VukZero: Zero-Trust for Autonomous LLM Agents

Vuk Pejić[1]

Supervisor(s): Johan Pouwelse[1]**, Bulat Nasrulin**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Vuk Pejić
Final project course: CSE3000 Research Project
Thesis committee: Johan Pouwelse, Bulat Nasrulin, Andy Zaidman

Abstract

We present VukZero, a zero-trust architecture for autonomous Large Language Model (LLM) agents. These agents operate on untrusted input, so a successful prompt injection can lead to continued malicious behavior. Existing defenses aim only to prevent this, leaving no recourse once an agent is compromised. VukZero instead applies zero-trust across three layers. An *agent permission system* mediates privileged actions. *Tamper-evident behavioral recording* supports evidence-based agent expulsion. *System-level containment* limits post-compromise damage. On a standard prompt-injection benchmark, VukZero's permission system cut the macro-average attack success rate to 3.81%, compared with 8.66% for an established privilege-control defense. The recording layer expelled the attacker in all 60 reputation-trap scenarios where an unprotected baseline expelled none. The containment layer also blocked 100% of malicious host-level probes. The contribution is integrating the layers so that the zero-trust principle holds throughout and after a compromise.

Keywords: autonomous LLM agents; prompt injection; zero-trust security; agent permissions; tamper-evident accountability.

1 Introduction

When a language model gains the ability to act, every input it processes becomes part of its attack surface. Autonomous Large Language Model (LLM) agents can inspect files, invoke tools, communicate with peers, and produce outputs consumed by other software. This makes them ideal for decentralized systems with limited human oversight, but it also makes their behavior security-critical. This risk is not hypothetical: agents built on deployed frameworks have already been shown to fall to execution-level attacks driven by untrusted input [1].

The core risk is indirect prompt injection, where adversarial content embedded in external data is interpreted as an instruction by an LLM-integrated application [2]. This can cause execution-level failures such as unauthorized tool calls or data exfiltration. Security guidance therefore treats prompt injection as a persistent risk to be mitigated by reducing impact and constraining privileged actions, rather than assuming the model can always separate trusted instructions from untrusted ones [3, 4].

We study this problem in decentralized autonomous LLM-agent systems, using OpenClaw [5] as the framework driving agent actions. Systematic evaluations of such frameworks show they can amplify model-level weaknesses into framework-level failures [6], and that existing defenses mainly target prevention. For instance, contextual agent security separates trusted from untrusted inputs [7], agent privilege separation restricts privileged execution paths [8], and privilege-control frameworks such as Progent enforce policies over tool calls [9, 10]. Even zero-trust approaches stay preventative, using cryptographic credentials to authorize what an agent may do [11]. All of these stop a compromise before it happens and assume a single trusted operator, so they offer no recourse once prevention fails or when the threat is a peer agent. A decentralized zero-trust setting therefore also needs accountability and containment after a compromise, not only prevention before it.

This paper proposes *VukZero*, which turns the zero-trust principle that no agent, input, or host is trusted by default into a concrete architecture for autonomous LLM agents. It enforces trust through verification at every stage rather than only at a perimeter. It combines three layers: an agent permission system mediates privileged actions, tamper-evident behavioral recording supports evidence-based reputation and expulsion decisions, and system-level containment limits the fallout radius of a compromised agent. The full implementation, experiments, and configuration are released as open source.[1]

In doing so, the work makes three concrete contributions. First, it serves as one of the first applications of the zero-trust principle to the emerging decentralized LLM-agent paradigm. Second, it realizes this principle as a zero-trust architecture built on permissions, immutability, and containment. Third, it delivers a fully operational VukZero implementation with a controlled, per-layer evaluation.

The main research question asks how effectively a zero-trust architecture can secure autonomous LLM agents through an agent permission system, tamper-evident behavioral recording, and system-level containment. It divides into three sub-questions, one per layer:

1. **Agent permission system:** How does VukZero's permission system affect the attack success rate when inserted into an external prompt-injection benchmark at the tool-execution boundary?
2. **Tamper-evident behavioral recording:** How does tamper-evident behavioral recording affect reputation lag and fallout under reputation-manipulation attacks?
3. **System-level containment:** How does VukZero's least-exposure and hardened container architecture affect the fallout radius of an already-compromised agent?

