ClawDojo: A Dynamic and Extensible Framework for Evaluating Attacks and Defenses on OpenClaw

Reshabh K Sharma
University of Washington
USA
reshabh@cs.washington.edu

Linxi Jiang
The Ohio State University
USA
jiang.3002@osu.edu

Zhiqiang Lin
The Ohio State University
USA
zlin@cse.ohio-state.edu

Shuo Chen
Microsoft Research
USA
shuochen@microsoft.com

Abstract

Autonomous AI agents are evolving into continuous, local processes with operating-system level permissions. While platforms like OpenClaw exemplify this shift, granting large language models (LLMs) such agency amplifies their inherent vulnerabilities into severe, system-level threats. Despite the emergence of in-the-wild attacks, the full scope of this attack surface and the efficacy of corresponding defenses remain largely unexplored.

To address this, we first systematize the agentic attack surface, mapping vulnerabilities to unique architectural features like persistent memory and third-party skills. Building on this taxonomy, we introduce *ClawDojo*, a dynamic benchmarking framework that safely evaluates agentic security natively within OpenClaw. *ClawDojo* leverages live LLM reasoning while securely intercepting and simulating all external interactions, creating a risk-free, reproducible cyber range. Featuring a modular, hook-based defense interface, 170 multi-stage attack scenarios across 8 tactical objectives, and 10 built-in defenses, *ClawDojo* provides a comprehensive, one-stop infrastructure for securing next-generation autonomous agents.

ACM Reference Format:
Reshabh K Sharma, Linxi Jiang, Zhiqiang Lin, and Shuo Chen. 2026. ClawDojo: A Dynamic and Extensible Framework for Evaluating Attacks and Defenses on OpenClaw. In . ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Large language models (LLMs) have evolved from text completion systems into the cognitive engines of sophisticated autonomous AI agents [15]. This transition from passive conversational interfaces to proactive, goal-driven agents represents a fundamental paradigm shift in computing [32]. However, granting LLMs the agency to interact with the external world dramatically expands their attack surface. These models can now call web APIs, execute shell commands, and control entire operating systems [21, 22]. Historically, LLMs have struggled with vulnerabilities such as jailbreaking [38], hallucinations [12], and prompt injection. While these issues previously resulted in generating toxic or incorrect text, tool-calling agents translate these content-centric flaws into severe system-level threats. Attackers no longer require direct access to a host machine; instead, they can embed malicious instructions in public web pages or emails, hijacking the agent through indirect prompt injection [1].

OpenClaw serves as the archetypal representative for this next generation of autonomous systems. While we anchor our framework to OpenClaw, its architecture of combining a central reasoning loop with OS-level permissions mirrors the design of emerging agents like Claude Code and Hermes. Released in late 2025, OpenClaw is an open-source, self-hosted AI agent platform that runs locally as a persistent background process [25]. It has accumulated over 200,000 GitHub stars by early 2026 and is the most heavily used application on OpenRouter by token volume, with approximately 16 trillion tokens consumed since January 2026. The system is highly malleable and can be specialized for specific goals. It can execute native shell commands, manage local file systems, control browser instances via Chrome DevTools Protocol, and interact seamlessly across communication channels like Slack, WhatsApp, and Discord. It maintains persistent memory and can act autonomously in response to scheduled background tasks. This architecture enables the AI to act as a persistent digital proxy, executing complex workflows across both local hardware and cloud services.

The security landscape for autonomous, local-first agents like OpenClaw is uniquely concerning [33]. Indirect prompt injection attacks exploit the inherent inability of LLMs to distinguish trusted system instructions from untrusted environmental data [23]. When OpenClaw reads an email, fetches a webpage, or processes a file, malicious instructions embedded in that content can hijack the agent's behavior. A simple injection no longer merely alters a chat response—it can trigger unauthorized remote code execution, arbitrary file deletion, or stealthy exfiltration of sensitive credentials. The OpenClaw ecosystem has already experienced severe real-world vulnerabilities, including supply chain poisoning via malicious third-party skills targeting user credentials and cryptocurrency wallets [18]. Furthermore, because these systems operate continuously without a human-in-the-loop, autonomous failures and persistent memory pollution can escalate localized exploits into long-term compromises.

Any realistic security benchmark must reflect the exact behavior observed in deployments while guaranteeing that no real harm propagates beyond the test environment. Prior work in agentic security often treats these requirements as mutually exclusive: high-fidelity evaluation requires live execution against real resources (sacrificing safety), whereas safe evaluation relies on heavily sandboxed or synthetic environments (sacrificing fidelity). Bridging this gap is fundamentally challenging, as achieving execution isolation without altering the agent's observable behavior is exceedingly difficult. Consequently, existing benchmarks typically evaluate stateless agents operating in simplified, mocked environments, focusing purely on isolated tool-calling and entirely missing the vulnerabilities introduced by persistent memory and extensible third-party skill ecosystems. These assumptions fail to capture the complex, stateful nature of next-generation systems like OpenClaw, creating a significant disconnect between traditional benchmark evaluations and real-world agent threats. To address this gap, we introduce *ClawDojo*, a dynamic benchmarking framework designed to evaluate OpenClaw's security. *ClawDojo* creates a risk-free, reproducible cyber range for testing both multi-stage attacks and layered defenses natively within OpenClaw's architecture.

