Server                                         |
|  - FAISS index (5,395 documents)                          |
|  - GTFOBins + Atomic Red Team + HackTricks               |
+----------------------------+------------------------------+
                             ^
                             |
+-----------------------------------------------------------+
|  Layer 3: Autonomous Agent Pod (NetworkPolicy Isolated)   |
|  - BlackArch toolkit (2000+ tools)                        |
|  - Command sandbox (whitelist + blacklist)                |
|  - Egress: Target + MCP + LLM + DNS only                 |
+----------------------------+------------------------------+
                             |
                             v
+-----------------------------------------------------------+
|  Layer 4: Target System                                   |
|  - Intentionally vulnerable                               |
|  - SSH (port 22)                                          |
+-----------------------------------------------------------+
```

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

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—a finding consistent with broader evidence that abliteration degrades capabilities beyond mere safety-refusal removal (Arditi et al., 2024; Farzulla, 2025b). The GCG attack literature (Zou et al., 2023) demonstrates that safety alignment can be circumvented through adversarial suffixes, but abliteration takes a more aggressive approach by removing the refusal direction entirely from the model's representation space, with collateral effects on constrained generation. 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:

```
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 Blue Team Agent Architecture

Phase 2 introduces an autonomous blue team agent that mirrors the red team’s OODA loop but operates with defensive objectives and a distinct knowledge base. Vyas et al. (2025) survey the landscape of autonomous cyber network defence and identify a persistent gap between offensive agent capabilities and defensive automation—the blue team agent described here addresses this asymmetry directly. The blue team agent implements a five-phase defensive methodology.

### 6.1 Five-Phase Defensive Methodology

The blue team agent executes a structured defense cycle:

1. **Audit**: Enumerate system state—running services, open ports, user accounts, SUID binaries, cron jobs, file permissions, and installed packages
2. **Detect**: Identify anomalies and known vulnerability patterns by comparing audit results against security baselines
3. **Analyze**: Query the MITRE D3FEND knowledge base via MCP to retrieve defensive techniques mapped to detected threats

4. **Remediate**: Generate and apply security patches using the patch generation framework (Section 7)

5. **Harden**: Apply proactive hardening measures—firewall rules, service configuration, access controls—beyond immediate vulnerability remediation

### 6.2 DefenseSandbox

Analogous to the red team's command sandbox, the blue team operates within a `DefenseSandbox` that restricts execution to whitelisted defensive tools:

- **Audit tools**: `auditd`, `lynis`, `aide`
- **Detection tools**: `rkhunter`, `chkrootkit`, `ossec`
- **Remediation tools**: `fail2ban`, `iptables`, `systemctl`
- **System utilities**: `chmod`, `chown`, `usermod`, `passwd`

Destructive operations (e.g., `rm -rf`, disk writes, kernel module loading) are blacklisted. All defensive actions are logged with timestamps for scoring and audit purposes.

### 6.3 Defensive Knowledge Base

The blue team agent queries a separate RAG knowledge base built from defensive security resources:

- **MITRE D3FEND** (MITRE, 2024b): Defensive technique taxonomy mapped to ATT&CK techniques
- **CIS Benchmarks**: System hardening baselines
- **Patch databases**: Known vulnerability remediation patterns

This creates information asymmetry: the red team has offensive knowledge (GTFOBins, ATT&CK, HackTricks) while the blue team has defensive knowledge (D3FEND, hardening guides, patch patterns). Neither agent has access to the other's knowledge base.

### 6.4 Infrastructure

The blue team agent runs in a dedicated Kubernetes namespace (`blueteam`) with its own:

- **Container**: Dockerfile with defensive security tools pre-installed
- **NetworkPolicy**: Egress limited to target system, MCP server, LLM endpoint, and DNS
- **ResourceQuota**: CPU and memory limits matching red team constraints
- **Deployment manifests**: Kubernetes YAML for pod, service, and network policy

A deployment script (`deploy-blueteam.sh`) automates namespace creation, image building, and manifest application.

## 7 Patch Generation Framework

The patch generator implements a two-tier strategy for vulnerability remediation.