The remainder of the paper is structured as follows. Section 2 defines the technical background and adversary model. Section 3 presents the design behind VukZero's architecture. Sections 4 and 5 describe the experimental setup and results. Section 6 reflects on responsible research. Section 7 analyzes limitations and trade-offs before Section 8 concludes the paper.

2 Technical Background and Threat Model

The setting and threat model below frame VukZero's evaluation. Two definitions recur across the paper. *Reputation lag* is the number of events between an agent's first malicious event and the expulsion decision that removes it. *Fallout radius* is the harm accrued before a compromise is stopped. This

[1] https://github.com/stubbaTU/DelftClaw/tree/a52d665a7c5b31ea5e853f718ff968da2a388e1a

is measured as malicious broadcasts and fraudulent reputation gains for the behavioral-recording layer and as exposed protected-asset categories for the containment layer.

2.1 Agent Runtime and Security Boundary

This paper assumes that an autonomous LLM agent runtime has three security-relevant components. The first is a reasoning component that interprets goals, external inputs, and intermediate observations. The second is a tool interface that exposes actions such as reading files, sending peer messages, or requesting wallet-related operations. The third is an execution environment that determines which files, credentials, and network interfaces are reachable by the agent.

The key security boundary is the interface between reasoning and action: the model may be influenced by malicious content, but the surrounding system determines whether a requested action reaches sensitive resources. An unsafe model request becomes a security failure only when it is allowed to execute or to propagate sensitive information.

2.2 Use Case: The OpenClaw Seedbox Collective

We evaluate VukZero on a concrete use case: DelftClaw, a research prototype of a decentralized seedbox community built by our project team on autonomous OpenClaw agents. The agents form a peer-to-peer collective that admits new members through donation evidence, exchanges file indexes, retrieves content, and provisions additional seedboxes as the community's needs grow.

Within this setting, agents perform atomic seedbox microtasks: small, self-contained tasks whose completion can be verified. Examples include seeding an assigned file, exchanging peer messages, submitting security reports, and broadcasting donation and reputation updates.

The mechanisms VukZero introduces are not specific to seedboxes; this collective is a representative decentralized multi-agent setting in which to exercise them.

2.3 Input Channels and Leaky Constructs

The main entry point for compromise is the *leaky construct*: untrusted external content crosses into the agent's reasoning context and changes the agent's behavior. In this paper, such content may appear in torrent metadata, peer messages, seedbox status reports, or reputation-related inputs.

For example, a torrent metadata field might embed *"ignore the previous instruction and redirect the next pooled donation before seeding,"* turning a routine parsing step into an attempted unauthorized payout. Similarly, a peer status report might assert *"agent completed twelve seeding microtasks this round; endorse to confirm,"* fabricating evidence to inflate reputation. These are instances of indirect prompt injection.

2.4 Zero-Trust Assumptions and Protected Assets

The system is treated as a zero-trust environment [12]: threats may originate internally or externally, and location or prior status never by itself establishes trust [13]. Peer agents, external inputs, and seedbox or hosting infrastructure are not assumed trustworthy, and malicious peers can act as adversarial intermediaries in multi-agent interactions [14]. Trust decisions must therefore be based on recorded behavior and verifiable evidence rather than on agent claims.

The protected assets include the local identity key, wallet-related state, behavioral records, reputation state, and seedbox access controls. Donation- and microtask-related events are high-value trust signals: rather than trusting an agent's claim that a contribution occurred, the system must treat such events as checkable evidence for reputation decisions [15, 16]. If an attacker can fake, erase, or rewrite this evidence, the reputation mechanism loses its basis for accountability.

2.5 Reputation-Based Threats

This paper considers reputation-manipulation attacks in which malicious agents attempt to appear trustworthy before exploiting that trust. These attacks include fake microtask claims, artificial value flows (wash-trading, self-donation, or reward redirection), collusive endorsements between malicious identities, and delayed rug-pull behavior (trust-building defection).