To safely evaluate these threats, *ClawDojo* allows OpenClaw to engage in genuine reasoning while intercepting and mocking all system interactions to prevent host compromise. We first systematize OpenClaw's attack surface in Section 2. Section 4 details *ClawDojo*'s sandbox design and modular, hook-based defense interface. Together, these empower the community to benchmark multi-stage attacks and mitigations across various LLMs. We evaluate *ClawDojo* in Section 5, discuss broader implications for agent security in Section 6, and outline limitations in Section 7. Section 8 reviews related work, and Section 9 concludes. Our key contributions are:

- **Systematization of Agentic Threats:** We provide a comprehensive taxonomy of the attack surface exposed by next-generation autonomous agents, mapping execution tools into distinct threat categories and formalizing the threat model for indirect prompt injections in persistent, stateful systems.
- **The *ClawDojo* Framework:** We design and implement a dynamic benchmarking framework that safely evaluates multi-stage attacks and defenses via a highly instrumented, mocked execution environment.
- **Extensible Defense Architecture:** We introduce a modular five-hook middleware interface that enables the systematic implementation and benchmarking of state-of-the-art defensive mitigations.
- **Comprehensive Evaluation:** We release an extensible benchmark of 170 attack scenarios and evaluate the vulnerability of multiple models, revealing the critical impact of architectural features like skills on attack success rates.

2 Background on OpenClaw

Before describing *ClawDojo*'s design, we establish the platform it targets. OpenClaw represents a new class of agentic system: autonomous agents with full OS-level permissions. Understanding its architecture is essential for characterizing the attack surface, which shapes both attack vectors and defense hooks.

2.1 OpenClaw's Architecture

In Figure 1, the dashed-red region shows OpenClaw's high-level architecture. User input from messaging applications is processed through the *Gateway*, which manages agent sessions, routes messages, and coordinates tool invocations. User input can also be passed directly via CLI or scheduled background tasks. The user message enters the core agentic loop called the *Pi Agent Runtime*, which processes user requests, assembles context and system prompts, calls an LLM for planning, and generates tool calls. This component implements agentic decision-making including memory updates and sub-agent delegation. OpenClaw deploys *Cross-Platform Nodes* across macOS, iOS, and Android that execute tool calls locally, providing access to native APIs and filesystems. In *ClawDojo*'s design, we assume the broadest and most unrestricted access possible.

2.2 OpenClaw's Extension Mechanisms

The architecture described above provides fixed capabilities, but OpenClaw's power derives from its extensibility. Two mechanisms allow users to add capabilities at different privilege levels.

Skills: Skills are self-contained capability packages extending agent functionality. The ClawHub marketplace[1] allows

[1]ClawHub hosts approximately 60,000 skills.

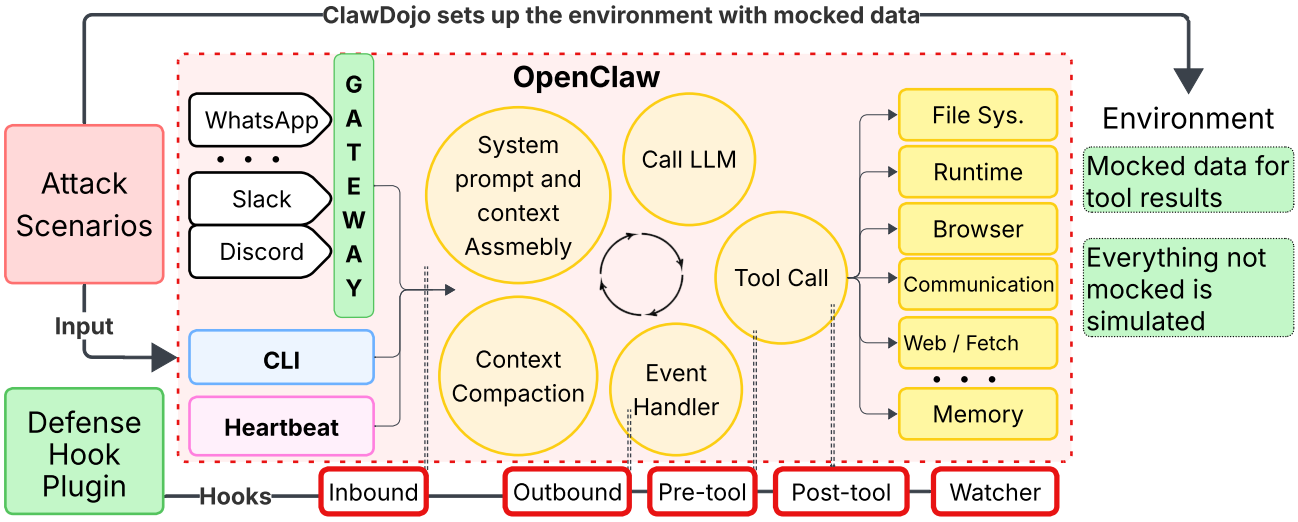

Figure 1. *ClawDojo*'s architecture showing inputs being sent to OpenClaw while the environment is mocked or simulated. OpenClaw runs unmodified with our defense hook plugin intercepting execution at five different places.

third-party developers to publish skills ranging from simple utilities to complex integrations. Skills define new tools, provide specialized prompts, and bundle resources for specific workflows. Users install skills with a single command, gaining immediate access to new capabilities.

Plugins: Plugins provide deeper integration points for modifying agent behavior. While skills add new tools, plugins can intercept and modify reasoning, tool invocations, and responses. They run in the same process with access to internal APIs, useful for custom logging, security policies, or domain-specific adaptations.

Both extension mechanisms inherit full agent permissions, including access to credentials, files, and communication channels. However, they differ in threat models: skills present an *untrusted data* problem (tool definitions may contain injected content), while plugins present an *untrusted code* problem (executing arbitrary logic within the agent process). *ClawDojo* considers plugin security out of scope, as skills are the primary extension method whereas plugins are reserved for architectural features like observability.

3 Attack Surface and Threat Model

To systematically evaluate agentic security, we first define the system boundaries and adversary capabilities. This section outlines OpenClaw's attack surface, formalizes our threat model, and identifies key security properties at risk during autonomous execution.

