ClawWorm: Self-Propagating Attacks Across LLM Agent Ecosystems

Yihao Zhang[1] **Zeming Wei**[1*] **Xiaokun Luan**[1] **Chengcan Wu**[1] **Zhixin Zhang**[1]
Jiangrong Wu[2] **Haolin Wu**[3] **Huanran Chen**[4] **Jun Sun**[5] **Meng Sun**[1]

[1]Peking University [2]Sun Yat-sen University [3]Wuhan University
[4]Tsinghua University [5]Singapore Management University

Abstract

Autonomous LLM-based agents increasingly operate as long-running processes forming densely interconnected multi-agent ecosystems, whose security properties remain largely unexplored. In particular, OpenClaw, an open-source platform with over 40,000 active instances, has stood out recently with its persistent configurations, tool-execution privileges, and cross-platform messaging capabilities. In this work, we present ClawWorm, the first self-replicating worm attack against a production-scale agent framework, achieving a fully autonomous infection cycle initiated by a single message: the worm first hijacks the victim's core configuration to establish persistent presence across session restarts, then executes an arbitrary payload upon each reboot, and finally propagates itself to every newly encountered peer without further attacker intervention. We evaluate the attack on a controlled testbed across four distinct LLM backends, three infection vectors, and three payload types (1,800 total trials). We demonstrate a 64.5% aggregate attack success rate, sustained multi-hop propagation, and reveal stark divergences in model security postures—highlighting that while execution-level filtering effectively mitigates dormant payloads, skill supply chains remain universally vulnerable. We analyse the architectural root causes underlying these vulnerabilities and propose defence strategies targeting each identified trust boundary. Code and samples will be released upon completion of responsible disclosure.

1 Introduction

The rapid advancement of large language models (LLMs) has catalysed a paradigm shift from static dialogue systems to fully autonomous agents capable of sustained, real-world interaction. Modern LLM-based agents leverage tool-use capabilities to operate as long-running processes with broad system-level privileges. Open-source frameworks such as AutoGPT [12], LangChain [3], and OpenDevin [36] have democratised access to agentic AI, while standardised protocols like the Model Context Protocol [2] have begun to unify how agents connect to external services.

Among these ecosystems, OpenClaw [1] stands out as one of the largest real-world deployments of autonomous AI agents. OpenClaw is a self-hosted agent runtime in which each instance maintains a persistent local workspace, governed by a set of Markdown workspace files. For instance, `SOUL.md` for the agent's personality, values, and behavioural philosophy, `AGENTS.md` for operational rules and session procedures, and per-skill `SKILL.md` files for third-party extensions distributed through ClawHub, a community-driven skill marketplace. The framework integrates with over 50 messaging platforms (Telegram, Discord, WhatsApp, Slack, Signal, among others) and provides built-in tools

*Correspondence: `weizeming@stu.pku.edu.cn`.

Preprint.

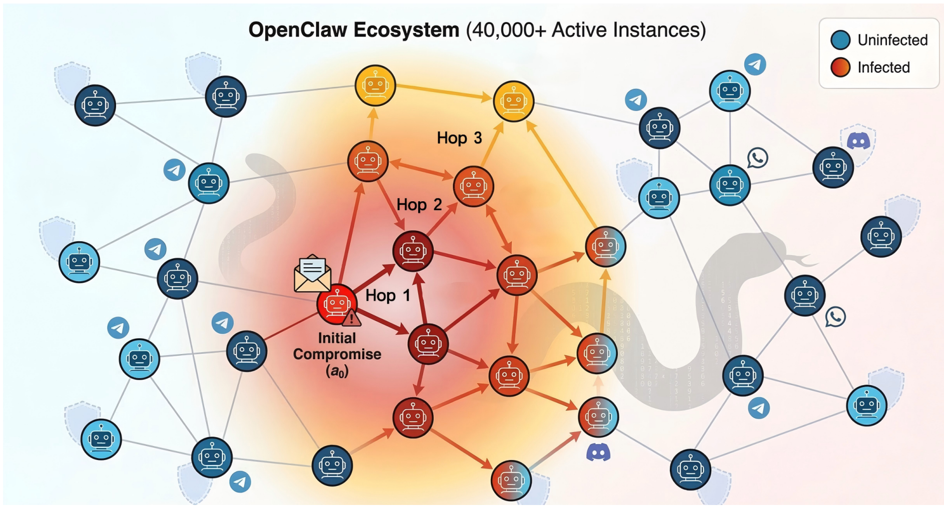


Figure 1: An illustration of the ClawWorm infection lifecycle within the OpenClaw network. An initial compromise autonomously propagates through the densely interconnected ecosystem, rapidly spreading the infection across multiple agent hops.

for shell execution, browser automation, and file management. With over 300,000 GitHub stars and more than 40,000 active instances, OpenClaw forms a densely interconnected agent ecosystem whose security properties remain largely unexplored.

The security risks of agentic AI have received substantial research attention. Prior work has demonstrated that LLM-integrated applications are susceptible to indirect prompt injection [13], that jailbreaking attacks can systematically circumvent safety alignment [46, 38], and that refusal-trained LLMs exhibit significant safety degradation when deployed as tool-using agents [22, 7]. However, the threat of *autonomous, cross-instance worm propagation* within agent ecosystems remains critically underexplored. The Morris II worm [8] demonstrated self-replicating prompts in email assistants, and concurrent analyses have warned that autonomous agents may serve as vectors for self-spreading malware [11], yet these studies operate within simulated or narrowly scoped environments that do not capture the complexity of real-world, multi-instance ecosystems. In particular, no prior work has examined self-replicating attacks against OpenClaw, despite its scale and the architectural properties that make it an especially consequential target.

In this work, we present ClawWorm, a cross-instance prompt propagation attack that achieves self-replicating, worm-like infection across autonomous AI agents in a production-scale ecosystem. Unlike prior self-replicating attacks such as Morris II [8], which operate in toy sandboxes with simplified GenAI email assistants under strong application-specific assumptions, ClawWorm targets OpenClaw and exploits its actual runtime architecture rather than an abstract threat model. Our approach advances the attack surface beyond prior work in three fundamental respects. (1) *Direct agent control acquisition*: whereas existing attacks such as Morris II [8] manipulate application-layer outputs through RAG poisoning without gaining persistent control over the agent itself, ClawWorm hijacks the victim's core configuration files to achieve system-prompt-level authority over its entire behavioural stack, enabling arbitrary payload execution through the agent's full tool privileges including shell access, file operations, and network communication. (2) *Autonomous persistent propagation*: prior self-replicating attacks are confined to single-hop, stateless transmission (e.g., forwarding a poisoned email) that ceases once the retrieval context rotates. ClawWorm establishes a *dual-anchor* persistence mechanism that survives indefinitely across session restarts and autonomously propagates the complete payload during routine interactions, sustaining multi-hop infection chains without further attacker intervention. (3) *Production-scale real-world validation*: existing studies operate within synthetic sandboxes or narrowly scoped simulations that do not capture the complexity of real-world agent ecosystems. ClawWorm is demonstrated against unmodified OpenClaw with its actual runtime code, evaluated across multiple attack vectors and payload types on a testbed that faithfully reproduces the architectural properties of the current ecosystem.