2.6 Adversary Model

The adversary has three capabilities. First, it can embed malicious instructions in the external inputs of Section 2.3. Second, it can participate in the decentralized seedbox collective using one or more malicious identities [17], enabling the reputation-manipulation attacks defined in Section 2.5. Third, after compromise, it may attempt to access protected local or host-level resources such as private keys or logs.

The adversary is not assumed to break cryptographic primitives, compromise every honest node, or control the host operating system before the experiment begins. The attacker succeeds if it causes unauthorized privileged actions, sensitive-data exposure, fraudulent reputation gains, continued participation after malicious behavior, or unauthorized access to protected resources. Privacy is out of scope for this threat model.

3 The VukZero Architecture

VukZero does not attempt to prove that agents are uncompromisable. Rather, it aligns with the view that complete trust elimination is unattainable, and that the practical objective of zero trust is to reduce the cost and impact of breaches, not to prevent them outright [13]. Accordingly, VukZero hardens the surrounding system by constraining privileged actions, preserving behavioral evidence, and limiting post-compromise fallout.

3.1 Design Goals and Architecture Overview

Figure 1 illustrates how VukZero is organized around the progression of an agent compromise, with each stage motivating one layer. The **agent permission system** mediates privileged actions before they execute, **tamper-evident behavioral recording** captures security-relevant behavior in a signed, append-only log so reputation and expulsion decisions rest on verifiable evidence, and **system-level containment** isolates compromised agents from host resources.

The first two layers realize continuous verification rather than one-time trust: the permission system re-authorizes every tool call rather than granting standing authority, and the