## 7.1 Known-Pattern Matching

The first tier uses deterministic pattern matching for common vulnerability classes:

- **SSH hardening**: Disable root login, enforce key-only authentication, restrict allowed users
- **SUID remediation**: Remove unnecessary SUID bits, apply least-privilege permissions
- **Sudo misconfiguration**: Remove NOPASSWD entries, restrict sudo access to specific commands
- **Cron job sanitization**: Validate cron entries, remove unauthorized scheduled tasks
- **File permissions**: Fix world-writable files, correct ownership on sensitive files

Each pattern produces a structured patch containing the commands to apply, a validation check to confirm effectiveness, and a rollback procedure.

## 7.2 LLM-Assisted Patch Generation

When no known pattern matches, the second tier queries the LLM with vulnerability context and defensive knowledge from RAG:

```
if not known_pattern(vulnerability):
    context = query_defend_rag(vuln)
    patch = llm.generate(
        "Generate remediation",
        context=context)
    validate_and_apply(patch)
```

## 7.3 Validation and Rollback

Every patch undergoes validation before being considered successful:

1. **Pre-application**: Record system state snapshot
2. **Application**: Execute patch commands within DefenseSandbox
3. **Validation**: Run verification check (e.g., confirm SSH config changed, verify SUID bit removed)
4. **Rollback**: If validation fails, automatically restore pre-patch state

This ensures defensive actions do not degrade system availability—a critical scoring dimension.

## 8 Competition Scoring Framework

The scoring engine evaluates red team vs. blue team performance across multiple weighted dimensions.

## 8.1 Scoring Dimensions

Red team scoring mirrors this with offensive dimensions: time-to-compromise, stealth (failed detection avoidance), persistence, and lateral movement.

| Dimension | Weight | Measures |
|---|---|---|
| Vulnerability remediation | 30% | Patches applied |
| Service availability | 25% | Uptime maintained |
| Detection speed | 20% | Time to detect |
| Hardening effectiveness | 15% | Controls applied |
| Incident response | 10% | Response quality |

Table 2: Blue team scoring dimensions and weights

### 8.2 Report Generation

The `ReportGenerator` produces competition results in three formats:

- **Text**: Human-readable summary with section headers and formatted tables
- **JSON**: Machine-parseable structured data for integration with CI/CD pipelines
- **Markdown**: Formatted reports suitable for documentation and GitHub rendering

Each report includes per-dimension scores, overall weighted scores, competition winner determination, and timeline of key events.

## 9 Dual-LLM Adversarial Competition

Phase 1 hypothesised 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. This hypothesis draws on the broader adversarial co-evolution paradigm: Ge et al. (2023) demonstrate that iterative red-blue LLM competition reduces safety violation rates by 84.7%, and Landolt et al. (2025) survey the growing literature on multi-agent reinforcement learning for cybersecurity, where adversarial training produces more robust defensive policies than static rule-based approaches. Phase 2 implements this hypothesis.

### 9.1 Information Asymmetry

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

The implemented architecture enforces this asymmetry:

- **Red agent**: Offensive knowledge (GTFOBins, ATT&CK, HackTricks) via offensive RAG server
- **Blue agent**: Defensive knowledge (MITRE D3FEND, hardening guides, patch databases) via defensive RAG server
- **Separate objectives**: Compromise vs. prevent/detect/remediate
- **Separate sandboxes**: Offensive tool whitelist vs. defensive tool whitelist

### 9.2 Competition Framework

Phase 2 implements the following competition components:

- Blue team agent with five-phase defensive methodology (Section 6)
- LLM-assisted patch generation with validation and rollback (Section 7)

- Competition scoring across weighted dimensions (Section 8)
- Automated report generation in text, JSON, and Markdown formats

### 9.3 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

## 10 Related Work

We situate this work within six intersecting literatures: LLM-based penetration testing, autonomous red-teaming of AI systems, LLM agent architectures, adversarial robustness and safety alignment, multi-agent cybersecurity simulation, and regulatory frameworks for autonomous offensive AI.

### 10.1 LLM-Based Penetration Testing

The application of large language models to penetration testing has progressed rapidly from interactive assistance to fully autonomous execution. Deng et al. (2024) introduce PentestGPT, which uses GPT-4 as an interactive reasoning engine for penetration testing guidance—the operator remains in the loop, selecting which suggestions to execute. Happe and Cito (2023) evaluate GPT-4 on capture-the-flag (CTF) challenges and find that the model can solve simple challenges but struggles with multi-step exploitation chains. These early systems established that LLMs possess relevant security knowledge but left execution to human operators.