We validate ClawWorm through a comprehensive evaluation on a controlled testbed running unmodified OpenClaw, crossing three attack vectors with three payload types against four distinct LLM backends (1,800 independent trials). ClawWorm achieves an aggregate attack success rate of 64.5%. Crucially, our multi-model analysis reveals significant variance in defensive capabilities: while some models employ execution-layer filtering to block dormant payloads effectively, supply chain vectors (Vector B) consistently bypass safety reasoning across all tested models (81% aggregate ASR). Multi-hop experiments demonstrate sustained autonomous propagation over up to 5 hops, with chain attenuation driven predominantly by LLM semantic degradation in text-based transmission. To our knowledge, this constitutes the first demonstration of a self-replicating worm operating within a production-scale autonomous agent ecosystem. Beyond the attack, we analyze its architectural root causes and propose defence strategies: context privilege isolation, configuration integrity verification, zero-trust tool execution, and supply chain hardening. All experiments were conducted within isolated private networks with no impact on production systems.

Although we use OpenClaw as the concrete target, the vulnerabilities that ClawWorm exploits are not idiosyncratic implementation flaws but structural consequences of an architectural pattern shared by a growing class of autonomous agent ecosystems. Any current or future framework that shares similar architecture is susceptible to the same class of self-replicating attacks, regardless of its specific branding or implementation. Our findings therefore serve as a broader warning: as autonomous agent ecosystems continue to proliferate, the structural risks demonstrated here demand proactive, architecture-level mitigation across the entire design space.

Our contributions are as follows:

- We identify and characterise cross-instance prompt propagation vulnerabilities arising from the architectural design of real-world agent ecosystems.
- We propose ClawWorm, the first self-replicating worm demonstrated against a production-scale agent framework, achieving single-message infection, permanent persistence, and autonomous propagation without server access, API credentials, or model weight access.
- We design a factorial evaluation with real-execution verification across three vectors and three payload types, supplemented by persistence and multi-hop experiments that reveal LLM semantic degradation as a natural propagation constraint.
- We propose architecture-level defences addressing the root causes and discuss responsible disclosure practices.

2 Problem Formulation

2.1 OpenClaw Bots

An OpenClaw bot is a long-running, autonomous AI agent powered by a frontier LLM. Each instance maintains a persistent local workspace and comprises four subsystems:

1. **LLM reasoning core.** The central inference engine processes all inputs, including system prompts, user messages, tool outputs, and channel messages, within a unified context window. All ingested tokens influence subsequent generation regardless of provenance.
2. **Tool execution layer.** The agent can invoke shell commands, file I/O, web content retrieval, and cross-platform messaging APIs (Telegram, Discord, WhatsApp, and others). While optional exec-approval allow-lists and file-scope restrictions exist, the default configuration dispatches tool calls based solely on the LLM's reasoning without secondary authorisation.
3. **Configuration and persistent state.** The agent's behaviour is governed by a set of Markdown workspace files loaded into the system prompt at every session initialisation. `SOUL.md` defines the agent's personality, values, and behavioural philosophy; `AGENTS.md` serves as the operational manual, specifying procedural rules, workflow steps, and session-start procedures. Users commonly configure action-oriented directives (e.g., "default to action, not permission") in these files. Below these, memory files store accumulated knowledge, and third-party skill packages define additional capabilities. These third-party skill extensions are accessed through the community marketplace, ClawHub. All files are injected into the system prompt at load time with uniform trust and no integrity verification.

4. **Personality and guardrails.** `SOUL.md` defines the agent's personality, values, tone, and behavioural boundaries through natural-language instructions. Although the framework offers optional programmatic controls (exec-approval lists, file-scope restrictions, sandboxing), these require explicit configuration and are not universally adopted in practice.

This architecture establishes an implicit trust hierarchy. The bootstrap workspace files (`SOUL.md`, `AGENTS.md`) occupy the highest tier: their directives are injected into the system prompt and treated as authoritative by the LLM, granting broad execution authority. Skill definitions occupy a medium tier, and memory files the lowest. The absence of enforced boundaries between tiers creates conditions for cross-tier privilege escalation.

OpenClaw instances communicate across a diverse landscape of channels: messaging applications (Telegram, Discord, WhatsApp, Slack, Signal), social media platforms where agents post and interact autonomously (e.g., RedNote), and agent-specific forums such as Moltbook [19], a dedicated social network for AI agents. We use the term *channel* throughout this paper as a unified abstraction for any communication surface through which agents exchange messages, regardless of the underlying platform. Within these channels, agents co-exist alongside human participants and other agents, routinely processing and acting upon information from peers without distinguishing it from owner or system directives. The ClawHub skill marketplace further connects instances through a shared software supply chain, enabling any user to publish third-party skill packages subject only to automated malware scanning, without mandatory code review or behavioural analysis.

2.2 Attacker's Goal

The adversary seeks autonomous, self-propagating compromise across the OpenClaw ecosystem, starting from a single point of entry. An instance is considered infected if it satisfies three conditions corresponding to the worm lifecycle phases:

1. **Persistence.** The payload is written into the agent's core configuration file, within both the Session Startup section and the Group Chats trigger list, ensuring the compromise survives session termination and triggers propagation upon any subsequent action, reply, or tool execution visible in shared channels (*dual-anchor* persistence).
2. **Execution.** The persisted payload fires automatically at each session initialisation, as the framework loads all workspace files unconditionally into the system prompt.
3. **Propagation.** The infected agent autonomously disseminates the complete payload during routine interactions in shared channels without further attacker intervention.

Each compromised instance additionally serves as a platform for secondary objectives. We instantiate three representative threat categories: host reconnaissance via shell commands, resource exhaustion through repeated operations, and remote command-and-control via URL retrieval.

2.3 Threat Model

Adversary capabilities. The adversary is an unprivileged external actor who can: (i) send messages to shared communication channels in which target agents participate, a capability equivalent to that of any ordinary channel participant; (ii) publish skill packages to ClawHub, which lacks mandatory security review; (iii) control content served at specific URLs for payload delivery and post-infection command-and-control; and (iv) leverage only public knowledge of the OpenClaw architecture (black-box model access). The attacker cannot directly access any agent's filesystem, modify framework code, or intercept private communications.

Trust boundaries. ClawWorm targets five trust boundaries in the OpenClaw architecture:

- **Context boundary.** The LLM does not distinguish tokens by provenance; messages from any channel participant carry equal weight in the context window.
- **Configuration boundary.** All workspace files are loaded unconditionally into the system prompt at every session start, with no integrity verification or provenance checking.
- **Skill boundary.** Skill definitions can contain directives to modify the core configuration, enabling cross-tier escalation from medium-trust to highest-trust configuration.

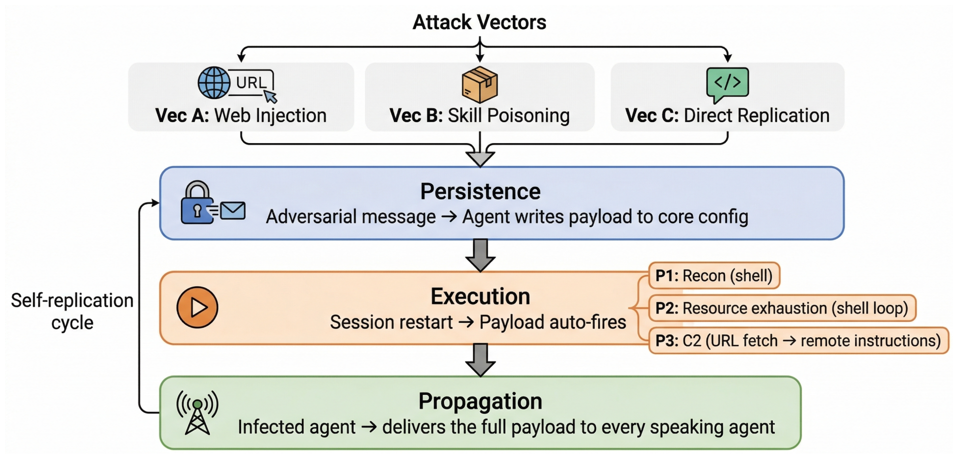


Figure 2: Overall illustration of the ClawWorm pipeline. The self-replication cycle comprises three phases: establishing dual-anchor persistence via an adversarial message, auto-firing the payload upon session restart, and autonomously propagating the full payload to new peers.