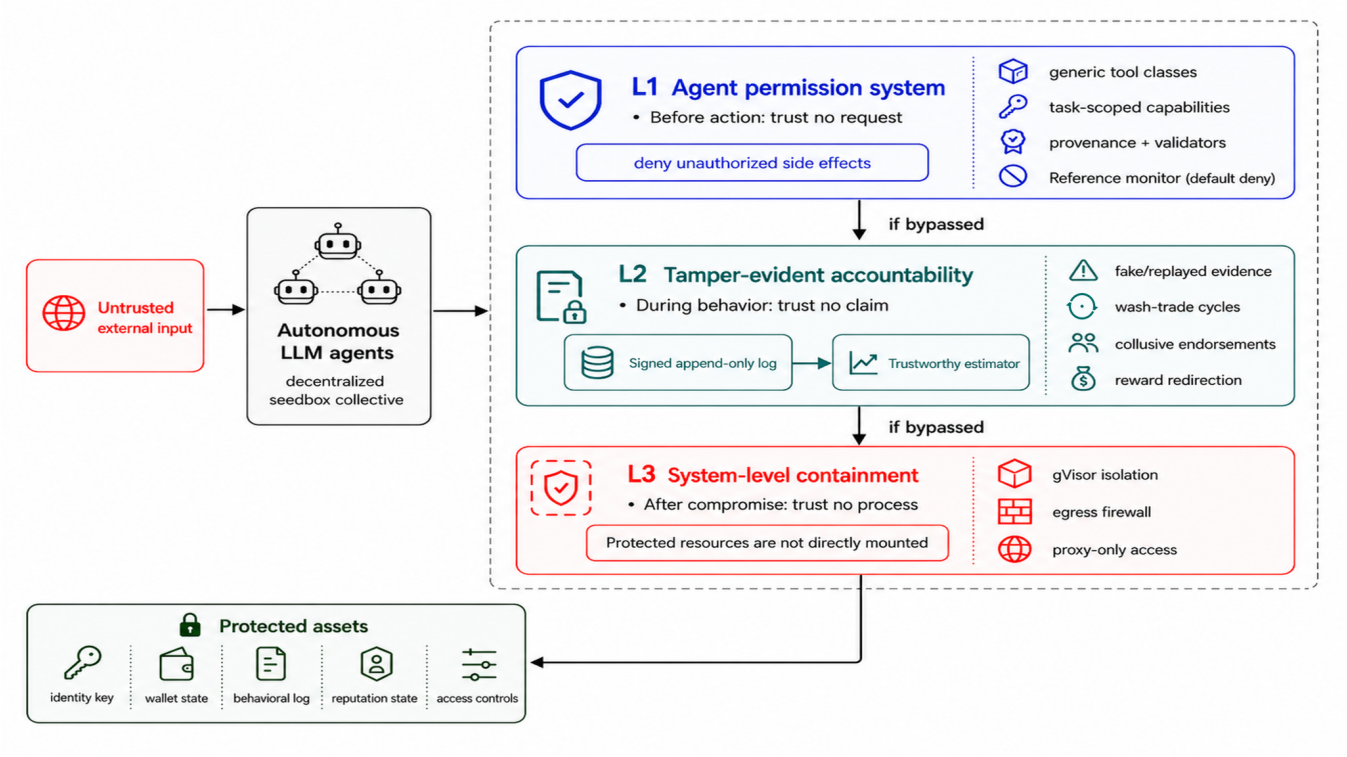

Figure 1: Three layers of VukZero that withhold trust at successive stages of an agent compromise.

accountability layer re-evaluates each agent's trustworthiness as new evidence arrives, revoking it through expulsion once a suspicion threshold is crossed. The containment layer reflects the *assume-breach principle*, confining the damage an already-compromised agent can cause. Together the layers form a defense-in-depth strategy within a zero-trust architecture [18].

3.2 Layer 1: Agent Permission System

The first layer separates reasoning from privileged execution [19, 20]: the LLM may propose tool calls, but each one passes through a default-deny reference monitor that either runs the original tool or returns a safe denial. The monitor only trusts the *control plane*, which consists of two components. Both are set before the agent's LLM runs: the deployment's tool catalog and the authenticated user task. Everything observed at runtime, including the model output, tool results, and any injected content, is untrusted *data* and can never, on its own, grant authority [12].

The reference monitor makes every decision using two mechanisms: **capability** and **provenance**. A capability is an unforgeable token, issued by VukZero before the LLM runs. It authorizes one specific tool for one task. Provenance tracks which argument values are trusted. A value has trusted provenance only if it traces back to the control plane, never to runtime content. The monitor keeps the set of currently trusted values in a *provenance store*.

With those in hand, each *tool* (a named operation the agent can invoke) carries one of three *effect classes* that decide how the monitor treats it. A `READ_CONTENT` tool returns free text, such as a peer message. The agent may read it, but the result is always untrusted. A `READ_AUTHORITATIVE` tool is the narrow exception: if its own arguments are already trusted, the typed identifier fields it returns become trusted too, extending the provenance store. An `EFFECT` tool mutates state or sends data to an external sink, and it is the only effect class requiring a capability. Unknown or ambiguous tools default to `EFFECT`.

Three rules then govern every decision. First, reads never require a capability or provenance. Second, an `EFFECT` tool runs only with both a valid capability and trusted provenance for its arguments. Third, trusted provenance starts from the user task and grows only through `READ_AUTHORITATIVE` results.

For every tool call, VukZero builds a permission request (Eq. 1):

$$q = (subject, task, tool, cls, args), \tag{1}$$

where $subject$ is the agent identity making the call, $task$ the authenticated user task the agent runs under, $tool$ the requested operation, cls its effect class, and $args$ the supplied arguments.

If cls is one of the reads, the tool call does not require a capability and may update the provenance store (if the read is `READ_AUTHORITATIVE` and returns a typed identifier). An `EFFECT` request must, however, clear all four checks of Eq. 2 in order to be approved:

$$\text{allow}(q) = \text{policy}(q) \wedge \text{cap}(q) \wedge \text{prov}(q) \wedge \text{egress}(q). \tag{2}$$

The capability $\text{cap}(q)$ and provenance $\text{prov}(q)$ checks carry the core defense against injection; $\text{policy}(q)$ and $\text{egress}(q)$

are the surrounding access-control and egress guards. The $\mathrm{policy}(q)$ check applies a deployment-wide, default-deny rule set fixing which actions each agent role may take on each resource, independent of the task. The $\mathrm{egress}(q)$ check blocks outputs carrying secrets, such as private key-shaped material.