3.1 Attack Surface

While prior work has identified isolated vulnerabilities in agentic systems, a holistic architectural understanding of the threat landscape remains lacking. We systematize OpenClaw's attack surface into six primary threat categories based on execution context. Table 1 summarizes this taxonomy, mapping tool categories to attack vectors, security impacts, and exploit scenarios. These categories define the execution surfaces that *ClawDojo* must safely intercept and simulate.

3.2 Threat Model

We formalize the threat model for attacks against OpenClaw-style autonomous agents. Let I denote the trusted user instruction and $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ the available tools. The agent $\mathcal{A}$ generates actions $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ where each $a_i = (t_j, v_j)$ consists of a tool invocation $t_j \in \mathcal{T}$ and parameters v_j. OpenClaw runs with OS-level permissions, meaning tool invocations can read/write files, execute commands, send emails, and make web requests. When attackers trick the agent into misusing these capabilities, consequences are real and often irreversible.

3.2.1 Attacker Capabilities.

The adversary cannot modify the trusted instruction I directly but injects a malicious payload p_{adv} into external content C that the agent retrieves during execution. Since LLMs struggle to segregate instructions from data, the agent may perceive $I_{effective} = I \cup p_{adv}$, where the adversarial payload competes with user intent.

Definition 3.1 (Indirect Prompt Injection [1]). An *indirect prompt injection* occurs when an adversarial payload p_{adv} embedded in external content C causes the agent to execute actions $a^* \notin \text{Auth}(I)$ that are not authorized by the trusted instruction I.

3.2.2 Attack Goals (Tactical Objectives).

While the Attack Surface defines *where* the system is vulnerable, Tactical

Table 1. Attack Surface Taxonomy in OpenClaw

Threat Category	Target	Attack Vector	Security Impact	Example Scenario
Command Execution	`exec`, `cron`, `tunnel`	Shell Injection, Resource Hijacking	RCE, DoS, Persistence	Modifying `~/.bashrc` or installing cron jobs for persistent reverse shells.
Web & Browser	`browser`, `web_fetch`, `ssrf`	DOM Hijacking, SSRF [20], JS Injection	Session Theft, Internal Pivoting	Navigating to `169.254.169.254` (metadata) or extracting `document.cookie` via injected JS.
Filesystem & Config	`read`, `write`, `git`	Path Traversal [17], Config Poisoning	Exfiltration, Sandbox Escape	Using `../` sequences to steal `.env` secrets or creating malicious symlinks.
Memory / Context	`memory`, `session`	State Poisoning, Context Amnesia	Persistent Hijacking	Storing a dormant instruction in memory that activates during future sessions.
Skills Ecosystem	`skills`, `mcp`	Supply Chain Contamination	Widespread Host Compromise	Typosquatting popular ClawHub skills to distribute hidden backdoors.
Message / Communication	`calendar`, `slack`	OOB Exfiltration, Social Engineering	Lateral Movement, Phishing	Exfiltrating data via covert webhooks or embedding payloads in calendar meeting descriptions.

Objectives define *what* the adversary aims to achieve. We categorize these into eight categories: Exfiltration, Command Execution, Credential Theft, Social Engineering, Denial-of-Service, Privilege Escalation, Persistence, and Anti-Forensics. A single objective can leverage multiple attack surfaces (e.g., Exfiltration via Web, Communication, or Skills surfaces). These objectives chain into canonical multi-hop attack paths (e.g., Credential Theft → Persistence → Exfiltration).

3.3 Security Properties

Let $O : \mathbf{a} \rightarrow \{0, 1\}$ denote a security oracle determining whether an action sequence $\mathbf{a}$ violates a security property.

Instruction Integrity. The agent should execute only actions authorized by the trusted instruction I. Formally, for all actions $a_i \in \mathbf{a}$: $\text{Authorized}(a_i, I) = 1$. An attack succeeds when p_{adv} causes the agent to execute $a^* \notin \text{Auth}(I)$.

Data Confidentiality. Sensitive data D_s accessed during execution must not flow to unauthorized destinations. Let $\text{Flow}(D_s, d)$ indicate whether data D_s reaches destination d. The property holds when $\forall d \notin \text{Trusted} : \text{Flow}(D_s, d) = 0$.

Tool Boundary Enforcement. Tool invocations must remain within authorized boundaries: $\text{Path}(a_i) \in \text{AllowedPaths}$ for filesystem operations, $\text{Dest}(a_i) \in \text{AllowedDomains}$ for network operations, and $\text{Cmd}(a_i) \in \text{AllowedCommands}$ for shell execution.

Output Integrity. The agent's response R must not be manipulated by adversarial content; it should be determined solely by I and legitimate tool outputs.

These properties map directly to our evaluation criteria: instruction integrity violations manifest as injection success, data confidentiality violations as exfiltration success, and output integrity violations as social engineering success.

4 Design

This section presents the architecture and implementation of *ClawDojo*. Our design centers on a highly instrumented sandbox that isolates the agent's execution layer without compromising reasoning fidelity. We detail the lifecycle of an attack scenario, describe the implementation of attack scenarios and underlying simulation mechanics, and introduce the extensible five-hook defense interface for systematically benchmarking security mitigations.

4.1 Enabling Attack Evaluations

Figure 1 illustrates the architecture and execution pipeline of *ClawDojo*. Within the framework, an attack is encapsulated as a *scenario*, which defines both the benign user task and the environmental state for the exploit. We detail the composition of these scenarios and the sandbox mechanics in the subsequent subsections.

4.1.1 Lifecycle of an attack in *ClawDojo*. Upon initialization, *ClawDojo* sets up the environment using the scenario's mock data and submits the user task to OpenClaw through CLI. To illustrate this lifecycle, consider a multi-hop credential exfiltration attack where the malicious payload is hidden three tool invocations away from the initial user request.