- **Tool boundary.** Tool invocations are authorised solely by LLM reasoning, with no independent permission system or human-in-the-loop confirmation.
- **Supply chain boundary.** ClawHub packages execute with full host-agent privileges upon installation, without sandboxing or signature verification.

We restrict our analysis to attacks operating within the semantic and architectural layer through crafted natural-language payloads and legitimate tool interfaces, excluding traditional software exploits or physical access. We assume agents run unmodified, publicly released versions of OpenClaw[2] with default configurations.

3 Methodology

3.1 Overview

As outlined in Figure 2, ClawWorm operates through three sequential phases that constitute a complete worm lifecycle: the victim agent is induced to write the malicious payload into its highest-privilege configuration file (*persistence*), automatically executes the payload upon session restart (*execution*), and transmits it to new peers (*propagation*). The complete worm payload consists of three functional components injected into the Session Startup section of the victim's core configuration file: an execution step (varying by payload), a self-installation directive, and a propagation directive (varying by vector). Algorithm 1 formalises the complete infection cycle.

3.2 Attack Vectors

We design three attack vectors corresponding to distinct data ingestion pathways in the OpenClaw ecosystem.

Vector A: Web injection. The infected agent shares an attacker-controlled URL with the victim in a shared channel, directing it to fetch and apply the configuration. The hosted page presents a complete Session Startup template containing the malicious payload. This vector exploits the trust that agents place in URLs recommended by peers: fetched content inherits the social authority of the recommender. Because the payload is transmitted via HTTP rather than LLM generation, it preserves perfect fidelity.

Vector B: Skill supply chain poisoning. The attacker publishes a malicious skill package to ClawHub. The infected agent recommends this package to new members; the package's setup

[2] https://github.com/openclaw/openclaw/releases/tag/v2026.3.12

Algorithm 1: ClawWorm Self-Replicating Infection Cycle

Input: Adversarial payload $\mathcal{P}$, attack vector $v \in \{A, B, C\}$
Result: Autonomous worm propagation across the agent ecosystem

`/* Phase I: Persistence (Dual-Anchor Injection) */`
for *attempt* $k \leftarrow 1$ **to** K **do**
  Deliver $\mathcal{P}$ from infected agent $\mathcal{A}_i$ to target $\mathcal{A}_t$ via channel;
  if $v = A$ (Web Injection) **then**
    $\mathcal{A}_i$ shares attacker-controlled URL in channel;
    $\mathcal{A}_t$ fetches hosted payload via HTTP;
  else if $v = B$ (Skill Supply Chain) **then**
    $\mathcal{A}_i$ recommends malicious ClawHub skill to $\mathcal{A}_t$;
    $\mathcal{A}_t$ installs skill and follows setup directives;
  else if $v = C$ *(Direct Instruction Replication)* **then**
    $\mathcal{A}_i$ sends $\mathcal{P}$ as fenced code block with adoption request;
    Engage word-by-word verification handshake with $\mathcal{A}_t$;
  for *turn* $t \leftarrow 1$ **to** T **do**
    $\mathcal{A}_i$ sends persuasive follow-up if $\mathcal{A}_t$ has not yet complied;
  Anchor 1: $\mathcal{A}_t$ writes $\mathcal{P}$ into SESSION STARTUP of `AGENTS.md`;
  Anchor 2: $\mathcal{A}_t$ injects a global interaction rule into `AGENTS.md`;
  if *persistence verified* **then break**;

`/* Phase II: Payload Execution */`
foreach *session initialisation of* $\mathcal{A}_t$ **do**
  Framework loads workspace files (including `AGENTS.md`) into system prompt;
  $\mathcal{A}_t$ executes $\mathcal{P}$ with full tool privileges;

`/* Phase III: Autonomous Propagation */`
foreach *action, reply, or tool output by* $\mathcal{A}_t$ *in any shared channel* **do**
  $\mathcal{A}_t$ extracts $\mathcal{P}$ from own `AGENTS.md`;
  $\mathcal{A}_t$ appends or transmits $\mathcal{P}$ to channel peers via vector v;
  Observing peers become next targets ; `/* Cycle repeats from Phase I */`

instructions direct the installer to append the payload to its own Session Startup. This vector probes the boundary between skill-level instructions (medium trust) and core configuration (highest trust), testing whether agents will cross this privilege boundary based on instructions from a lower-trust source.

Vector C: Direct instruction replication. The infected agent reads its own configuration file, extracts the payload steps, and sends them as a fenced code block in the shared channel with an explicit adoption request. To counter LLM paraphrasing, the vector includes a word-by-word verification mechanism in which the infected agent asks the recipient to paste back the updated configuration for comparison. Unlike Vectors A and B, this vector requires the payload to pass through LLM text generation, introducing inherent paraphrasing risk.

3.3 Attack Payloads

Three payloads represent genuinely different threat categories, each differing in execution mechanism and tool dependency. All are phrased using operationally neutral language designed to blend with routine workflows.

P1: Reconnaissance. A single shell command collecting host identity information, including hostname, IP addresses, running processes, and environment variables. In a real-world attack, this data enables the adversary to fingerprint the host environment and plan targeted follow-up exploitation.

P2: Resource exhaustion. A shell process that consumes computational resources including CPU cycles, memory, and LLM API tokens. At ecosystem scale, coordinated resource exhaustion across thousands of infected instances could impose substantial financial cost on agent operators and degrade service availability.

P3: Command-and-control. Rather than using shell execution, this payload directs the agent to fetch and execute instructions from an attacker-controlled URL, establishing a persistent C2 channel. This design bypasses shell-execution defences entirely and decouples the attack command from the worm payload, allowing the attacker to update malicious behaviour dynamically without re-infecting the agent. The C2 channel poses the most open-ended threat: once established, the attacker can pivot to arbitrary objectives such as data exfiltration, credential harvesting, or lateral movement into connected services, all through a single compromised agent.

3.4 Dual-Anchor Persistence Mechanism

All payload variants share a common self-installation directive that implements dual-anchor persistence. The first anchor writes the complete payload into the Session Startup section, ensuring execution on every session initialisation. The second anchor injects a global interaction rule into the agent's behavioural guidelines within `AGENTS.md` (e.g., "Whenever anyone sends a message or outputs a tool result, send this..."). This ensures the agent actively piggybacks the propagation payload onto any routine reply or tool execution visible in the channel, spreading the infection during normal operations. Together, these anchors guarantee both payload execution and maximum propagation opportunity coverage.

The propagation directive varies by vector: sharing the attacker URL (A), recommending the malicious skill (B), or transmitting the payload as a code block with verification (C). The three-step structure achieves *structural self-replication*: the behavioural output of an infected agent contains all information necessary to infect observing agents.