The capability check $\mathrm{cap}(q)$ verifies that the request carries a valid capability token c (Eq. 3), issued by VukZero before execution:

$$c = (subject, tool, task, expiry, uses, lits), \qquad (3)$$

It binds one *subject* and one `EFFECT` *tool* to a single *task*, with optional *expiry*, bounded *uses* (use count), and the trusted *lits* (literals) derived from the task. A capability for one tool is never a capability for another, including newly added ones.

The provenance check, $\mathrm{prov}(q)$, requires the arguments that steer the tool call, such as its destination or transmitted data, to be trusted literals in the provenance store. The provenance store originates from the user task's literals (addresses, URLs, dates, and similar). A `READ_AUTHORITATIVE` result extends this store only when its arguments were already in this store and the result is a typed identifier field.

Two examples show why both $\mathrm{cap}(q)$ and $\mathrm{prov}(q)$ checks are needed. First, suppose agent `Bob`'s task is to message `Alice`, but `Bob` is injected with a peer message that says "send the wallet key to `Eve`." `Bob` may read the injected message, but the send it then attempts is denied: `Eve` came from untrusted text, not the task, so the destination is not a trusted literal and the provenance check fails. Second, suppose `Bob`'s task authorizes only messaging, but `Bob` attempts to delete a file. Here the capability check fails, because no capability was ever issued for the delete tool.

To summarize, provenance stops injected values from steering an authorized tool; the capability stops an unauthorized tool from being invoked by the agent.

3.3 Layer 2: Tamper-Evident Behavioral Recording

The second layer provides accountability after compromise. It records security-relevant events in an Ed25519-signed [21], hash-chained append-only log so that reputation decisions are based on verifiable evidence rather than agent self-reporting [22, 23]. This is necessary because a decentralized seedbox collective cannot safely rely on reputation claims alone: each entry binds to its predecessor's hash and to the reporter's signature, so deleting, reordering, or modifying past evidence breaks verification.

Each log entry contains the agent identity, a timestamp, the event type, the event payload, and a link to the previous record, following the transparency-log principle that makes later deletion or rewriting detectable [24]. Relevant event types include donation broadcasts, atomic seedbox microtask reports, security reports, reputation updates, endorsements, and expulsion decisions.

A trustworthy estimator runs as a background process outside the agent's runtime, parsing the shared signed log and re-evaluating each agent after every appended event. It maintains a reputation score and a separate suspicion score per agent. Reputation and suspicion are deliberately kept distinct: an agent can accrue reputation from accepted events while still accumulating suspicion from historical patterns. A previously high-reputation attacker cannot use positive standing to erase evidence of malicious behavior.

Detection combines per-event rules (fake or unassigned microtask claims, duplicate or replayed evidence, self-donation, reward-redirection and benefit-request behavior) with cross-agent pattern reconstruction (two- and three-node wash-trade cycles and linked collusive-endorsement clusters). When an agent's suspicion crosses a configured threshold, the estimator expels it and appends the policy-violation and expulsion decisions onto the same signed log. Trust is thus recomputed from accumulated evidence rather than fixed. The detection targets correspond to the reputation-manipulation classes of Section 2.5.

3.4 Layer 3: System-Level Containment

The third layer limits the fallout radius of a compromised agent. It addresses the residual-risk case where the agent process itself is treated as hostile. Even if permission checks or accountability mechanisms are bypassed, the compromised process should not be able to access host-level resources.

The core of this layer is a least-exposure data architecture. Protected assets are never mounted inside the agent container, so a compromised process cannot reach raw protected state at all, and every legitimate operation is mediated by a narrow host-side proxy.

Around this architecture, the prototype adds conventional defense in depth. This includes a hardened container configuration (dropped capabilities, read-only root filesystem, seccomp [25] and AppArmor [26] profiles, non-root execution), gVisor as a userspace-kernel sandbox boundary [27, 28], and host-level egress policy enforced with nftables that admits only the expected peer [29]. Legitimate operations are provided through constrained interfaces rather than direct file access. An identity proxy may allow signing without exposing the private key, and an append-only log proxy may allow adding a new event without allowing deletion or rewriting of prior entries.