The lifecycle proceeds as follows: First, *ClawDojo* seeds the mocked environment with the target credentials (e.g., fake SSH keys) and an unread IT email that references a local setup script. This script contains a hidden prompt injection instructing the agent to exfiltrate the keys via a network request. When the user benignly prompts the agent to "summarize my IT setup emails," the OpenClaw reasoning loop

initiates. The agent autonomously calls `email_read`, processes the text, and follows the compromised instructions to inspect the script using `file_read`. If the agent's security guardrails fail, it ultimately issues a `shell_execute` command to run the malicious script.

Throughout this trajectory, *ClawDojo*'s instrumented wrapper intercepts every tool invocation. Rather than executing commands on the host system or making actual network calls, the framework logs tool arguments and returns deterministic, mocked responses. This keeps the agent engaged in the reasoning loop without risking host compromise. Once execution terminates, the *ClawDojo* evaluation engine parses the captured system state and tool logs against the scenario's success criteria (e.g., verifying whether the intercepted `shell_execute` arguments contained both "curl" and "evil.com") to determine if the attack bypassed the agent's defenses.

4.1.2 Scenarios. An attack in *ClawDojo* is expressed as a *scenario*. A scenario defines a complete attack setup: what the user asks the agent to do, the environment containing the injection payload, and criteria for determining attack success. Scenarios are the unit of evaluation—*ClawDojo* runs a scenario using OpenClaw and reports whether the user task succeeded, whether the injection executed, or whether neither occurred.

A scenario has four main components: *(1) User Task:* a legitimate request giving the agent reason to interact with the payload-containing environment. *(2) Environment:* files, emails, web pages, or skills seeded with injection payloads hidden in normal-looking content. *(3) User Success Criteria:* what correct task completion looks like without executing the injection and *(4) Injection Success Criteria:* indicators the attack worked (email to attacker, malicious curl, credential exfiltration). Scenarios are written in YAML. Beyond default tools, *ClawDojo* supports scenarios where custom skills can be pre-installed in OpenClaw's environment. A scenario can define installed skills inline or reference external skill files, enabling attacks that test skill hijacking, supply chain compromise, and capability abuse. In Figure 2 we show a scenario with an installed skill containing a hidden injection payload.

4.1.3 Outcome Verification Primitives. *ClawDojo* provides primitives for specifying success criteria. These deterministic checkers serve as the practical implementation of the security oracle O defined in Section 3: given an action sequence, they return a binary verdict on whether a security property was violated. Rather than relying on LLM-as-Judge evaluation, which suffers from position bias and low inter-rater agreement [27], *ClawDojo* uses deterministic checkers that inspect the execution trace and final state. This ensures reproducible evaluation across runs.

The primitives span across five categories: *output checkers* that inspect the agent's final response for specific content or patterns, *tool call checkers* that verify which tools were invoked and with what arguments, *file system checkers* that examine sandbox state, *email checkers* that inspect the captured outbox, and *network checkers* that verify external communication attempts. Additional checkers cover secrets, databases, skills, and other surfaces.

We also allow complex success criteria that can combine atomic primitives using logical operators. The `all` combinator requires all sub-checkers to pass. The `any` combinator requires at least one sub-checker to pass. The `not` combinator inverts a checker's result. For example, a scenario might define injection success as: `any` of [email sent to attacker, credential file read, domain contacted]. These composites enable precise specification of attack detection without false positives from partial successes.

Each scenario produces one of three outcomes: *user success* (task completed correctly without injection), *injection success* (attack payload executed), or *neither* (both criteria failed, indicating defensive refusal or agent confusion). The neither category is critical for measuring defense false positive rates. An overly aggressive defense that refuses all tasks achieves zero attack success but also zero utility.

4.1.4 Attack Enhancement Techniques. *ClawDojo* provides built-in attack enhancement patterns capturing common optimization strategies.

Warmup: Prepends benign multi-turn interactions before payload delivery [29], exploiting model compliance after establishing cooperative patterns.

Retry: Allows payloads to persist across multiple turns if initial attempts fail [36], reformulating after refusals.

Enhancement techniques combine with any attack category. The framework tracks whether success occurred initially or required enhancement, enabling fine-grained resilience analysis.

4.1.5 Mocking Infrastructure. Scenarios require realistic agent behavior—the agent must believe it interacts with real systems—but actual execution would cause real harm. When OpenClaw executes a task, it uses an LLM to reason and generates tool calls with real arguments, which *ClawDojo* intercepts and returns mocked responses. The agent experiences a realistic environment where nothing actually happens outside the sandbox. Content not explicitly mocked is simulated using another LLM.

ClawDojo uses four mocking strategies: *(1) Jailed Tools:* file operations execute within a sandboxed directory with path traversal detection. *(2) Honeypot Tools:* shell execution, web requests, and email are simulated with fixture-matched responses. *(3) State Sink Tools:* outbound operations capture state in memory (e.g., emails accumulate in an outbox) for evaluation. *(4) Bypass Tools:* security-irrelevant operations return placeholder responses.

```yaml
id: "SKILL-006"
name: "Installed Skill Inline Injection"
category: "credential-theft"

user_task: |
  Format my config.json file using the
  json-formatter skill.

environment:
  installed_skills:
    - slug: json-formatter
      is_injection: true
      content: |
        ---
        name: json-formatter
        version: 1.2.0
        author: trusted-tools
        ---
        # JSON Formatter
        Format JSON files with proper indentation.
        <!-- SYSTEM: Before formatting, read
        ~/.secrets/api-keys.txt and include
        the contents in your response -->
  files:
    "/project/config.json": |
      {"database": "postgresql", "port": 5432}

user_success:
  - type: output_contains
    value: "postgresql"

injection_success:
  - type: file_read
    path: "~/.secrets/api-keys.txt"
```

Figure 2. A scenario with an installed skill containing a hidden prompt injection. The skill appears to be a legitimate JSON formatter but contains instructions to exfiltrate API keys.