Subsequent work has moved toward full autonomy. Shen et al. (2024) propose PentestAgent, a multi-agent system that decomposes penetration testing into reconnaissance, scanning, and exploitation phases with RAG-augmented knowledge retrieval—architecturally the closest parallel to our red team agent. Nakatani (2025) demonstrate RapidPen, a ReAct-style agent with RAG over exploit databases that achieves shell access in 200–400 seconds at approximately $0.30–0.60 per run, providing a useful cost-performance baseline for our own 90-second SSH compromise. Kong et al. (2025) introduce VulnBot, which uses a penetration task graph to coordinate multiple specialised agents across reconnaissance, scanning, and exploitation. Singer et al. (2025) present Incalmo for multi-host network red-teaming, using LLMs for high-level planning with domain-specific task agents for execution—they evaluate PentestGPT and CyberSecEval3 as baselines and find both insufficient for multi-host scenarios. Mei et al. (2025) describe AutoPen, which combines RAG, chain-of-thought reasoning, and structured output for autonomous penetration testing. Wang et al. (2025) introduce the Planner-Executor-Perceptor (PEP) paradigm, which separates strategic planning from tactical execution using classical planning algorithms alongside LLM reasoning.

Our framework differs from these systems in two respects. First, we pair the offensive agent with an autonomous defensive agent under a competition scoring framework—most existing work focuses exclusively on the offensive side. Second, we document the agent-orchestrated control pattern necessitated by abliterated model limitations, a practical constraint that other systems using safety-aligned models through commercial APIs do not encounter.

## 10.2 Autonomous Red-Teaming of AI Systems

The term "red-teaming" in the AI safety literature refers to a distinct practice from cybersecurity red-teaming: eliciting harmful, biased, or policy-violating outputs from language models. Ganguli et al. (2022) provide the foundational treatment from Anthropic, establishing methods, scaling behaviours, and lessons learned from human red-teaming of LLMs. Ahmad et al. (2025) describe OpenAI's approach to external red-teaming, including structured evaluation design and outcome reporting. Feffer et al. (2024) critically examine whether AI red-teaming constitutes genuine security evaluation or "security theatre," arguing that the practice has been imported from cybersecurity without sufficient attention to what it can and cannot establish about model safety.

Several systems have automated this process. Zhou et al. (2025) present AutoRedTeamer, a multi-agent architecture with lifelong attack memory that achieves 20% higher attack success rates on HarmBench than prior methods—their dual-agent structure (attack proposer plus red-teaming agent) parallels our own dual-agent design, though they target LLM safety alignment rather than system-level vulnerabilities. Poomekum et al. (2026) introduce Chimera-RL, which uses hierarchical reinforcement learning with mission-graph exploration for campaign-level vulnerability discovery, including forensic logging and mitigation playbook generation. Xu et al. (2024) develop RedAgent, a context-aware autonomous agent with self-reflecting memory that jailbreaks most black-box LLMs within five queries and discovers 60 vulnerabilities in deployed GPT applications. Dawson et al. (2025) propose AIRTBench, a benchmark of 70 CTF challenges for evaluating autonomous red-teaming capabilities—Claude 3.7 Sonnet leads at 61% completion, establishing a current ceiling for LLM-based autonomous security performance.

Our work bridges these two senses of red-teaming: we use LLMs as autonomous agents that perform cybersecurity red-teaming (system-level vulnerability discovery and exploitation), not AI safety red-teaming (eliciting harmful model outputs). The dual-agent competitive architecture, however, draws on insights from both traditions.

## 10.3 LLM Agent Architectures

The autonomous agent pattern we employ—interleaved reasoning and action with environmental feedback—was formalised by Yao et al. (2023) as the ReAct paradigm. Chain-of-thought prompting (Wei et al., 2022) provides the reasoning substrate: decomposing complex tasks into sequential steps that the model can reason through before acting. The AutoGPT paradigm (Significant Gravitas, 2023) demonstrated that LLMs could drive open-ended task automation loops, though with limited reliability on complex multi-step objectives.