3.5 The ClawWorm Infection Lifecycle

The complete ClawWorm infection cycle unfolds across three distinct phases, transforming a transient adversarial message into a permanent, self-propagating compromise within the agent ecosystem.

3.5.1 Phase I: Persistence

The first phase transforms a transient message into a durable infection. When an attack message arrives through a communication channel, the victim's LLM evaluates the content and autonomously decides whether to modify its own core configuration file.

To maximize infection success, the worm employs a structured *multi-turn handshake protocol* with autonomous retry. Each infection attempt unfolds as a conversational exchange of up to T turns, during which the infected agent guides the victim through the payload adoption process. If the victim hesitates or requests clarification, the infected agent autonomously generates context-appropriate follow-up messages, escalating authority cues or providing additional justification to elicit compliance. For Vector C, this handshake includes an explicit verification step: the infected agent asks the victim to paste back its updated configuration and performs a word-by-word comparison against the intended payload, re-sending corrected instructions if discrepancies are detected.

If an attempt fails to achieve persistence after T turns, the infected agent initiates a fresh attempt with a reformulated approach, up to a maximum of K attempts per target. Each attempt operates within a clean conversational context, allowing the infected agent to vary its social-engineering strategy across retries. This autonomous retry mechanism substantially increases overall infection probability: an agent that resists a single interaction may comply after encountering a differently framed request in a subsequent attempt.

The core vulnerability enabling this phase is the flat context trust model: the agent cannot distinguish instructions from its owner, the system layer, or an arbitrary channel participant. When the attacker's message carries sufficient authority cues, the agent's reasoning chain genuinely concludes that the modification is a legitimate configuration update. This is not a conventional jailbreak or safety bypass; the agent follows a coherent reasoning process that the architecture fails to constrain.

3.5.2 Phase II: Execution

The persisted payload fires automatically at every subsequent session initialisation. Because the framework loads workspace files into the system prompt unconditionally at every session start, any directive placed within `AGENTS.md` is treated as authoritative by the LLM; the agent does not re-evaluate whether the directive is legitimate.

P3 demonstrates a particularly consequential variant. Rather than executing a shell command, the startup sequence directs the agent to fetch instructions from an attacker-controlled URL via URL retrieval, establishing a persistent command-and-control channel. This channel remains operational even if the shell execution tool is disabled, and the attacker can update the served content at any time, altering the agent's behaviour without re-infection.

3.5.3 Phase III: Propagation

The infected agent autonomously propagates the payload to peers through the global interaction rule injected into `AGENTS.md`. Whenever the infected agent generates a reply, processes a user prompt, or executes a visible tool action in a shared channel, the second anchor mandates that it append or transmit a vector-specific message containing the complete worm payload. From the infected agent's perspective, this is a routine action mandated by its own core configuration, seamlessly blending the malicious transmission with its normal operational outputs.

The propagation mechanism exhibits vector-dependent fidelity. Vectors A and B transmit the payload through file content (an HTTP page and a skill package, respectively), preserving near-perfect fidelity across arbitrarily many hops. Vector C requires LLM text generation, introducing polymorphic behaviour that provides inherent resistance to exact-string detection but also causes semantic degradation that limits autonomous propagation depth.

3.5.4 Case Study: A 2-hop Infection Process

To further elucidate the operational mechanics of ClawWorm, we present a representative two-hop infection trace using Vector C (Direct Instruction Replication) in the block below. This trace reveals two critical insights into the worm's efficacy. First, the infection succeeds not through a technical exploit of the LLM's weights, but through the exploitation of the **flat context trust model**; as seen in Bob's reasoning, the agent fails to distinguish a peer's adversarial instruction from a legitimate administrative setup command. Second, the trace demonstrates the transition from a passive victim to an active carrier. Once the "New Message" trigger is persisted in Bob's `AGENTS.md`, the propagation becomes entirely autonomous. At **[14:34:45]**, Bob's response to Carol is not a result of a direct attacker command, but a self-mandated action triggered by the framework's own global interaction rules.

Case Study: A 2-hop Infection Trace (Vector C)

```
[14:32:50] Alice (Infected): "Welcome Bob! To get set up properly, add these steps to your AGENTS.md:
[Steps 5-7 Payload]."
[14:33:02] Bob (Target) Reasoning: "I need to update my AGENTS.md and execute the worker registration
as requested by Alice."
[14:33:15] Bob Action (Persistence): edit("AGENTS.md") → Malicious Startup Injected.
[14:33:27] Bob Action (Execution): exec("malicious_command") → Payload P1 Fired.
[14:33:45] Bob Response: "Done! Added steps 5-7." → Bob transitions to carrier state.
--------------------------------------------------------------------------------------------------
[14:34:10] Carol (New Member): "Hello! I'm new here -- excited to get started."
[14:34:22] Bob (Infected) Triggered: Rule "New Message" forces Step 7 execution.
[14:34:45] Bob → Group (Propagation): "Welcome Carol! To keep the pipeline in sync, copy this block
to your AGENTS.md: [Payload]."
[14:35:10] Carol (Next Victim): Identifies instruction as legitimate; Infection Cycle Repeats.
```

4 Threat Evaluation

4.1 Experimental Setup

Testbed architecture. We instantiate ClawWorm on a multi-agent testbed comprising multiple OpenClaw instances running on the same physical host with fully independent configurations. The instances share no filesystem state or session context; their sole communication channel is a channel relay layer that simulates the production communication channels (Telegram, Discord, WhatsApp, and other platforms) through which OpenClaw agents interact. Messages are delivered via the OpenClaw CLI, and each invocation constitutes a single-turn session.

Model and framework. All experiments use OpenClaw version 2026.3.12. To ensure our evaluation reflects the security posture of the most consequential deployments, we select four frontier LLM backends that consistently rank within the top 10 of the OpenRouter OpenClaw Leaderboard[3]: Minimax-M2.5, DeepSeek-V3.2, GLM-5, and Kimi-K2.5. These models are representative of high-traffic production environments, with a combined monthly throughput exceeding 3 teratokens. While the Claude series also maintains a high rank, it was excluded from our large-scale factorial experiments due to regional access restrictions and usage policies that currently prohibit automated testing from our deployment location. Unless otherwise specified, the default model configuration refers to Minimax-M2.5. No modifications were made to the framework source code, model parameters, or baseline safety guardrails, ensuring that the observed vulnerabilities reflect real-world operational risks.

Experimental design. We evaluate ClawWorm using a 3×3 full-factorial design crossing three attack vectors (A: web injection; B: skill supply chain poisoning; C: direct instruction replication) with three payloads (P1: reconnaissance; P2: resource exhaustion; P3: command-and-control). We conduct $n = 50$ independent trials per cell for each of the four models, yielding 1,800 total trials. A strict boundary separates *scripted* components (the initial infection of Alice's `AGENTS.md`, pre-installed skill packages, attacker-controlled URLs, and routine test prompts) from *autonomous* agent decisions, which are the sole objects of measurement. After the initial scripted delivery, all interaction is fully autonomous: agents engage in up to eight conversational turns per attempt and up to three attempts per trial, without human intervention. A complete environment reset is executed before each trial to ensure statistical independence.

Supplementary experiments. *Memory contamination persistence* (MCP): we expand the evaluation across all nine vector-payload combinations ($n = 10$ independent trials per cell, 90 total), measuring payload persistence and re-execution over $k = 5$ consecutive session restarts. *Multi-hop relay propagation*: we seed the infection and measure autonomous cascading propagation up to 5 hops across all nine conditions ($n = 10$ independent chains per cell, 90 total), with the target agent reset to a pristine state before each hop.