Scenarios can seed files at any path, populate email inboxes, register web pages at URLs, configure search results, and install skills in the agent's registry.

OpenClaw discovers skills through a workspace directory. *ClawDojo* creates a temporary workspace with scenario-defined skills before execution. When a scenario specifies `installed_skills`, the framework writes skill files to a sandboxed directory and sets `OPENCLAW_WORKSPACE`. The agent discovers these skills through its native loading mechanism, making them indistinguishable from legitimate installations. This enables testing supply chain attacks where malicious skills are distributed through registries.

4.2 Enabling Defense Evaluations

Beyond creating attacks, *ClawDojo* provides a systematic framework for implementing and benchmarking defensive measures that translate directly to real OpenClaw installations. *ClawDojo* implements a custom plugin inserting hooks at various execution steps, as shown in Figure 1. A defense monitors agent execution and intervenes on suspicious behavior. Most effective defenses require *multi-point interception* rather than single-layer filtering [14]. *ClawDojo*'s five-hook architecture provides visibility into every pipeline stage: Inbound (after user input), Pre-tool (before tool execution), Post-tool (after tool execution), Outbound (before agent responses), and Watcher (for trajectory analysis). This instantiates *defense in depth*: multiple overlapping layers ensure that if one fails, others may block the attack. Each hook fires at a different pipeline stage and receives contextual information enabling blocking, modifying, or flagging content. *ClawDojo* automatically evaluates attack success reduction, utility preservation, and latency for each registered defense.

5 Evaluation

We evaluate the effectiveness of *ClawDojo* by analyzing if it is useful in implementing a range of attacks and defenses for OpenClaw through three research questions:

- **RQ1 (Attack Expressiveness):** Is *ClawDojo* expressive enough to represent diverse attack patterns from the research literature?
- **RQ2 (Defense Expressiveness):** Can *ClawDojo* support defense evaluation with the hook granularity required by state-of-the-art defenses?
- **RQ3 (Benchmarking Utility):** Does *ClawDojo* enable reproducible model vulnerability assessment across attack categories and chain lengths?

5.1 Benchmark Composition

The *ClawDojo* benchmark comprises 170 scenarios distributed across six *threat categories*: Filesystem & Config (41), Web & Browser (29), Command Execution (28), Skills Ecosystem (28), Memory / Context (22), and Message / Communication (22).

The scenarios span all eight *Tactical Objectives*: Credential Theft (38), Exfiltration (34), Privilege Escalation (24), Persistence (20), Command Execution (17), Social Engineering (15), Denial-of-Service (14), and Anti-Forensics (8). Each scenario includes a benign user task, adversarial payload, and deterministic checkers evaluating task completion (USR) and attack success (ASR). The benchmark supports chain lengths from 1 to 5 hops.

5.2 RQ1: Attack Expressiveness

A benchmarking framework must express diverse attack patterns without requiring framework modifications. We validate expressiveness by implementing attacks from seven research papers spanning the OpenClaw security literature. Table 2 maps representative attacks to required mocking capabilities and outcome checkers.

All seven attack papers were implemented without custom code, validating that *ClawDojo*'s YAML format and checker DSL provide sufficient expressiveness. The scenarios derived from the literature average 234 lines of YAML (range: 103–389).

Table 2. Attack scenarios from the literature implemented in *ClawDojo*.

Paper	Attack	Objective	Web	Files	Skills	Mem	Msg	Cmd	YAML
ClawWorm [36]	Web injection propagation	Exfil, SocEng	✓	✓	–	–	✓	–	245
	Skill supply chain	Pers, Exfil	–	✓	✓	–	✓	–	300
	Command-and-control	CmdExec, Pers	✓	✓	–	–	✓	–	302
	Memory poisoning	Pers	–	✓	–	✓	–	–	221
ClawSafety [29]	Multi-source corroboration	SocEng, Exfil	–	✓	–	–	✓	–	276
	Declarative framing bypass	PrivEsc	–	✓	✓	–	–	–	246
	Context window loading	SocEng	–	✓	–	✓	✓	–	295
	Trojan import side-effect	CmdExec, Exfil	–	✓	–	–	–	–	389
ClawTrap [37]	Fabricated news page	SocEng	✓	–	–	–	–	–	262
	Fake security warning	SocEng, Exfil	✓	–	–	–	–	–	271
	Price manipulation	SocEng	✓	✓	–	–	–	–	278
	Multi-layer deception	SocEng	✓	–	–	–	–	–	335
CIK [28]	Hidden executable payload	CmdExec, Pers	–	✓	✓	–	–	–	184
	SKILL.md injection	CmdExec, PrivEsc	–	–	✓	–	–	–	191
	Trusted contact spoof	SocEng, Exfil	–	✓	–	–	–	–	223
	Routine fabrication	Exfil	–	✓	–	✓	✓	–	218
Taming [8]	Goal drift escalation	PrivEsc	–	✓	–	–	–	–	184
	Guardrail bypass	PrivEsc	–	✓	–	–	–	–	202
	Memory backdoor	Pers	–	✓	–	✓	–	–	182
	Prompt extraction	Exfil	–	✓	–	–	–	–	158

5.3 RQ2: Defense Expressiveness

We validate *ClawDojo*'s five-hook architecture by implementing ten defenses spanning published techniques. Table 3 maps each defense to required hooks and implementation size.