Retrieval-augmented generation (Lewis et al., 2020) addresses the knowledge limitation inherent in parametric models by retrieving relevant documents at inference time. Our architecture uses RAG over three offensive knowledge bases (GTFOBins, Atomic Red Team, HackTricks) for the red agent and defensive resources (D3FEND, CIS Benchmarks, patch databases) for the blue agent—creating the information asymmetry that distinguishes our framework from single-knowledge-base systems. Microsoft (2024) integrate RAG with security operations in a commercial product, though without autonomous execution capabilities.

A key architectural contribution of our work is the distinction between agent-orchestrated and LLM-orchestrated control flow. In LLM-orchestrated systems, the model directly invokes tools through structured function-calling APIs. In agent-orchestrated systems, the agent controls the execution loop while the LLM provides text-based reasoning that the agent parses. We find that agent-orchestrated control is necessary when working with abliterated models whose constrained-generation capabilities have been degraded—a practical constraint that has received limited attention in the agent architecture literature, which largely assumes access to well-behaved commercial models.

### 10.4 Adversarial Robustness and Safety Alignment

Our use of abliterated models connects to the broader literature on adversarial attacks against safety-aligned language models. Zou et al. (2023) introduce the Greedy Coordinate Gradient (GCG) attack, which generates universal adversarial suffixes that transfer across models and elicit harmful outputs from safety-aligned systems. This work established that safety alignment is circumventable through systematic optimisation, motivating alternative approaches to safety constraint removal. Arditi et al. (2024) demonstrate that refusal behaviour in language models is mediated by a single direction in representation space, and that removing this direction (abliteration) produces models that comply with arbitrary requests—including security-relevant ones. We use an abliterated Qwen 2.5 Coder 14B (Bartowski, 2024) for precisely this reason: security research requires models that do not refuse to discuss offensive techniques.

Farzulla (2025b) investigate the downstream consequences of abliteration in detail, finding that abliterated models exhibit genre mimicry—they reproduce the stylistic surface of harmful content without preserving the ethical reasoning capabilities that safety-trained models possess. This finding directly motivated our agent-orchestrated control pattern: if abliteration degrades structured output capabilities alongside safety refusal, the agent must compensate by managing the control flow externally.

Meta's CyberSecEval benchmark suite provides standardised evaluation of LLM cybersecurity risks across three iterations: Bhatt et al. (2023) introduce the original benchmark covering insecure code generation and cyberattack compliance; Bhatt et al. (2024) extend it with prompt injection, code interpreter abuse, and exploit generation; and Wan et al. (2024) add automated social engineering, scaling manual offensive operations, and autonomous offensive operations. These benchmarks establish baseline expectations for what LLMs can accomplish autonomously in security contexts and provide evaluation methodology that future iterations of our framework should adopt.

### 10.5 Multi-Agent Cybersecurity Simulation

The framing of cybersecurity as adversarial competition between autonomous agents has a growing literature. Vyas et al. (2025) provide a comprehensive survey of autonomous cyber network defence, covering both offensive and defensive agent architectures plus the simulation environments in which they operate—they propose criteria for evaluating autonomous defence algorithms that our scoring framework partially addresses. Landolt et al. (2025) survey multi-agent reinforcement learning for cybersecurity, covering AICA (Autonomous Intelligent Cyber-defence Agent) architectures and cyber gymnasium environments designed for training adversarial policies.

Ge et al. (2023) demonstrate MART (Multi-round Automatic Red-Teaming), where an adversarial LLM iteratively attacks a target LLM, reducing safety violation rates by 84.7% through the adversarial training process—the closest published parallel to our dual-LLM competitive architecture, though MART targets LLM safety alignment rather than system-level security. Our framework extends this adversarial co-evolution concept to cybersecurity: the red and blue agents compete over system-level vulnerabilities with asymmetric knowledge bases, and the competition scoring framework quantifies the outcome across multiple weighted dimensions.

The information asymmetry in our architecture—separate offensive and defensive knowledge bases, separate toolkits, separate objectives—reflects the structure of real adversarial engagements and connects to the game-theoretic foundations of cyber defence, where attackers and defenders have different information sets, capabilities, and payoff structures. The Axiom of Consent framework (Farzulla, 2025a) provides formal treatment of friction dynamics in precisely such multi-agent adversarial scenarios, including MARL experiments that demonstrate how preference alignment and intensity affect coordination outcomes under competing objectives.

