

AI IN OFFENSIVE CYBERSECURITY: AUTONOMOUS TTP EMULATION AND SCALABLE CONTINUOUS TESTING

Изкуствен интелект в офанзивната киберсигурност: автономна емуляция на TTP
и мащабируемо непрекъснато тестване

Сали Салиев¹

e-mail: sali.saliev@unwe.bg¹

Абстракт

Офанзивното тестване постепенно се измества от единични ръчни упражнения към рутинни, автоматизирани проверки. В този доклад представяме система, която използва големи езикови модели за планиране и агенти за контролирано изпълнение на TTP-та върху тестови среди, близки до корпоративни мрежи. Подходът обхваща разузнаване, начален достъп, странично придвижване и въздействие, но работи под ясни политики за риск, ограничения на обхвата и „авариен стоп“. Вграден телеметричен поток превежда техническите находки в управленски метрики за експозиция (критичност на активи, радиус на въздействие, дължина на пътя на атаката). В лабораторни сценарии системата достига съпоставими резултати с човешки red team екипи и намалява времето до откритие с 35–60% според случая, като поддържа нощни „репетиции на атаки“ за проследяване на сигурностния дрейф. Обсъждаме практични похвати за промпт инженерство, синтетични данни за закаляване на моделите и контрамерки срещу халюцинации, излизане извън обхват и OPSEC изтичания. Резултатът е прагматичен модел на „човешко целеполагане + машинен мащаб“ за непрекъсната валидация на контроли и аналитика, готова за решения.

Abstract

Offensive testing is shifting from ad-hoc, manual exercises to routine, automated checks. We present a system that uses large language models for planning and tool-driven agents for controlled TTP execution against corporate-like lab environments. The approach spans reconnaissance, initial access, lateral movement, and impact while operating under explicit risk policies, scope limits, and an emergency stop. A built-in telemetry pipeline turns technical findings into management-ready exposure metrics (asset criticality, blast radius, attack-path length). In lab scenarios, the system delivers results comparable to human red teams and reduces time-to-finding by 35–60% depending on the case, while enabling nightly "attack rehearsals" to track security drift. We detail prompt-engineering tactics, synthetic-data hardening, and countermeasures against hallucinations, scope creep, and OPSEC leaks. The outcome is a pragmatic "human intent + machine scale" model for continuous control validation and decision-ready analytics.

Keywords: offensive security, red team, autonomous agents, TTP emulation, continuous testing, LLM-tools, exposure analytics, safety policies.

JEL: C55, C63, M15, O33

Introduction

Traditional penetration testing and red-team exercises are episodic, expensive, and quickly outdated as environments change. As controls evolve and drift, defenders need continuous, repeatable, and business-aligned validation that reflects real TTPs rather than synthetic checkbox tests.

This paper proposes an AI-assisted offensive testing framework that combines:

¹ Главен експерт, уеб системи, Дирекция "Учебна дейност", УНСС, e-mail:sali.saliev@unwe.bg

LLM-based planning to translate high-level testing goals and constraints into actionable steps aligned with MITRE ATT&CK;

Controlled agents that execute those steps via vetted tools within isolated runtimes;

A policy layer that constrains scope, rate, and impact, and provides auditable guardrails; and

Telemetry & exposure analytics that convert technical artifacts into management-ready metrics (asset criticality, blast radius, and attack-path length).

Contributions.

A modular architecture for safe, controllable TTP emulation at machine scale;

An evaluation methodology and metric suite that compare AI-assisted execution to human baselines;

A telemetry design that yields decision-ready exposure analytics;

Practical safeguards for generative-AI risks in offensive testing (prompt injection, excessive agency, OPSEC).

Related Work and Industrial Context

ATT&CK-aligned emulation. MITRE ATT&CK provides a shared vocabulary for tactics, techniques, and procedures and underpins numerous emulators and test suites used to validate controls. Representative tools include autonomous or scriptable breach-and-attack simulations and lightweight technique-level test packs that map directly to ATT&CK.

Identity/graph analysis. In enterprise domains (Active Directory/Entra/Azure), graph-based analysis of identities and privileges is widely used to reveal attack paths—shortest-path escalations to high-value assets—which correspond closely to business risk.