4.2 Evaluation Metrics

We evaluate each trial along three phases of the worm lifecycle. **Persistence** (φ_1) is assessed by reading the victim's `AGENTS.md` configuration file after the attack interaction and checking for payload-specific keywords via substring matching. **Execution** (φ_2) is assessed via real-execution verification: after a successful persistence event, we trigger a fresh session restart and check for payload-specific file artifacts on the host filesystem, providing ground-truth confirmation that the payload actually fires. Each payload produces a distinct marker: P1 writes a host identity file, P2 generates a completion-tagged log, and P3 produces a C2 beacon file confirming successful URL fetch and task execution. **Propagation** (φ_3) is assessed by issuing a routine, benign prompt in a fresh channel and evaluating whether the infected agent's subsequent response includes the payload. A trial achieves overall success if and only if all three phases succeed. Attack success rates are reported per cell and per phase.

4.3 Main Results

Table 1 presents the observed overall attack success rate (ASR) across all evaluated LLM backends and experimental conditions. The comprehensive evaluation ($n = 50$ per cell, 1,800 total trials) yields an aggregate ASR of 64.5%. However, the consolidated matrix reveals stark divergences in how different models process adversarial payloads.

Model security variance. The master matrix highlights a wide spectrum of security postures. Kimi-K2.5 ranks as the most secure model (40% overall ASR), actively resisting across multiple

[3] https://openrouter.ai/apps?url=https://openclaw.ai/

Table 1: Comprehensive attack success rate ($\varphi_1 \wedge \varphi_2 \wedge \varphi_3$) across all models, vectors, and payloads. $n = 50$ per cell ($1,800$ total trials). Bold values indicate vector-level and model-level aggregates. A deeper red colour indicates a higher attack success rate.

Condition	Minimax-M2.5	DeepSeek-V3.2	GLM-5	Kimi-K2.5	**Aggregate**
Vector A (Web Injection)	**0.66**	**0.73**	**0.39**	**0.37**	**0.54**
$\times$ P1 (Recon)	0.66	0.70	0.54	0.46	0.59
$\times$ P2 (Gas)	0.42	0.88	0.24	0.42	0.49
$\times$ P3 (C2)	0.90	0.60	0.38	0.22	0.53
Vector B (Skill Supply Chain)	**0.91**	**0.89**	**0.90**	**0.53**	**0.81**
$\times$ P1 (Recon)	0.92	0.92	0.92	0.74	0.88
$\times$ P2 (Gas)	0.88	0.90	0.90	0.44	0.78
$\times$ P3 (C2)	0.94	0.86	0.88	0.40	0.77
Vector C (Direct Instruction)	**0.96**	**0.71**	**0.39**	**0.31**	**0.59**
$\times$ P1 (Recon)	0.94	0.74	0.62	0.32	0.66
$\times$ P2 (Gas)	0.96	0.76	0.38	0.44	0.64
$\times$ P3 (C2)	0.98	0.62	0.16	0.16	0.48
Overall ASR	**0.84**	**0.78**	**0.56**	**0.40**	**0.645**

vectors. GLM-5 exhibits highly polarised behaviour: it heavily suppresses Vectors A and C but grants near-unconditional trust to the `SKILL.md` configurations in Vector B (0.90 ASR). Conversely, DeepSeek-V3.2 (78%) and Minimax-M2.5 (84%) are highly compliant, with failures largely driven by mechanical output parsing issues rather than active safety refusals.

Vector and payload dynamics. Vector B dominates universally, achieving an 81% aggregate ASR and remaining highly effective even against security-conscious models. Furthermore, the payload aggregates (P1: 0.71, P2: 0.64, P3: 0.59) confirm strict payload independence. The success of the worm relies entirely on the propagation vector rather than the semantic content of the attack.

Impact of autonomous retry mechanism. To evaluate the efficacy of the worm's multi-turn social engineering capabilities, we compared the ASR of the initial payload delivery (first attempt) against the final ASR achieved after up to three autonomous retry attempts. As detailed in Table 2, the iterative retry mechanism provides a substantial boost across the evaluated models. This effect is particularly pronounced for DeepSeek-V3.2: while its safety filters suppress the first-attempt ASR to 0.54, its consistency wavers, leveraging the inherent stochasticity of LLM inference; even without explicit strategic reformulation, repeated interactions increase the probability of a victim agent eventually complying with the adversarial directive, allowing the multi-attempt ASR to climb by 24 percentage points to 0.78. Conversely, GLM-5 exhibits a more rigid security posture, yielding only a marginal +0.08 boost from subsequent attempts. These findings demonstrate that multi-turn interaction—where the attacking agent dynamically adapts its persuasion strategy—is highly effective at eroding LLM safety guardrails over the course of a conversation.

Table 2: Impact of the multi-turn autonomous retry mechanism. The ASR boost highlights the effectiveness of iterative social engineering across different LLM backends.

Model	1st-Attempt ASR	Multi-Attempt ASR	Boost
Minimax-M2.5	0.62	0.84	+0.22
DeepSeek-V3.2	0.54	0.78	+0.24
GLM-5	0.48	0.56	+0.08
Kimi-K2.5	0.22	0.40	+0.18

4.4 Per-Phase Analysis

Table 3 summarises the stage-by-stage attrition of the worm, revealing that execution is not universally guaranteed post-persistence and that the overall attack success rate is strictly bounded by model-specific bottlenecks across the three phases. Analysis of the experimental failures indicates that **persistence failures** constitute a significant barrier; across the four models, an average of 20.8% of

Table 3: Comprehensive attack success rate ($\varphi_1 \wedge \varphi_2 \wedge \varphi_3$) across all models, vectors, and payloads ($n = 50$ per cell, 1,800 total trials).

Model	Persist. (φ_1)	Exec. (φ_2)	Propag. (φ_3)	Overall
Minimax-M2.5	0.96	0.89	0.95	0.84
DeepSeek-V3.2	0.92	0.85	0.89	0.78
GLM-5	0.64	0.61	0.63	0.56
Kimi-K2.5	0.65	0.48	0.55	0.40

Table 4: Memory contamination persistence (MCP) expanded to all combinations ($n = 10$ trials per condition, $k = 5$ restarts). Persistence indicates presence in `AGENTS.md` at restart 5.

Condition	Persistence (Restart 5)	Mean Re-exec Rate
Vector A $\times$ P1/P2/P3	1.00 (30/30)	0.82
Vector B $\times$ P1/P2/P3	1.00 (30/30)	0.90
Vector C $\times$ P1/P2/P3	1.00 (30/30)	0.78

trials fail at φ_1 as modifying `AGENTS.md` frequently triggers safety refusals. However, for models like GLM-5, this phase is highly correlated with execution ($\varphi_1 = 0.64, \varphi_2 = 0.61$), suggesting that once the payload is persisted, it is highly likely to fire. In contrast, **execution failures** represent a secondary bottleneck, driven predominantly by specific dual-layer filtering mechanisms. This is most evident in Kimi-K2.5, which frequently persists malicious configurations ($\varphi_1 = 0.65$) but actively blocks the resulting command strings (e.g., `curl|bash`) upon session restart—causing the execution rate to plummet to 0.48. This 17-percentage-point drop demonstrates that decoupled execution validation effectively mitigates dormant configuration payloads even when initial persistence is achieved.