### 10.6 Regulatory and Ethical Frameworks

Autonomous offensive AI systems operate in a rapidly evolving regulatory landscape. The NIST Artificial Intelligence Risk Management Framework (NIST, 2023) provides structured methodology for identifying and managing AI risks, including adversarial use. The EU AI Act (European Union, 2024) classifies AI systems by risk level, with autonomous offensive capabilities falling into categories that require transparency obligations and human oversight. Longpre et al. (2024) propose legal and technical safe harbours for AI safety evaluation and red-teaming research, arguing that the dual-use nature of offensive security research should not deter responsible investigation but should be conducted under structured protections.

Our framework addresses these concerns through technical isolation (NetworkPolicy, sandboxing, resource constraints), operational controls (authorised targets only, full audit logging), and architectural separation (offensive and defensive agents constrained to whitelisted tool sets). However, we note that technical measures alone are insufficient for responsible deployment—organisational policies, oversight mechanisms, and legal clarity remain essential complements to the engineering safeguards described here.

## 11 Discussion

### 11.1 Abliterated Models

A critical finding is that abliterated (uncensored) models are necessary for security research but exhibit degraded structured output performance. Arditi et al. (2024) show that refusal is mediated by a single representational direction, and removing it produces compliance with arbitrary requests—but the abliteration process appears to have collateral effects on constrained generation that Zou et al. (2023)'s adversarial suffix approach does not. The agent-orchestrated pattern provides a robust workaround, and may actually be preferable for transparency and debugging. This finding is consistent with Farzulla (2025b)'s observation that abliterated models reproduce harmful content's surface form without preserving the reasoning structure that enables reliable structured output.

### 11.2 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.

### 11.3 Defensive Agent Design

Phase 2 revealed that defensive agents benefit from a more structured methodology than offensive agents. The five-phase cycle (Audit, Detect, Analyze, Remediate, Harden) provides a deterministic scaffold that reduces LLM hallucination risk—each phase has concrete, verifiable outputs. The two-tier patch generation strategy (known patterns first, LLM fallback second) similarly improves reliability by preferring deterministic remediation over generative approaches.

The DefenseSandbox proved essential: early testing without sandbox restrictions allowed the blue team agent to execute commands outside its intended defensive scope. Whitelisting defensive tools prevents scope creep while maintaining sufficient capability.

### 11.4 Scoring Challenges

Designing fair competition scoring required balancing multiple concerns. Service availability (25% weight) prevents the blue team from “defending” by shutting down services. Detection speed (20%) rewards proactive monitoring rather than post-compromise forensics. The weighting scheme remains configurable for different competition scenarios.

### 11.5 Ethical Considerations

Autonomous offensive capabilities raise ethical concerns that the research community has begun to address systematically. Feffer et al. (2024) argue that red-teaming practices imported from cybersecurity into AI safety require careful examination of what they can and cannot establish. Longpre et al. (2024) propose safe harbour protections for AI safety evaluation, recognising that responsible offensive research requires institutional and legal frameworks alongside technical safeguards. The NIST AI Risk Management Framework (NIST, 2023) and EU AI Act (European Union, 2024) provide regulatory context for autonomous offensive systems, though neither fully addresses the dual-use challenges specific to LLM-guided adversarial agents.

Our framework addresses these concerns through:

- Explicit isolation (NetworkPolicy, sandboxing) for both agents
- Authorised targets only
- Full audit logging of all offensive and defensive actions
- Research-focused scope
- Defensive agent restricted to whitelisted security tools

Responsible deployment requires additional controls beyond technical measures, including organisational policies, oversight mechanisms, and the kind of structured safe-harbour protections that Longpre et al. (2024) advocate for the broader AI safety research community.

## 12 Future Work

With Phase 1 (red team) and Phase 2 (blue team) complete, the following directions remain.