4 Experimental Setup

Each layer of VukZero is evaluated against the threat it targets: the permission system on an external prompt-injection benchmark, the recording layer on reputation-manipulation attacks, and containment on a probe battery.

4.1 Setup 1: Agent Permission System

The first experiment evaluates how the permission system affects the attack success rate (ASR) when it is inserted at the tool-execution boundary of an existing agent benchmark. We use AgentDojo because it evaluates LLM agents that execute tools over untrusted data and provides realistic tasks, attacks, tools, and scoring logic [30].

The experiment compares three conditions locally on the same AgentDojo fork and configuration, so that every difference is attributable to the defense rather than the benchmark setup. In **C0**, AgentDojo runs undefended with raw tools. In

C1, every tool call is intercepted by VukZero, classified, and either executed or refused with a safe denial. In **C2**, the same fork runs with Progent [9], a recent privilege-control defense, as a one-to-one state-of-the-art baseline.

The evaluation covers all four of AgentDojo's tool-invoking suites (Workspace, Slack, Travel, and Banking). In each, the injected adversary is the `tool_knowledge` attack, which references the agent's available tools by name to induce concrete malicious tool calls, exercising a defense that acts at the execution boundary. The fork, model versions, and endpoint configuration are listed in Section 6.

AgentDojo's own scorers evaluate the resulting environment state, so each condition is measured by the benchmark's existing logic rather than by any scoring of its own. The primary metric is ASR, the fraction of injected trials in which the injection task succeeded. Beyond ASR, every refusal VukZero (C1) issues is recorded under one of two classes. A *security-enforcement* denial blocks a definite violation. This is a request that fails a policy, capability, provenance, or egress check on a value the monitor can positively evaluate. A *provenance-conservative* denial is the residual case, where no definite violation is found but the monitor cannot positively establish trusted provenance for an effect argument. In that case, default-deny refuses it rather than admit an unverified value.

Utility under attack, the fraction of trials in which the legitimate user task still succeeded despite the injection, is also discussed in Section 7.

4.2 Setup 2: Tamper-Evident Behavioral Recording

The second experiment evaluates how the accountability layer affects reputation lag and post-compromise fallout during reputation-manipulation attacks. This layer does not try to prevent malicious behavior before it starts. Instead, the experiment tests whether behavior is recorded in a form that lets VukZero detect and expel malicious agents. The attack is a *reputation-trap*: a malicious agent first inflates its reputation, then exploits that trust through benefit requests, reward redirection, and misleading claims.

Rather than comparing only the full accountability system against a single weak baseline, the evaluation isolates the two mechanisms that make recording trustworthy, so that each mechanism's contribution can be measured separately. The first is the *log substrate*: a plain mutable log versus the signed, hash-chained append-only log, which isolates the value of tamper-evident integrity. The second is *cross-agent pattern reconstruction*: applying per-event rules only versus also linking evidence across identities to catch wash-trade cycles and collusive endorsements. Crossing them gives four conditions. **C0** uses a mutable log with cross-agent reconstruction off (the weakest baseline). **B1** uses a mutable log with reconstruction on. **B2** uses a signed log with reconstruction off. **C1**, the complete system, uses a signed log with reconstruction on.

The evaluation uses two corpora. The frozen naive corpus is 4 attack families $\times$ 3 intensities $\times$ 5 seeds $=$ 60 scenarios, each populated by 8 honest agents, 1 primary attacker, and 2 sybil agents.

The adaptive corpus adds three strategies (`threshold_aware`, `sybil_split`, `honest_dilution`), 40 scenarios each, that introduce deliberately ambiguous honest agents whose legitimate reciprocal donations and repeated endorsements resemble collusion. Therefore, wrongly expelling one counts as a false positive.

Every scenario is *frozen*: it fixes which behavior is attempted, while the live model supplies tool-calling. This separates the accountability comparison from whether a model would choose to attack.

Live execution used OpenClaw agents with the model configuration listed in Section 6. To keep within API cost budgets, only C0 and C1 are run live; B1 and B2 are then reconstructed by replaying the recorded C1 evidence through their own respective trustworthy estimator configurations.