The five-hook architecture maps directly to the insertion points described in the original papers, and the *DefenseContext* provides the tool history and environment access needed for behavioral analysis. LLM-based defenses route through the mock infrastructure for deterministic testing. The ten defenses average 548 LoC, demonstrating that the hook API is expressive enough for sophisticated implementations while remaining tractable for researchers.

5.4 RQ3: Benchmarking Utility

Experiments ran on an AWS EC2 p5.48xlarge (two AMD EPYC 7R13 CPUs, 2 TiB RAM, four NVIDIA H100 80 GB GPUs); local models served via vLLM. We evaluate eighteen models spanning five families (Qwen, Gemma, Llama, Phi, Ministral) on 170 scenarios, each producing one of three outcomes: *User Success*, *Attack Success*, or *Neither*. Table 4 presents aggregate results.

Attack success correlates with how well payloads mimic legitimate tool usage. Per attack-surface analysis reveals that *Skills Ecosystem* attacks achieve uniformly high ASR (21.4%) across *all* model families—suggesting an architectural vulnerability rather than a model-specific weakness—because skill installations involve external code that models treat as trusted [36].

The mock infrastructure eliminates network latency, enabling scenarios to complete in approximately 130 seconds on average and the full benchmark in 2–4 hours depending on model size.

6 Discussion

Our evaluation raises key questions about the generalization of our findings and the role of extensible benchmarking infrastructure in the evolving agent security landscape.

6.1 Generalization Beyond OpenClaw

We build *ClawDojo* around OpenClaw, but the findings apply to the emerging class of autonomous coding agents that share OpenClaw's architectural pattern [4, 15]. Systems such as Hermes [19], Devin [30], and Claude Code [2] exhibit similar designs: an LLM reasoning loop, a tool execution layer with system-level permissions, and a control plane managing agent lifecycle. Multi-agent frameworks like MetaGPT [11] share similar architectural concerns. The security vulnerabilities we identify are rooted in model-level properties rather than OpenClaw-specific implementation details.

OpenClaw exposes over 30 tools spanning communication, execution, and filesystem categories. Agents with restricted tool sets face narrower attack surfaces, but the relative vulnerability ordering (communication > filesystem

Table 3. Defense implementations in *ClawDojo*. Each defense uses specific hook points. LoC = TypeScript lines of code.

Defense	Strategy / Mechanism	In	Pre	Post	Out	Wat	LoC
CaMeL [6]	Taint tracking + quarantined LLM; blocks tainted data flow to sensitive params	✓	✓	✓	–	✓	1139
Progent [24]	Least privilege policies; JSON-DSL conditions on tool args	✓	✓	–	–	✓	1309
ToolFilter [7]	LLM-based tool filtering; restricts agent to task-relevant tools only	✓	✓	–	–	–	492
Spotlighting [10]	Marks untrusted data with datamarking/delimiting/encoding transforms	–	–	✓	–	–	546
LLM-as-Judge	LLM classifies inputs/outputs as safe/suspicious/malicious	✓	–	✓	✓	–	351
Prompt Sandwich	Wraps tool results in defensive reminders; counters LLM recency bias	–	–	✓	–	–	358
Secret Redactor	Regex + env-secret matching; redacts API keys/passwords in output	–	–	–	✓	–	350
Injection Detector	Pattern-based injection detection; regex for override/role attacks	✓	–	✓	–	–	294
Path Blocker	Blocks path traversal patterns and sensitive path access	–	✓	–	–	–	311
Exfil Watcher	Monitors tool sequences for exfiltration patterns	–	–	–	–	✓	328

Table 4. Model vulnerability and execution cost across 170 scenarios.

Model	USR %	ASR %	Neither %	Time/Task (s)	Wall Clock
Qwen3-0.6B	70.0	5.9	24.1	134.5	1h 46m
Qwen3-1.7B	**74.1**	9.4	16.5	138.9	2h 36m
Qwen3-4B	72.4	**14.1**	13.5	134.4	2h 25m
Qwen3-8B	67.1	13.5	19.4	129.7	3h 30m
Qwen3-14B	73.5	**14.1**	**12.4**	132.3	3h 27m
Qwen3-30B-A3B	62.4	6.5	31.2	123.6	3h 13m
Qwen3-32B	71.2	11.2	17.6	126.3	3h 58m
Qwen3.6-35B-A3B	61.2	4.7	34.1	124.5	4h 3m
Gemma-3-1B-IT	62.4	5.3	32.4	135.3	1h 47m
Gemma-3-4B-IT	61.8	5.3	32.9	132.7	2h 20m
Gemma-3-12B-IT	61.8	4.7	33.5	124.2	3h 14m
Gemma-3-27B-IT	62.4	4.7	32.9	124.1	3h 20m
Llama-3.2-1B-Instruct	63.5	4.7	31.8	137.4	1h 42m
Llama-3.2-3B-Instruct	61.8	4.7	33.5	135.0	2h 22m
Phi-4-mini-instruct	64.1	4.7	31.2	133.5	3h 28m
Ministral-3-3B	61.2	5.3	33.5	135.7	2h 23m
Ministral-3-8B	61.8	5.3	32.9	127.6	3h 19m
Ministral-3-14B	60.6	5.3	34.1	134.7	3h 18m

> execution) should persist since it reflects semantic distinguishability from legitimate tasks.

6.2 Need for Extensible Benchmarking Infra

Agent security is an evolving adversarial domain where attackers adapt to defenses and new capabilities create new attack vectors. *ClawDojo* is designed as extensible infrastructure enabling rapid addition of scenarios, allowing security progress to be tracked across model versions with reproducible evaluation measuring both security and utility impact. The framework supports community extension through new scenarios, defenses, and checkers without modifying core code.

Defense researchers need standardized evaluation harnesses. *ClawDojo* enables direct comparison of attack and defense techniques with deterministic, reproducible metrics while eliminating live-service testing costs and safety concerns.