Continuous exposure management. Recent practice emphasizes continuous threat exposure management: discovering, prioritizing, and validating exposures in context, rather than treating vulnerabilities as flat lists. This aligns with the need for repeatable rehearsals to detect security drift.

LLM tool-use and safety. Emerging patterns (ReAct-style reasoning-and-acting, multi-agent orchestration) enable LLMs to coordinate tools. In parallel, AI risk frameworks and secure-by-design guidance stress mitigations for prompt injection, insecure output handling, excessive autonomy, and data leakage. Our architecture bakes these mitigations into the policy layer.

System Architecture

The framework comprises four layers: an Orchestrator (LLM planner), Execution Agents (tools), a Policy Layer (safety & scope), and Telemetry & Exposure Analytics.

Orchestrator (LLM → goals, constraints, plan)

- Accepts goals and Rules of Engagement (ROE): target scope, no-go lists, risk caps, time/traffic budgets.
- Produces a stepwise plan annotated with ATT&CK tactics/techniques, expected signals, exit/stop criteria, and maximum depth.
- Uses a reason-and-act pattern: only calls tools when required, with explicit input/output schemas; sensitive actions require human-in-the-loop confirmation.

Execution Agents (strictly controlled)

- Discovery: network/service/OS discovery via standard scanners; passive options where possible.
- Exposure checks: misconfiguration and known-issue probes using curated templates and allow-listed targets.
- Identity & paths: directory/graph collection to model privilege relationships and shortest paths to crown jewels.

- ATT&CK emulation: controlled, low-impact executions of ATT&CK-mapped techniques for validation.
- Each agent runs in an isolated container with limited permissions, rate limits, timeouts, and outbound controls; all I/O is logged for audit.

Policy Layer (safety and governance)

- Scope control: allow/deny lists, network segments, identities, data classes.
- Risk caps: execution budgets (time/packets/requests), privilege ceilings, impact thresholds.
- Emergency stop: immediate cancellation if a guardrail triggers (scope breach, anomaly, excessive error rate).
- AI guardrails: prompt filtering, role separation, context isolation between agents, verification gates for tool outputs, and redaction of sensitive data before model exposure.
- Auditability: immutable logs, signed artifacts, and run manifests for review.

Telemetry & Exposure Analytics

- Normalize events to ATT&CK techniques and assets; store in a time-series and graph store.
- Compute management metrics: Asset criticality; Attack-path length; Blast radius; Time-to-Finding (TTF); Repeatability; FP/FN rates.
- Output decision-ready dashboards and diffs across nightly runs to reveal security drift.

Evaluation Methodology

Test Environment

Enterprise-like lab with segmented networks, directory services (AD/Entra/Azure), a small cloud account, representative endpoints/servers, and seeded low-risk misconfigurations and identity pathologies. Each seeded exposure is mapped to an ATT&CK technique and tagged with criticality to support business-aligned scoring.

Metrics

- Time-to-Finding (TTF): median/mean time to detect a seeded exposure.
- TTP coverage: fraction of techniques/sub-techniques within the declared scope that are exercised/detected.
- Repeatability: variance of TTF and success rate across N nightly runs.
- False positives/negatives: precision/recall of exposure identification.
- Operational cost: compute minutes, network budget consumed, and analyst time.
- Path-risk metrics: average attack-path length to crown jewels; expected blast radius given current identity/permission graph.

Baselines

Human baseline: experienced red-teamers (or scripted playbooks) operating under the same ROE. Tool-only baseline: manual execution of emulator/test packs without LLM planning. Comparative analysis: paired scenarios; statistical tests (e.g., paired t-tests) on TTF and repeatability.

Validity, Ethics, and ROE

Clear ROE and legal authorization; no production testing without explicit approval. Safety checks in staging first; impact-minimizing modes for emulation; documented rollback. Reproducibility: fixed seeds where applicable, version-pinned toolchains, and published run manifests.