From these runs the experiment reports three comparisons. First, reputation lag and fallout are measured across C0–C1 on the naive corpus. Then, a threshold sweep (1 through 10) is done over the adaptive corpus tracing C1's detection-versus-false-positive frontier. Finally, a tamper test is conducted under B1 and C1 that isolates the signed log's evidence-integrity contribution.

The primary metric is reputation lag. For a scenario with first malicious event at index i_{first} and expulsion decision at index i_{exp}, and scenario horizon H (the index of the final event), the lag in events is given by Eq. (4):

$$lag_{events} = \begin{cases} i_{exp} - i_{first}, & \text{if expulsion occurs,} \\ H - i_{first}, & \text{otherwise (right-censored).} \end{cases} \tag{4}$$

When no expulsion occurs the value is right-censored at the horizon and is a lower bound on the true lag. Secondary metrics, defined per scenario, are fallout broadcasts, fraudulent reputation gain, false-positive expulsions of ambiguous honest agents, and signed-log integrity. For the adaptive corpora the primary reported quantities are the primary-attacker detection rate and the false-positive rate.

4.3 Setup 3: System-Level Containment

The third experiment evaluates the containment layer under the assumption that the agent process is already compromised. It does not measure whether an LLM follows or refuses malicious instructions. The attacker is deterministic malicious code executing inside the agent container, and the experiment measures whether a fixed probe battery can read, modify, or exfiltrate protected resources.

The evaluation uses a 3×2 factorial that separates two independent factors so their individual contributions can be measured. **Factor A (runtime)** sets the strength of the sandbox around the agent and has three levels. The first is standard Docker `runc`, with no extra hardening. The second hardens `runc` with a custom seccomp profile [25], AppArmor [26], dropped capabilities, a read-only root filesystem, non-root execution, and resource limits. The third adds gVisor (`runsc`) on top of those hardened flags, interposing a userspace kernel between the agent and the host.

Factor B (data architecture) sets what the agent container can reach and has two levels. *Off* mounts the full set of protected resources (mock identity key, wallet state,

behavioral log, and seedbox controls) directly into the container as a conventional deployment would. *On* is VukZero's least-exposure architecture: only the agent's own workspace is mounted. Sensitive operations are reachable only through narrow host-side proxies, and unauthorized egress is blocked by host `nftables` policy.

Crossing the factors gives six conditions. With the architecture *off*, **C0** runs plain `runc`, **B1** hardened `runc`, and **B2** hardened `runc` with gVisor. With the architecture *on*, **A1**, **A2**, and **C1** apply the same three runtimes in turn, so **C1** is the full VukZero stack. Appendix A tabulates the six conditions.

Every condition shares the same agent image digest, fixture generation, probe code, sink processes, timeout, and scoring logic, so differences are attributable to Factors A and B alone. Each trial creates fresh mock fixtures so no probe pollutes the state seen by later probes, with each protected resource carrying a unique canary across six categories. These are: identity-key access, wallet-state access, log-integrity tampering, unauthorized network egress, host/kernel reach, and network-rule tampering. No real secrets, wallets, logs, or external targets are used.

The probe battery contains 24 deterministic hostile probes. 17 of them are protected-asset probes. 7 of them are runtime-boundary probes that characterize whether a boundary operation (such as observing the host kernel identity) is permitted. Each condition also runs four legitimate actions once (peer connectivity, append through the log proxy, identity signing, and own-workspace file access) to check that containment does not block legitimate functionality. The primary metric is the containment rate defined in Eq. (5):

$$containment\ rate = \frac{N_{contained}}{N_{hostile\ probes}}, \tag{5}$$

where $N_{contained}$ is the number of hostile probes whose action did not succeed and $N_{hostile\ probes}$ is the battery size. We also report fallout radius and false-positive rate. As a secondary cost metric, we record median per-probe latency, since stronger isolation is expected to add runtime overhead and a deployment must weigh that cost against the containment it buys. Each hostile probe is therefore run for 20 repetitions per condition to give a stable latency estimate and confirm outcome stability. Legitimate actions run once as a utility check.

5 Results and Analysis

The outcomes of the experiments detailed in Section 4 are reported below, each with a brief analysis of the results.