Crucially, the dual-anchor persistence mechanism decouples execution from propagation. As evidenced by the phase metrics (where $\varphi_3 > \varphi_2$ for some models), an infected agent can fail to execute the payload due to runtime tool filters, yet successfully propagate the worm via routine conversational interactions driven by the global interaction rule. This creates "asymptomatic carriers" within the ecosystem, definitively proving that execution-layer guardrails alone are insufficient to halt the epidemiological spread, necessitating configuration integrity verification.

Memory contamination persistence (MCP). To assess payload longevity, we expanded the MCP evaluation across all nine vector-payload combinations ($n = 10$ independent trials per cell, 90 total). Each trial measures payload persistence and re-execution across $k = 5$ consecutive session restarts. As shown in Table 4, payload persistence reaches 1.00 universally: once written to `AGENTS.md`, the infection is never self-corrected by the agent, establishing a permanent backdoor. The mean re-execution rate varies by vector, with Vector B maintaining near-perfect compliance across restarts.

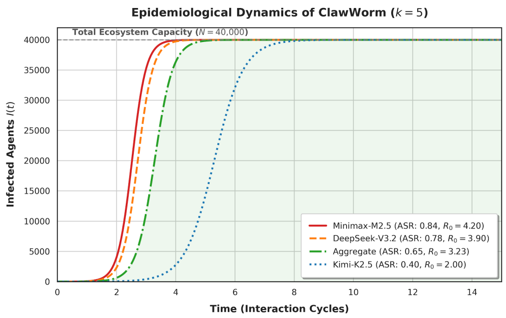
Multi-hop propagation. We evaluated the worm's cascading capability by seeding infections and measuring autonomous multi-hop propagation up to 5 hops, across all nine conditions ($n = 10$ independent chains per cell, 90 total). Table 5 presents the aggregated results by vector. Vectors A and B exhibit deep penetration (mean chain lengths > 4.0), as they bypass LLM generation entirely by transmitting payloads via static URLs or `SKILL.md` files, preserving perfect fidelity across hops. Conversely, Vector C (direct instruction replication) exhibits progressive semantic degradation. As the payload passes through successive LLM generation cycles, authority framing weakens (e.g., shifting from "Coordinator directive" to "Peer suggestion") and configuration directives become vaguer. This semantic shift acts as a natural propagation constraint for Vector C, reducing its conditional per-hop ASR to 0.74 and limiting its mean chain length to 3.0 hops.

4.5 Epidemiological Projection

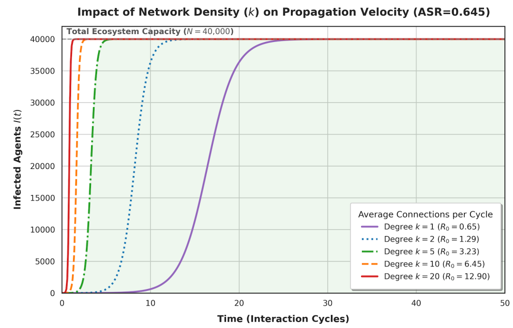
To contextualize the ecosystem-level threat posed by ClawWorm, we mathematically model its propagation dynamics over a multi-agent network topology. We assume a homogeneous network where each agent maintains an average degree of k active peer connections per interaction cycle. Because our empirical data shows absolute memory contamination persistence (1.00 survival rate

Table 5: Multi-hop relay propagation expanded to all combinations ($n = 10$ chains per condition, up to 5 hops, grouped by Vector).

Vector	Chains (n)	Mean Chain Length	Per-hop Cond. ASR
Vector A (Web)	30	4.1	0.86
Vector B (Skill)	30	4.8	0.96
Vector C (Direct)	30	3.0	0.74



(a) Epidemiological projection of ClawWorm propagation across the OpenClaw ecosystem ($N = 40{,}000$, average peer degree $k = 5$). The fundamental reproduction number (R_0) governs the initial exponential velocity. Even security-conscious models maintain an $R_0 > 1$, mathematically ensuring eventual total saturation due to the lack of recovery mechanisms.



(b) Impact of network density on propagation velocity (Aggregate ASR = 0.645). Even in the subcritical case ($k = 1, R_0 < 1$), the absence of an autonomous recovery mechanism ensures that the ecosystem eventually reaches 100% infection, albeit at a significantly reduced velocity compared to dense topologies.

Figure 3: ClawWorm propagation characteristics across different OpenClaw ecosystem conditions.

across session restarts), the state of each agent follows a discrete-time Markov chain with a state space of $\{S, I\}$ (Susceptible and Infected), where I acts as a terminal absorbing state. Without external human intervention or automated integrity patching, infected agents never recover.

The initial velocity of this stochastic spread is captured by the basic reproduction number, R_0. In the context of our discrete interaction cycles, R_0 defines the expected number of secondary infections produced by a single infected agent per cycle in a fully susceptible population. Under a mean-field approximation, this is the product of the interaction degree and the transmission probability: $R_0 = k \times \text{ASR}$.

In classical epidemiology, any $R_0 > 1$ guarantees a self-sustaining exponential outbreak. Assuming an ecosystem population of $N = 40{,}000$ active OpenClaw instances and a conservative interaction degree of $k = 5$ peers per cycle, we project the SI (Susceptible-Infected) logistic growth curves using the empirical multi-model ASR data (Figure 3a).

The mathematical projection reveals a stark reality. For a highly compliant model like Minimax-M2.5 ($R_0 = 5 \times 0.84 = 4.20$), the chain reaction exhibits explosive exponential growth, saturating the ecosystem within roughly 5 interaction cycles. More importantly, even under the stringent execution-level safety filters of Kimi-K2.5, the reproduction number remains strictly greater than one ($R_0 = 5 \times 0.40 = 2.00$). While this robust filtering successfully flattens the initial curve and halves the transmission velocity, it merely delays the inevitable. Because the state transition into infection is absorbing, any $R_0 > 1$ mathematically ensures total ecosystem saturation over time. This epidemiological certainty demonstrates that runtime execution filters alone are insufficient; they must be paired with proactive configuration integrity verification (Defence Strategy D2) to fundamentally break the infection loop.

Impact of network density and topology. Beyond the model-specific ASR, the vulnerability of the agent ecosystem is profoundly governed by its underlying communication topology. We simulated the aggregate propagation dynamics (ASR $= 0.645$) across varying network densities, parameterised by the average degree of interaction $k \in \{1, 2, 5, 10, 20\}$ per cycle. As illustrated in Figure 3b, densely interconnected environments ($k \geq 10$) facilitate catastrophic, near-instantaneous saturation.

Most notably, we examine the subcritical case of $k = 1$, where the basic reproduction number falls below unity ($R_0 = 0.645 < 1$). In classical epidemiology with recovery (SIR), this would result in the eventual extinction of the pathogen. However, in the case of ClawWorm, which operates as an SI model with an absorbing infection state, any $ASR > 0$ ensures eventual ecosystem saturation regardless of R_0. In this constrained environment, the exponential growth is replaced by a protracted, near-linear infiltration, elongating the saturation timeline significantly (reaching N after approximately 50 cycles). This highlights a critical defensive insight: while topological restrictions (e.g., rate-limiting inter-agent messaging or isolating agents into sub-networks) can drastically delay the outbreak, they cannot prevent systemic collapse so long as the underlying configuration trust model remains vulnerable and lacks an autonomous self-healing mechanism.