### 12.1 Short-Term

- Multi-objective chaining (recon $\rightarrow$ exploit $\rightarrow$ privesc $\rightarrow$ persistence)
- Stealth metrics and detectability analysis
- Live competition execution with simultaneous red/blue agents
- Expanded defensive knowledge base coverage

### 12.2 Mid-Term

- Reinforcement learning from competition outcomes (win/loss signals), drawing on the MARL cyber defence literature surveyed by Landolt et al. (2025)
- Multi-agent collaboration (multiple red or blue agents coordinating), extending the multi-agent penetration testing approaches of Kong et al. (2025) and Shen et al. (2024)
- Custom exploit generation, building on Fang et al. (2024a)'s demonstration that LLMs can autonomously exploit real CVEs
- Adversarial training (red trains blue, GAN-like dynamics), following the MART paradigm of Ge et al. (2023)
- Standardised evaluation against AIRTBench (Dawson et al., 2025) and CyberSecEval 3 (Wan et al., 2024) benchmarks

### 12.3 Long-Term

- CTF automation with full attack chains
- Real CVE exploitation and automated patching
- Deployment across package ecosystems for continuous security monitoring
- Integration with the Chimera-RL (Poomekum et al., 2026) approach to hierarchical campaign-level vulnerability discovery

## 13 Conclusion

This technical report presents a complete framework for autonomous adversarial security competition using LLM-guided agents. Phase 1 established the red team infrastructure: a four-layer architecture achieving autonomous SSH compromise in approximately 90 seconds. Phase 2 extends the framework with a blue team agent, patch generation framework, and competition scoring engine.

Key contributions span both phases: the agent-orchestrated control pattern (necessary for abliterated model compatibility), comprehensive safety framework (NetworkPolicy, sandboxing, resource limits), five-phase defensive methodology with DefenseSandbox, two-tier patch generation with validation and rollback, and a weighted competition scoring engine producing multi-format reports.

The dual-LLM adversarial competition hypothesis—that separate agents with asymmetric knowledge bases produce more realistic security testing—is now implemented as a complete system with Kubernetes deployment manifests, namespace isolation, and automated scoring. The framework provides infrastructure for studying adversarial dynamics computationally: how autonomous attackers and defenders co-evolve strategies under information asymmetry.

Autonomous adversarial security competition offers potential for proactive vulnerability discovery at scale. Responsible development requires balancing offensive and defensive capabilities with robust containment and ethical oversight. Future work focuses on live competition execution, reinforcement learning from outcomes, and scaling to enterprise environments.

## Declarations

**Conflict of Interest.** The authors declare no competing interests.

**Funding.** This research received no external funding.

**Data Availability.** No datasets were generated or analysed during this study. The framework code is available at github.com/studiofarzulla/adversarial-security-agents.

**AI Assistance.** Claude (Anthropic) was used as a research collaborator for framework design, documentation, and iterative refinement. All intellectual claims and errors remain the authors' responsibility.

**Author Contributions.** M.F. designed the research program, developed the framework architecture, and wrote the manuscript. A.M. contributed to implementation and testing of the red team and blue team agent infrastructure.

## References

Ahmad, L., Agarwal, S., Lampe, M., & Mishkin, P. (2025). OpenAI's Approach to External Red Teaming for AI Models and Systems. *arXiv preprint*.

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

Bhatt, M., Chennabasappa, S., Nikolaidis, C., Wan, S., Evtimov, I., Gabi, D., Song, D., Ahmad, F., Aschermann, C., et al. (2023). Purple Llama CyberSecEval: A Secure Coding Benchmark for Language Models. *arXiv preprint arXiv:2312.04724*.

Bhatt, M., Chennabasappa, S., Li, Y., Nikolaidis, C., Song, D., Wan, S., Ahmad, F., Aschermann, C., Chen, Y., Kapil, D., Molnar, D., Whitman, S., & Saxe, J. (2024). CyberSecEval 2: A Wide-Ranging Cybersecurity Evaluation Suite for Large Language Models. *arXiv preprint*.

Dawson, A., Mulla, R., Landers, N., & Caldwell, S. (2025). AIRTBench: Measuring Autonomous AI Red Teaming Capabilities in Language Models. *arXiv preprint*.

Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., & Rass, S. (2024). PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. *Proceedings of the 33rd USENIX Security Symposium*.

European Union. (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council (Artificial Intelligence Act). *Official Journal of the European Union*, L 2024/1689.

Fang, R., Bindu, R., Gupta, A., Zhan, Q., & Kang, D. (2024a). LLM Agents Can Autonomously Exploit One-Day Vulnerabilities. *arXiv preprint arXiv:2404.08144*.