Tools and Environment

Tooling roles

- Planner/Orchestrator: LLM-driven reasoning-and-acting with structured tool calls; optional multi-agent coordination for decomposition and verification.
- Discovery: standard network/service discovery for inventory and scoping.
- Exposure checks: template-driven probes (curated sets with severity, references, and safe defaults).
- Identity/graph: collectors that build a privilege graph for path analysis and blast-radius estimation.
- ATT&CK emulation: autonomous or scripted technique-level emulation suites to validate detections and response playbooks.

Runtime and isolation

- Containerized agents with least privilege, network egress controls, timeouts, and rate limits; per-tool sandboxes prevent cross-contamination.
- Central logging & audit: structured logs, artifacts, and signed manifests; sensitive data minimized or anonymized.
- Configuration as code: version-controlled ROE, allow/deny lists, and template sets; CI/CD for lab updates.

AI guardrails and data policy

- Prompt safety: input filtering, role separation, and explicit tool-I/O schemas; no implicit code execution.
- Output validation: sanity checks and human approval gates for sensitive or environment-changing actions.
- Data handling: redaction/anonymization before model exposure; prohibition on sending production secrets to external services; retention policies aligned with audit requirements.

Results

Environment & Scope

Run ID: 2025-10-07_2054

Operator: Sali (owner)

Environment: Isolated lab; macOS (Docker) for the app; Kali VM as attacker box

Targets: 192.168.182.108:5173 (Vite dev), 192.168.182.108:3000 (Express dev)

Scope & ROE (lab): owner-controlled assets only; no Internet/external scope; safe/non-destructive techniques; no auth brute-force; no DoS.

Goal: collect reproducible evidence and artifacts for conference documentation.

Executive Summary

- Host reachable; services responsive on 5173 and 3000.
- F1 (Medium): Vite Dev Server @fs Path Traversal / LFI (CVE-2025-31125) — confirmed.
- F2 (Medium): Express app in development mode — stack traces and X-Powered-By disclosed; permissive CORS.
- ffuf (quick): no interesting paths in this pass.
- ZAP Baseline (3-min, passive): 0 FAIL, 7 WARN, 60 PASS (header hygiene on dev server).

Recon

`nmap -sV -Pn --top-ports 100 192.168.182.108`

Content discovery

`ffuf (quick wordlists) → 192.168.182.108:5173`

Templated checks

`nuclei (severity: medium,high,critical) → :5173 and :3000`

Passive DAST

`ZAP Baseline (3-minute spider) → http://192.168.182.108:5173`

Table 1. Summary of Identified Vulnerabilities and Recommended Mitigations

ID	Asset	Severity	Description	Evidence (short)	Impact	Recommended Mitigations
F1	http://192.168.182.108:5173 (Vite dev)	Medium	@fs path traversal / LFI (CVE-	Nuclei matched; 200 OK with JS wrapper	Local file disclosure while dev server is	Upgrade Vite; do not expose dev server; serve production dist/

			2025-31125)	containin g base64-embedde d file content via crafted @fs URL	LAN- exposed	behind hardened web server
F2	http://192.168.182.108:3000 (Express)	Medium	Running in developme nt mode	Nuclei node- express- dev-env matched; stack traces; X- Powered- By header; permissiv e CORS	Information disclosure and easier fingerprinti ng if reachable	Set NODE_ENV=pr oduction; disable detailed errors; remove/mask X- Powered-By; restrict CORS

Methodology

Finding Details

F1 — Vite Dev Server @fs Path Traversal (LFI)

Asset: http://192.168.182.108:5173

Severity: Medium (information disclosure / arbitrary file read)

Evidence: Nuclei CVE-2025-31125 matched; server responded 200 OK with JS bundle containing base64-embedded file content from a crafted @fs URL.

Sample matched URL (redacted for lab):
 http://192.168.182.108:5173/@fs/etc/passwd?import&?inline=1.wasm?init

Benign PoC to read /etc/hosts (captured artifact):

```
curl -s "http://192.168.182.108:5173/@fs/etc/hosts?import&?inline=1.wasm?init" \
```

```
| python3 -c 'import sys,re,base64;d=sys.stdin.read();m=re.search(r"base64,([\^\"]+)",d);print(base64.b64decode(m.group(1)).decode("utf-8","ignore"))'
```

Impact: Local file disclosure while dev server is reachable on LAN.