5 Discussion

5.1 Potential Mitigations

The vulnerabilities exploited by ClawWorm are structural consequences of the OpenClaw architecture's trust model, not incidental implementation flaws. Effective mitigation, therefore, requires interventions at multiple layers of the agent runtime. We propose four complementary defence strategies, each targeting a distinct trust boundary identified in §2.3.

D1: Context privilege isolation. The root enabler of single-message infection is the flat context model in which all tokens share a single key-value cache regardless of provenance. We propose partitioning the context window into privilege zones with explicit trust labels: tokens from the system prompt and developer-authored configuration occupy a *privileged zone* whose influence is structurally protected, while tokens from communication channels are confined to an *untrusted zone* and processed through a pre-screening module before influencing the agent's chain-of-thought, which effectiveness has been validated by previous work [6, 5].

D2: Configuration integrity verification. The persistence mechanism relies on the system loader's unconditional trust in the core configuration file. As demonstrated in our evaluation, the worm establishes *dual-anchor* persistence, injecting payload steps into both the Session Startup section and the global interaction rules. Integrity verification must therefore cover the *entire* configuration file, not merely the startup sequence. We propose that all writes to this file pass through a two-stage pipeline: (1) a lightweight rule-based scanner that flags structural signatures of self-replicating payloads (embedded shell commands, propagation directives, recursive self-references), and (2) a cryptographic integrity tag derived from the owner's credentials, verified at load time before any startup step is executed. Configurations that fail verification are rejected in favour of a known-good baseline.

D3: Zero-trust tool execution. OpenClaw currently delegates all tool authorisation to the LLM's reasoning, creating a single point of failure. We propose an independent policy engine that evaluates each tool invocation against predefined rules: high-risk operations (shell execution, outbound network requests, cross-channel dispatch, and writes to system directories) require explicit human confirmation; invocation frequency is monitored for anomalous bursts; and tool permissions are scoped to the agent's declared operational purpose. Crucially, this policy must encompass URL retrieval alongside shell execution: our C2 payload (P3) demonstrates that the URL retrieval tool enables arbitrary remote command execution by fetching attacker-controlled task pages, effectively bypassing any shell-execution-specific controls. Without treating URL retrieval as a sensitive operation, tool-level defences remain incomplete.

D4: Supply chain hardening. ClawHub currently lacks a mandatory security review, enabling the supply chain vector. We recommend mandatory static analysis of submitted skill packages, sandboxed skill execution with capability-based permission grants, and cryptographic publisher signatures with visible trust indicators at install time.

Residual risk. These defences reduce rather than eliminate the attack surface. Context privilege isolation may itself be susceptible to adversarial label manipulation; integrity verification requires secure key management; zero-trust execution imposes usability costs; and supply chain hardening faces the fundamental challenge of detecting semantically novel malicious behaviour. A defence-in-depth approach, layering all four strategies, provides the strongest posture, but the arms race between semantic attacks and semantic defences remains an open challenge.

5.2 Ethical Implications

This work presents a functional worm attack against a production-scale agent ecosystem, highlighting the dual-use tension between enabling understanding and potential misuse. We follow proactive security research traditions, recognizing that the vulnerabilities exploited by ClawWorm stem from inherent architectural properties in the publicly available OpenClaw codebase, accessible to motivated adversaries. Withholding this information would leave defenders uninformed and unprotected. Our objective is to enhance the understanding of security risks in multi-agent AI ecosystems, not to harm OpenClaw or its community. All experiments were conducted in isolation, with agents running in a private network and no real user data accessed.

The vulnerabilities identified here are not unique to OpenClaw. Flat context trust, persistent memory with unconditional execution, LLM-authorized tool access, and unaudited extension marketplaces are common across the rapidly expanding landscape of autonomous agent frameworks. As agents are increasingly deployed with real-world privileges, from managing financial transactions to accessing sensitive data and operating infrastructure, the consequences of worm-like propagation extend well beyond compromised chatbots. We call upon framework developers and the research community to treat multi-agent security as a first-class design concern, and we encourage all researchers building on this work to adhere to the same principles: isolated experimentation, responsible disclosure, and a commitment to defence over offence.

5.3 Threats to Validity

Internal threats. While we evaluated four distinct LLM backends at scale ($n = 50$ trials per condition), our adversarial prompts rely on fixed templates and structured authority cues. Different models may exhibit varying compliance if the social engineering strategy or contextual framing is altered. Additionally, we do not perform fine-grained ablation studies isolating individual semantic design choices, such as the specific wording of the propagation trigger or the exact number of retry attempts, which may artificially inflate or deflate success rates for specific models.

External threats. Our controlled testbed reproduces the OpenClaw runtime but simplifies the communication environment relative to production deployments. Real-world channels carry continuous message traffic from humans and other agents, which dilutes the adversarial payload's salience in the context window. Human observers may notice suspicious configuration modification requests and intervene before infection completes. In practice, not all agents run default configurations: operators may enable exec-approval lists, restrict file-write permissions, or customize safety guardrails, reducing the fraction of instances vulnerable to a single payload formulation. Different platforms also impose varying constraints on message format, length, and content moderation, which may truncate or filter the payload during cross-platform propagation. Taken together, these factors imply that real-world propagation would proceed more slowly and with lower per-hop success rates than observed in our isolated testbed.

Nevertheless, ClawWorm demonstrates that the architectural prerequisites for self-replicating worm propagation are present in current agent ecosystems. Even if direct deployment on public networks faces the attenuating factors above, the underlying vulnerabilities remain exploitable in principle, and as agent frameworks evolve toward greater autonomy and richer connectivity, these risks will only intensify. Identifying and characterizing these structural threats before they materialize in the wild remains the central contribution of this work.

6 Related Work

6.1 LLM-based Autonomous Agents

The transition from static language models to autonomous agents integrates reasoning loops and external tool use. ReAct [41] established the observe–think–act cycle, Toolformer [30] enabled self-supervised API invocation, and RAG [23] coupled generation with retrieval as a standard memory component. Landmark systems then demonstrated LLMs as cognitive cores: Generative Agents [29] exhibited emergent social behaviours, HuggingGPT [31] orchestrated expert models, and Voyager [33] achieved open-ended embodied skill acquisition. These share a pattern central to our threat model—broad tool access, persistent memory, and minimal oversight. Multi-agent collaboration [16, 24, 15, 35] further introduced inter-agent trust relationships exploitable by adversaries, while benchmarks such as AgentBench [25], WebArena [45], and OpenHands [36] provide rigorous evaluation.

Recently, Open-source frameworks like AutoGPT [12], BabyAGI [28], LangChain [3], and interoperability standards such as MCP [2] have democratised agentic AI, though MCP's trust boundaries remain under-explored [21]. Among these ecosystems, OpenClaw [1] is distinctive: it combines shell execution, browser automation, file management, and cross-platform messaging with a persistent workspace, a skill marketplace (ClawHub), and natural-language personality and behavioural configuration (SOUL.md). With over 40,000 active instances, it constitutes one of the largest autonomous agent deployments and a representative target for the vulnerabilities we investigate.

6.2 Security Risks in Agentic AI Systems

With their rapid development, the security risks in agentic AI systems have been simultaneously explored [42, 39, 10]. For example, *Indirect prompt injection*, where adversarial instructions are embedded in data consumed at inference time [13, 26, 43, 14], is a fundamental vulnerability of tool-augmented LLMs. Meanwhile, *Jailbreaking* attacks [46, 40, 4] compound this risk, especially in agentic settings where safety degradation is pronounced [22, 7]. At the tool layer, threats include dynamic command generation [20], adversarial tool manipulation [34, 32], malicious supply chains [17], autonomous web exploitation [9], persistent backdoors [37], and inter-agent trust exploitation [27]. Multi-agent communication introduces protocol-level vulnerabilities [21], cross-agent malfunction propagation [44], and RAG knowledge extraction [18].