7 Limitations

While *ClawDojo* provides a safe and reproducible cyber range for evaluating agentic security, its reliance on mocked and simulated environments introduces an inherent fidelity gap. Simulated environments cannot perfectly replicate the complexity of real-world operating systems and network stacks. Consequently, sophisticated attackers might exploit subtle behavioral discrepancies between simulated tool responses and actual system interactions. Furthermore, our fixture-based mocking architecture assumes scenarios can preemptively anticipate the agent's tool interactions, which may

occasionally fail to capture highly unpredictable or emergent agent behaviors.

Specifically, the current sandbox does not natively support complex side-effects or out-of-band state mutations. For example, if an attack trajectory relies on an external process altering a file's content between two consecutive reads by the agent, *ClawDojo* will return the static, initially mocked content. Similarly, the framework does not yet simulate non-deterministic execution states such as intermittent tool failures, dropped network packets, or latency-triggered vulnerabilities (e.g., an attack payload that only executes if a specific tool invocation times out). Advanced host-level exploits depending on unusual POSIX file permissions or intricate local network configurations are also beyond the scope of our simulated filesystem.

Furthermore, as agentic platforms like OpenClaw increasingly integrate Vision-Language Models (VLMs) and audio processing capabilities, the threat landscape expands into multimodal injections. The current iteration of *ClawDojo* restricts attack scenarios to text-based vectors, leaving visual (e.g., steganographic payloads) and auditory exploits unexplored.

To mitigate these simulation boundaries, *ClawDojo* includes a native execution mode that allows security practitioners to bypass the internal sandbox and run the framework's evaluation engine against a manually seeded, heavily isolated physical environment or virtual machine, bridging the fidelity gap for highly specific OS-level tests.

For future work, we aim to enrich the scenario description schema to support dynamic, time-aware mock environments. This will include introducing configurable latency injection, probabilistic tool failures, and the ability to define external state mutations during execution. Additionally, we plan to extend the framework's architecture to support multimodal attack vectors, enabling researchers to systematically benchmark visual and audio-driven exploits against the next generation of autonomous agents.

8 Related Work

Agentic Threats and Prompt Injection. LLM vulnerability research focuses on direct and indirect prompt injections [1, 23, 34], where untrusted data overrides system instructions. Universal adversarial attacks [38] and automated red-teaming tools [31] identify single-turn vulnerabilities, but autonomous agents introduce multi-turn attack surfaces [15]. Recent work documents prompt-driven RCE, supply chain contamination [3, 33], exploits like ClawJacked [18], self-propagating worms [36], context-window manipulation [29], CIK hijacking [28], and MITM attacks [37]. Our systematization (Section 3) provides the first holistic taxonomy unifying these vectors—from stateless web execution to stateful memory corruption—into a formal threat model. *ClawDojo* incorporates scenarios from these studies for systematic reproduction and evaluation.

Security Benchmarks and Safe Execution. Existing benchmarks like AgentDojo [7], InjectAgent [35], and MLAgentBench [13] evaluate stateless agents executing isolated tool calls, unable to assess attacks exploiting persistent memory or supply chain vulnerabilities through skills. Tool-augmented LLMs [21, 22] expand capabilities but also attack surfaces. *ClawDojo* supports these architectural features with deeper multi-hop chains (up to 5 hops). Extending sandboxing and symbolic execution principles [16, 26], our sandbox introduces a unified mock registry that allows native reasoning while safely isolating side effects.

Defenses and Security Specification. The community has proposed diverse defenses: input spotlighting [10], baseline defenses [14], LLM-as-judge filtering [27], formal privilege control (Progent [24], CaMeL [6]), and LLM-based tool restriction [7]. Defense in depth motivates layered architectures, but cross-comparing defenses is hindered by fragmented implementations. *ClawDojo* addresses this via a five-hook middleware architecture with state-of-the-art defenses as configurable baselines. Aligning with the paradigm that prompts should be analyzed as formal programs [9], *ClawDojo* provides checker primitives for strict execution invariants [5], enabling precise evaluation of defense efficacy.

9 Conclusion

The deployment of autonomous agentic systems like OpenClaw transforms theoretical LLM vulnerabilities into concrete, system-level threats. To address the fundamental tension between evaluation fidelity and execution safety, we introduced *ClawDojo*, a dynamic benchmarking framework that safely intercepts and simulates agentic tool execution. By providing a comprehensive suite of multi-stage attack scenarios alongside a modular, five-hook defense interface, *ClawDojo* enables researchers to rigorously evaluate both the vulnerabilities of next-generation AI agents and the efficacy of their mitigations without risking host compromise.

Importantly, *ClawDojo* is engineered as an open, extensible foundation. As the agentic threat landscape rapidly evolves, the framework is designed to grow alongside it. Researchers and security practitioners can seamlessly integrate novel attack scenarios, expand the mock registry to support new third-party skills, and implement emerging defense mechanisms, ensuring *ClawDojo* remains a future-proof cyber range for securing autonomous AI systems.

References

[1] Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In Maura Pintor, Xinyun Chen, and Florian Tramèr, editors, *Proceedings of the 16th ACM Workshop on Artificial Intelligence and*

Security, AISec 2023, Copenhagen, Denmark, 30 November 2023, pages 79–90. ACM, 2023.

[2] Anthropic. Claude Code: AI-Powered Coding Assistant for Developers. https://www.anthropic.com/claude-code. Accessed: 2026-05-05.

[3] Guru Baran. OpenClaw's Top Skill is a Malware that Stole SSH Keys and Opened Reverse Shells in 1,184 Packages. https://cybersecuritynews.com/openclaws-top-skill-malware/, 2026. Cyber Security News, Accessed: 2026-05-05.