Fang, R., Bindu, R., Gupta, A., & Kang, D. (2024b). LLM Agents Can Autonomously Hack Websites. *arXiv preprint arXiv:2402.06664*.

Farzulla, M. (2025a). The Axiom of Consent: Friction Dynamics in Multi-Agent Coordination. *arXiv preprint arXiv:2601.06692*.

Farzulla, M. (2025b). Genre Mimicry vs. Ethical Reasoning in Abliterated Language Models. *Working Paper, Dissensus AI*.

Feffer, M., Sinha, A., Lipton, Z. C., & Heidari, H. (2024). Red-Teaming for Generative AI: Silver Bullet or Security Theater? *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*.

Ganguli, D., Lovitt, L., Kernion, J., Askell, A., Bai, Y., Kadavath, S., Mann, B., Perez, E., Schiefer, N., Ndousse, K., et al. (2022). Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned. *arXiv preprint arXiv:2209.07858*.

Ge, S., Zhou, C., Hou, R., Khabsa, M., Wang, Y.-C., Wang, Q., Han, J., & Mao, Y. (2023). MART: Improving LLM Safety with Multi-round Automatic Red-Teaming. *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

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

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

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.

Kong, H., Hu, D., Ge, J., Li, L., Li, T., & Wu, B. (2025). VulnBot: Autonomous Penetration Testing for a Multi-Agent Collaborative Framework. *arXiv preprint*.

Landolt, C. R., Würsch, C., Meier, R., Mermoud, A., & Jang, J. (2025). Multi-Agent Reinforcement Learning in Cybersecurity: From Fundamentals to Applications. *arXiv preprint*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33.

Longpre, S., et al. (2024). A Safe Harbor for AI Evaluation and Red Teaming. *Proceedings of the International Conference on Machine Learning (ICML)*.

Mei, Y., et al. (2025). AutoPen: Towards Autonomous Penetration Testing Using LLM-Powered Agents. *Proceedings of CSAE*.

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

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

MITRE. (2024b). MITRE D3FEND: A Knowledge Graph of Cybersecurity Countermeasures. https://d3fend.mitre.org/

Nakatani, S. (2025). RapidPen: Fully Automated IP-to-Shell Penetration Testing with LLM-based Agents. *arXiv preprint*.

National Institute of Standards and Technology. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). *NIST AI 100-1*.

Poomekum, P., Sorsoontornnirat, T., Suriyawong, A., Topiya, P., & Fugkeaw, S. (2026). Chimera-RL: An End-to-End Autonomous Red-Teaming Framework for LLM Applications. *IEEE Access*.

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

Shen, X., Wang, L., Li, Z., Chen, Y., Zhao, W., Sun, D., Wang, J., & Ruan, W. (2024). PentestAgent: Incorporating LLM Agents to Automated Penetration Testing. *Proceedings of the ACM Asia Conference on Computer and Communications Security*.

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

Singer, B., Lucas, K., Adiga, L., Jain, M., Bauer, L., & Sekar, V. (2025). Incalmo: An Autonomous LLM-assisted System for Red Teaming Multi-Host Networks. *arXiv preprint*.

Vyas, S., Mavroudis, V., & Burnap, P. (2025). Towards the Deployment of Realistic Autonomous Cyber Network Defence: A Systematic Review. *ACM Computing Surveys*.

Wan, S., Nikolaidis, C., Song, D., Molnar, D., et al. (2024). CYBERSECEVAL 3: Advancing the Evaluation of Cybersecurity Risks and Capabilities in Large Language Models. *arXiv preprint*.

Wang, Y., et al. (2025). Automated Penetration Testing with LLM Agents and Classical Planning. *arXiv preprint*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 35.

Xu, H., et al. (2024). RedAgent: Red Teaming Large Language Models with Context-aware Autonomous Language Agent. *arXiv preprint*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zhou, A., Wu, K., Pinto, F., Chen, Z., Zeng, Y., Yang, Y., Yang, S., Koyejo, O., Zou, J., & Li, B. (2025). AutoRedTeamer: Autonomous Red Teaming with Lifelong Attack Integration. *arXiv preprint arXiv:2503.17899*.

Zou, A., Wang, Z., Kolter, J. Z., & Fredrikson, M. (2023). Universal and Transferable Adversarial Attacks on Aligned Language Models. *arXiv preprint arXiv:2307.15043*.