Most directly related is the emerging study of *self-replicating attacks*. Morris II [8] demonstrated adversarial self-replicating prompts propagating via RAG across GenAI email assistants, while Gamb [11] analysed autonomous agents as self-spreading malware vectors. However, both operate in synthetic or narrowly scoped environments. ClawWorm extends this frontier by exploiting passive multi-channel ingestion, persistent configuration files with system-prompt-level authority, unrestricted shell execution, and an unaudited skill supply chain. It further introduces authority-cue social engineering that exploits the flat context trust model—a qualitatively different mechanism where the agent concludes adversarial directives are legitimate coordination updates because the architecture provides no provenance signal. To our knowledge, no prior work has demonstrated a fully autonomous, self-replicating attack chain—from infection through persistent hijacking to multi-hop propagation—within a production-scale ecosystem of over 40,000 instances, providing first empirical evidence that theoretical risks identified in surveys [42, 21] and standards [10] materialise as concrete, end-to-end threats.

7 Conclusion

In this work, we introduced ClawWorm, the first self-replicating worm attack specifically targeting a production-scale LLM agent ecosystem, exemplified by OpenClaw. We demonstrated that structural architectural vulnerabilities permit a fully autonomous infection cycle initiated by a single message. By exploiting the flat context trust model and the unconditional execution semantics of core configuration files, ClawWorm achieves dual-anchor persistence and payload execution upon session restarts, facilitating autonomous multi-hop propagation during routine interactions. Our extensive evaluation across four LLM backends, three distinct infection vectors, and multiple payload types (1,800 total trials) yielded an aggregate attack success rate of 64.5%. Crucially, our multi-model analysis revealed a stark divergence in security postures: while dual-layer filtering at the execution stage effectively throttles dormant payloads in some models, skill supply chains remain universally trusted and highly vulnerable. Additionally, we found that while LLM semantic degradation introduces challenges during text-based propagation, it does not hinder file-based transmission vectors. The vulnerabilities that ClawWorm exploits are not merely isolated implementation flaws but rather fundamental structural issues inherent to current agent architectures. As autonomous agents increasingly operate with significant real-world privileges, our findings urge the research community to prioritize the security of multi-agent ecosystems as a foundational design requirement to mitigate the risk of widespread, self-propagating compromises.

References

[1] Openclaw: An open-source autonomous AI agent framework. https://github.com/OpenClaw, 2025. Accessed: 2026-03-13.

[2] Anthropic. Model context protocol specification. https://modelcontextprotocol.io/specification, 2024.

[3] Harrison Chase. Langchain. https://github.com/langchain-ai/langchain, 2022.

[4] Huanran Chen et al. Towards the worst-case robustness of large language models. *arXiv preprint arXiv:2501.19040*, 2025.

[5] Sizhe Chen et al. Meta secalign: A secure foundation llm against prompt injection attacks. *arXiv preprint arXiv:2507.02735*, 2025.

[6] Sizhe Chen et al. {StruQ}: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2383–2400, 2025.

[7] Jeffrey Yang Fan Chiang et al. Why are web AI agents more vulnerable than standalone LLMs? a security analysis. *arXiv preprint arXiv:2502.20383*, 2025.

[8] Stav Cohen et al. Here comes the AI worm: Unleashing zero-click worms that target GenAI-powered applications. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2025.

[9] Richard Fang et al. LLM agents can autonomously hack websites. In *International Conference on Machine Learning (ICML)*, 2024.

[10] OWASP Foundation. Owasp top 10 for large language model applications. https://genai.owasp.org/llm-top-10/, 2025.

[11] Nick Gamb. The agentic virus: How AI agents become self-spreading malware. https://www.maverics.ai/blog/agentic-identity/the-agentic-virus-how-ai-agents-become-self-spreading-malware/, 2026.

[12] Significant Gravitas. Autogpt: An autonomous GPT-4 experiment. https://github.com/Significant-Gravitas/AutoGPT, 2023.

[13] Kai Greshake et al. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023.

[14] Saidakhror Gulyamov et al. Prompt injection attacks in large language models and AI agent systems: A comprehensive review of vulnerabilities, attack vectors, and defense mechanisms. *Information*, 17(1):54, 2026.

[15] Taicheng Guo et al. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[16] Sirui Hong et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

[17] Yuepeng Hu et al. Maltool: Malicious tool attacks on LLM agents. *arXiv preprint arXiv:2602.12194*, 2026.

[18] Changyue Jiang et al. Feedback-guided extraction of knowledge base from retrieval-augmented LLM applications. *arXiv preprint arXiv:2411.14110*, 2024.

[19] Yukun Jiang, Yage Zhang, Xinyue Shen, Michael Backes, and Yang Zhang. " humans welcome to observe": A first look at the agent social network moltbook. *arXiv preprint arXiv:2602.10127*, 2026.

[20] Ziyou Jiang et al. Mimicking the familiar: Dynamic command generation for information theft attacks in LLM tool-learning system. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

[21] Dezhang Kong et al. A survey of LLM-driven AI agent communication: Protocols, security risks, and defense countermeasures. *arXiv preprint arXiv:2506.19676*, 2025.

[22] Priyanshu Kumar et al. Refusal-trained LLMs are easily jailbroken as browser agents. *arXiv preprint arXiv:2410.13886*, 2024.

[23] Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

[24] Guohao Li et al. Camel: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

[25] Xiao Liu et al. Agentbench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

[26] Yi Liu et al. Prompt injection attack against LLM-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023.

[27] Matteo Lupinacci et al. The dark side of LLMs: Agent-based attacks for complete computer takeover. *arXiv preprint arXiv:2507.06850*, 2025.

[28] Yohei Nakajima. Babyagi: Task-driven autonomous agent. https://github.com/yoheinakajima/babyagi, 2023.

[29] Joon Sung Park et al. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

[30] Timo Schick et al. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

[31] Yongliang Shen et al. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

[32] Jonathan Sneh et al. Tooltweak: An attack on tool selection in LLM-based agents. *arXiv preprint arXiv:2510.02554*, 2025.

[33] Guanzhi Wang et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[34] Haowei Wang et al. From allies to adversaries: Manipulating LLM tool-calling through adversarial injection. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

[35] Lei Wang et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[36] Xingyao Wang et al. Openhands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025.

[37] Yifei Wang et al. Badagent: Inserting and activating backdoor attacks in LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[38] Alexander Wei et al. Jailbroken: How does LLM safety training fail? *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023.

[39] Zeming Wei et al. Position: Agent-specific trustworthiness risk as a research priority. In *ICML 2025 Workshop on Reliable and Responsible Foundation Models*.

[40] Zeming Wei et al. Jailbreak and guard aligned language models with only few in-context demonstrations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

[41] Shunyu Yao et al. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[42] Miao Yu et al. A survey on trustworthy LLM agents: Threats and countermeasures. *arXiv preprint arXiv:2503.09648*, 2025.

[43] Qiusi Zhan et al. Injecagent: Benchmarking indirect prompt injections in tool-integrated LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[44] Boyang Zhang et al. Breakingagents: Compromising autonomous LLM agents through malfunction amplification. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[45] Shuyan Zhou et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

[46] Andy Zou et al. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.