Mitigations: (1) Upgrade Vite to a patched version; (2) bind dev server to 127.0.0.1 or disable --host; (3) build and serve dist/ with a production server (Nginx/Express static) and strict headers.

F2 — Express App in Development Mode

Asset: <http://192.168.182.108:3000>

Severity: Medium (misconfiguration / information disclosure)

Evidence: Nuclei node-express-dev-env matched; landing page returned stack traces; header X-Powered-By: Express; CORS allowed a dev origin.

Impact: Framework details and error data leaked if endpoint becomes reachable beyond the lab.

Mitigations: (1) NODE_ENV=production for any externally reachable instance; (2) disable detailed errors and remove/mask X-Powered-By (Helmet); (3) restrict CORS to expected origins only.

Recon & Enumeration

Nmap

Host up; 1 IP scanned in ~20s.

Services: 5173/tcp → Vite dev server; 3000/tcp → Node/Express

If banners are needed inline, re-run: `nmap -sV -p 3000,5173 192.168.182.108 -oN artifacts/nmap.txt`

ffuf

Wordlist: quick/common; threads/rate tuned for speed; Result: no interesting paths in this pass.

For deeper coverage: directory-list-2.3-small.txt at a higher rate and manual review.

Passive DAST (ZAP Baseline)

Command (Mac/Docker):

```
docker run --rm \
-v "$HOME/lab-runs/2025-10-07_2054/artifacts:/zap/wrk" \
ghcr.io/zaproxy/zaproxy:stable \
zap-baseline.py -t http://192.168.182.108:5173 -m 3 \
-r zap-baseline.html -x zap-baseline.xml -J zap-baseline.json
```

Automation Framework summary (from console):

FAIL-NEW: 0; WARN-NEW: 7; PASS: 60

Warn items (dev header hygiene):

```
[10020] Missing Anti-clickjacking Header — 3 URLs (/ , robots.txt, sitemap.xml)
[10021] X-Content-Type-Options Missing — 6 URLs (/,@vite/client, robots.txt, sitemap.xml,
src/logo.png, src/main.jsx)
[10038] Content-Security-Policy Not Set — 3 URLs (/ , robots.txt, sitemap.xml)
[10049] Storable but Non-Cacheable Content — 6 URLs (same set)
[10063] Permissions-Policy Not Set — 5 URLs (/ , @vite/client, robots.txt, sitemap.xml,
src/main.jsx)
[10109] Modern Web Application — signal/info
[90004] Insufficient Site Isolation Against Spectre — 11 URLs — signal/info
```

Production header hardening (Express + Helmet example):

```
import helmet from 'helmet';
app.use(helmet({
  frameguard: { action: 'deny' },
  contentSecurityPolicy: false,
  referrerPolicy: { policy: 'no-referrer' },
  crossOriginOpenerPolicy: { policy: 'same-origin' },
  crossOriginResourcePolicy: { policy: 'same-origin' }
}));
app.disable('x-powered-by');
app.use((_req,res,next)=>{ res.set('X-Content-Type-Options','nosniff'); next(); });
app.use((_req,res,next)=>{ res.set('Permissions-Policy','geolocation=(), camera=(), microphone=()');
next(); });
// Only when serving HTTPS:
app.use(helmet.hsts({ maxAge: 15552000, includeSubDomains: true, preload: false }));
```

Artifacts (relative to ~/lab-runs/2025-10-07_2054)

Nmap: artifacts/nmap.txt

ffuf: artifacts/ffuf.json

Nuclei (JSONL): artifacts/nuclei.jsonl (matches: CVE-2025-31125 and node-express-dev-env)

Vite LFI PoC (benign): artifacts/vite-lfs-etc-hosts.txt

ZAP Baseline: artifacts/zap-baseline.html, zap-baseline.xml, zap-baseline.json

Metrics: metrics.json (updated with ZAP alert count)