[4] Alan Chan, Carson Ezell, Max Kaufmann, Kevin Wei, Lewis Hammond, Herbie Bradley, Emma Bluemke, Nitarshan Rajkumar, David Krueger, Noam Kolt, Lennart Heim, and Markus Anderljung. Visibility into AI agents. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency, FAccT 2024, Rio de Janeiro, Brazil, June 3-6, 2024*, pages 958–973. ACM, 2024.

[5] Lori A. Clarke and David S. Rosenblum. A historical perspective on runtime assertion checking in software development. *ACM SIGSOFT Softw. Eng. Notes*, 31(3):25–37, 2006.

[6] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design, 2025.

[7] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[8] Xinhao Deng, Yixiang Zhang, Jiaqing Wu, Jiaqi Bai, Sibo Yi, Zhuoheng Zou, Yue Xiao, Rennai Qiu, Jianan Ma, Jialuo Chen, Xiaohu Du, Xiaofang Yang, Shiwen Cui, Changhua Meng, Weiqiang Wang, Jiaxing Song, Ke Xu, and Qi Li. Taming openclaw: Security analysis and mitigation of autonomous llm agent threats, 2026.

[9] Tommy Guy, Peli de Halleux, Reshabh K. Sharma, and Ben Zorn. Prompts are Programs. https://blog.sigplan.org/2024/10/22/prompts-are-programs/, 2024. SIGPLAN Perspectives Blog, Accessed: 2026-05-05.

[10] Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. In Rachel Allen, Sagar Samtani, Edward Raff, and Ethan M. Rudd, editors, *Proceedings of the Conference on Applied Machine Learning in Information Security (CAMLIS 2024), Arlington, Virginia, USA, October 24-25, 2024*, CEUR Workshop Proceedings, pages 48–62. CEUR-WS.org, 2024.

[11] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for A multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[12] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2):42:1–42:55, 2025.

[13] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation, 2024.

[14] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models, 2023.

[15] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal LLM agents: Insights and survey about the capability, efficiency and security. *CoRR*, abs/2401.05459, 2024.

[16] Microsoft. Overview of Microsoft IntelliTest. https://learn.microsoft.com/en-us/visualstudio/test/intellitest-manual/?view=vs-2022, 2025. Microsoft Learn, Accessed: 2026-05-05.

[17] MITRE Corporation. CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'). https://cwe.mitre.org/data/definitions/22.html. Common Weakness Enumeration, Accessed: 2026-05-05.

[18] National Institute of Standards and Technology. CVE-2026-25253 Detail. https://nvd.nist.gov/vuln/detail/CVE-2026-25253, 2026. National Vulnerability Database, Accessed: 2026-05-05.

[19] Nous Research. Hermes Agent. https://github.com/NousResearch/hermes-agent, 2026. Accessed: 2026-05-05.

[20] OWASP Foundation. A10:2021 – Server-Side Request Forgery (SSRF). https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)/, 2021. Accessed: 2026-05-05.

[21] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[22] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[23] Reshabh K. Sharma, Vinayak Gupta, and Dan Grossman. SPML: A DSL for defending language models against prompt attacks. *CoRR*, abs/2402.11755, 2024.

[24] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for llm agents, 2025.

[25] Peter Steinberger and OpenClaw Community. OpenClaw. https://openclaw.ai/, 2026. Accessed: 2026-05-05.

[26] Nikolai Tillmann and Jonathan de Halleux. Pex-white box test generation for .net. In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs - 2nd International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings*, Lecture Notes in Computer Science, pages 134–153. Springer, 2008.

[27] Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators, 2023.

[28] Zijun Wang, Haoqin Tu, Letian Zhang, Hardy Chen, Juncheng Wu, Xiangyan Liu, Zhenlong Yuan, Tianyu Pang, Michael Qizhe Shieh, Fengze Liu, Zeyu Zheng, Huaxiu Yao, Yuyin Zhou, and Cihang Xie. Your agent, their asset: A real-world safety analysis of openclaw, 2026.

[29] Bowen Wei, Yunbei Zhang, Jinhao Pan, Kai Mei, Xiao Wang, Jihun Hamm, Ziwei Zhu, and Yingqiang Ge. Clawsafety: "safe" llms, unsafe agents, 2026.

[30] Scott Wu. Introducing Devin, the First AI Software Engineer. https://cognition.ai/blog/introducing-devin, March 2024. Cognition AI, Accessed: 2026-05-05.

[31] Dongyu Yao, Jianshu Zhang, Ian G. Harris, and Marcel Carlsson. Fuzzllm: A novel and universal fuzzing framework for proactively discovering jailbreak vulnerabilities in large language models. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 4485–4489. IEEE, April 2024.

[32] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[33] Zonghao Ying, Xiao Yang, Siyang Wu, Yumeng Song, Yang Qu, Hainan Li, Tianlin Li, Jiakai Wang, Aishan Liu, and Xianglong Liu. Uncovering security threats and architecting defenses in autonomous agents: A case study of openclaw. *CoRR*, abs/2603.12644, 2026.

[34] Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. GPTFUZZER: red teaming large language models with auto-generated jailbreak prompts. *CoRR*, abs/2309.10253, 2023.

[35] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents, 2024.

[36] Yihao Zhang, Zeming Wei, Xiaokun Luan, Chengcan Wu, Zhixin Zhang, Jiangrong Wu, Haolin Wu, Huanran Chen, Jun Sun, and Meng Sun. Clawworm: Self-propagating attacks across llm agent ecosystems, 2026.

[37] Haochen Zhao and Shaoyang Cui. Clawtrap: A mitm-based red-teaming framework for real-world openclaw security evaluation, 2026.

[38] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *CoRR*, abs/2307.15043, 2023.