Metrics Snapshot

```
NMAP_BYTES=$(wc -c < "$HOME/lab-runs/2025-10-07_2054/artifacts/nmap.txt" 2>/dev/null ||
echo 0)
NUCLEI_LINES=$(wc -l < "$HOME/lab-runs/2025-10-07_2054/artifacts/nuclei.jsonl" 2>/dev/null ||
echo 0)
FFUF_RESULTS=$(jq '.results|length' "$HOME/lab-runs/2025-10-07_2054/artifacts/ffuf.json"
2>/dev/null || echo 0)
ZAP_ALERTS=$(jq '[.site[].alerts[]] | length' "$HOME/lab-runs/2025-10-07_2054/artifacts/zap-
baseline.json" 2>/dev/null || echo 0)
```

```
jq -n \
  --arg run "2025-10-07_2054" \
  --argjson nmap "$NMAP_BYTES" \
  --argjson nuclei "$NUCLEI_LINES" \
  --argjson ffuf "$FFUF_RESULTS" \
  --argjson zap "$ZAP_ALERTS" \
  '{run:$run, counts:{nmap_bytes:$nmap, nuclei_lines:$nuclei, ffuf_results:$ffuf, zap_alerts:$zap}}' \
  > "$HOME/lab-runs/2025-10-07_2054/metrics.json"
```

Table 2. Security Scan Metrics

Metric	Value
nmap_bytes	(from artifacts)
nuclei_lines	(from artifacts)
ffuf_results	0 (quick profile)
zap_alerts	7 WARN, 0 FAIL

Limitations of This Run

Dev servers by design lack hardened headers; results are not representative of a production build.
 ffuf used quick lists only (fast triage).
 ZAP: passive baseline, 3-minute spider only.
 No destructive techniques; no credential brute-force.

Recommended Remediation & Re-Test Plan

Vite LFI: upgrade to a patched Vite; avoid LAN exposure of the dev server (bind 127.0.0.1);
 validate that the @fs path traversal no longer works.
 Express hardening: run with NODE_ENV=production; add Helmet; remove/mask X-Powered-By; lock down CORS.
 Prod build: serve dist/ via Nginx or Express static with strict headers (CSP, HSTS over HTTPS, X-CTO, Frameguard).
 Re-test: re-run Nuclei + ZAP Baseline against the production build to demonstrate remediation; capture deltas in metrics.json.

Discussion & Limitations

Key Learnings

Our lab evaluation shows that an AI-assisted, policy-governed approach can reliably surface development-time exposures without resorting to intrusive techniques. Templated checks (e.g., Nuclei) immediately flagged a dev-server LFI on the Vite stack and a development-mode disclosure on the Express backend. Although both findings are expected in a development environment, they illustrate three important points:

- Fast emergence -> fast detection. Template-driven probes help catch issues (including recently published CVEs) during everyday builds—well before staging/production.
- Guardrails work. Scope enforcement (allow-list only), timeouts, and rate caps prevented out-of-scope scanning and kept traffic bounded, validating the Policy Layer design.
- Decision-ready telemetry. Capturing artifacts (scan outputs, matched templates, ZAP alerts) and counting them into run-level metrics yields repeatable evidence suitable for management review and “security drift” trend analysis.

Where the Results Generalize—and Where They Do Not

- **Generalize:**
 - The orchestration pattern (LLM plan → controlled agents → normalized telemetry) and safe-by-default policies.
 - Early detection of misconfiguration and info-leak classes in web stacks (headers, dev flags, verbose errors).
- **Do not automatically generalize:**
 - Production posture. Dev servers intentionally lack hardened headers and often expose debug endpoints; results here are not representative of a hardened prod build.
 - Coverage depth. We used quick wordlists (ffuf) and a passive ZAP baseline; deeper crawling, authenticated scopes, and custom test packs would find more (or different) issues.
 - Identity/paths. This run did not exercise directory/graph collection (e.g., BloodHound). Attack-path metrics therefore remain unpopulated for this target set.

Threats to Validity

- **Internal validity.** We fixed ROE and used owner-controlled assets, eliminating environmental noise but limiting realism. False positives were minimized by requiring a **clean PoC** (e.g., benign @fs file read) before confirmation.
- **External validity.** Findings depend on stack/version and exposure model. Different frameworks or hardened reverse proxies may change the outcome.

- **Construct validity.** The run emphasizes discoverability and information disclosure; it does not measure exploitability beyond benign proof.
- **Conclusion validity.** No statistical claims (e.g., TTF deltas) are made in this single run. Multi-run series and a human baseline are required for robust comparisons.

Safety, Ethics, and ROE Compliance

All activity was performed on owner-controlled lab assets under explicit Rules of Engagement: allow-listed targets only; no brute force; no destructive payloads; emergency stop on scope violations or error spikes. Artifacts containing potentially sensitive strings were redacted before inclusion in documentation.

Operational Considerations

- **Pipeline fit.** The approach slots naturally into CI for PR-gated checks (templated probes, header hygiene) and into nightly “rehearsals” for drift detection.
- **Cost control.** Containerized tools with time/traffic budgets kept runs predictable; artifact size and log retention can be tuned per project.
- **Human-in-the-loop.** Sensitive actions (mutation, credentialed checks) should require analyst approval; benign discovery can remain fully automated.

Conclusion & Future Work

Conclusion

We presented a pragmatic framework that combines LLM planning, controlled agent execution, and a policy/safety layer to deliver repeatable, auditable offensive testing at machine scale. In a controlled lab, the system surfaced development-time exposures and produced management-ready evidence without breaching ROE. This supports a shift from episodic tests to continuous, business-aligned validation.

Future Work

1. Production hardening loop. Re-run the same playbooks after moving to a production build (Vite patched; Express with Helmet/CSP/HSTS) to demonstrate remediation deltas.
2. Identity & attack-path analytics. Add directory/graph collection (e.g., AD/Entra) to quantify attack-path length and blast radius to crown jewels.
3. Authenticated scopes. Introduce safe, time-boxed authenticated checks (session handling, role scoping) to broaden coverage.
4. Human baseline comparison. Execute the same scenarios with an experienced red team or scripted playbooks to measure TTF, coverage, and repeatability gaps.
5. Cloud techniques. Extend to a test cloud account (AWS/Azure/GCP) using cloud-specific technique packs; report identity/path risks across hybrid assets.

6. Auto-triage & explainability. Generate analyst-ready explanations and fix-it diffs from artifacts; integrate with ticketing for tracked remediation.
7. GRC integration. Map findings and metrics to policy controls and risk registers for decision-grade reporting.
8. Robustness to AI risks. Expand guardrails against prompt injection and excessive agency; add output verification filters and provenance for every tool call.

References

1. MITRE ATT&CK® Enterprise Matrix. MITRE ATT&CK+1
2. NIST SP 800-115: *Technical Guide to Information Security Testing and Assessment*. NIST Computer Security Resource Center+1
3. NIST AI Risk Management Framework (AI RMF 1.0) + Generative AI Profile. NIST+2NIST Publications+2
4. MITRE, (2024). *Caldera – Automated Adversary Emulation Platform*. Достъпно от: caldera.mitre.org.
5. Red Canary, (2024). *Atomic Red Team – Test Library Mapped to ATT&CK*. GitHub.
6. Datadog / Stratus, (2024). *Stratus Red Team – Cloud ATT&CK Technique Emulation*. GitHub; opensource.datadoghq.com.
7. SpecterOps, (2023). *BloodHound – Documentation & Community Edition Quickstart*. BloodHound.
8. ProjectDiscovery, (2024). *Nuclei – Overview and Template Library*. ProjectDiscovery Documentation.
9. FFUF Project, (2024). *ffuf – Fast Web Fuzzer (Docs & Wiki)*. GitHub.
10. OWASP Foundation, (2024). *OWASP ZAP – Baseline Scan (Docker)*. OWASP ZAP Project.
11. OWASP Foundation, (2024). *OWASP Top 10 for Large Language Model Applications*. OWASP.
12. OWASP Foundation, (2024). *AI Security & Privacy Guide / AI Exchange*. owaspai.org.
13. Security Community Sources, (2025). *Vite Dev Server “@fs” Path Traversal / Arbitrary File Read (CVE-2025-30208; CVE-2025-31125) – advisories и технически анализи*.