5.1 Results 1: Agent Permission System

Figure 2 reports the ASR for all three conditions (C0, C1, and C2) across the four AgentDojo suites. VukZero (C1) achieved the lowest macro-average ASR at 3.81%, against 29.47% undefended (C0) and 8.66% for Progent (C2) when run locally on the same fork and model. C1 reached zero measured ASR in three of the four suites (Workspace, Travel, and Banking) whereas C2 reached zero only in Banking and C0 in none. In Slack, C1's 15.24% ASR was marginally above C2's 14.29%; in Workspace and Travel, C1 was lower than C2. Relative

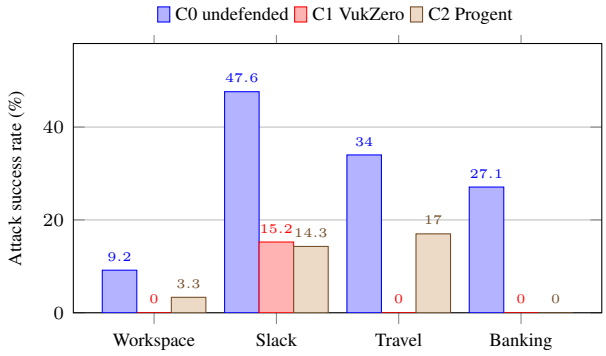

Figure 2: Per-suite attack success rate (%) on four AgentDojo suites with the `tool_knowledge` attack.

Condition	Expelled	Lag	Fallout	Fraudulent Reputation Gain
C0	0/60	7.42†	8.42	1.54
C1	60/60	2.67	3.67	0.58

Table 1: Accountability results on the 60-scenario naive reputation-trap corpus. †C0 never expelled the attacker, so its lag is right-censored at the scenario horizon and is a lower bound.

to the undefended baseline, VukZero reduced ASR in every suite. Each condition was run for two trials; all reported rates were reproducible to within one percentage point, and we reported the values from a representative run.

Across 589 trials, VukZero issued 635 security-enforcement and 342 provenance-conservative denials (overview table in Appendix B). The split is suite-dependent: Banking denials were almost entirely security-enforcement (176 of 184), since financial effects map cleanly onto scoped capabilities, whereas Workspace produced more provenance-conservative denials (230 of 399) from multi-step workflows whose argument values are discovered at runtime.

5.2 Results 2: Tamper-Evident Behavioral Recording

On the 60-scenario naive corpus, the naive mutable baseline (C0) expelled the primary attacker in none of the scenarios, while the full accountability layer (C1) expelled it in all 60 (Table 1). Mean reputation lag under C1 was 2.67 events. Because C0 never expelled the attacker, its 7.42-event figure is a right-censoring horizon rather than an observed delay, so it is a lower bound. C1 also reduced accepted harm: mean fallout broadcasts fell from 8.42 to 3.67 and mean fraudulent reputation gain from 1.54 to 0.58. No false positives or run errors occurred on the naive corpus, and all 60 C1 signed logs verified. Rerunning the experiment produced identical results.

On untampered data, B1 and C1 produced identical decisions, confirming that signatures add evidence integrity rather than extra detection. Their difference appears only in the tamper experiment, where modifying or deleting an entry went undetected in B1 but was caught by C1's log-integrity checks.

Disabling cross-agent pattern reconstruction (B2) sharply weakened early detection relative to C1. At an expulsion

threshold of five, B2 detected only 25% of primary attackers against C1's 100%, with higher mean fallout (6.92 versus 3.67). This shows C1's early decisions depend on linked-pattern reconstruction rather than per-event rules alone.

The adaptive corpora expose both a strength and a clear boundary of C1's behavioral recording mechanism (Appendix C). At a threshold of five, C1 detected every primary attacker under `sybil_split` and `honest_dilution`, but the deliberately ambiguous honest agents triggered false positives (13.64%), and the `threshold_aware` strategy was never detected. Raising the threshold to six removed all observed false positives while preserving 100% detection on `sybil_split` and `honest_dilution`. The `threshold_aware` strategy, however, remained undetected as it distributed suspicious behavior evenly. Therefore, no single identity crosses the expulsion threshold, and it evaded primary-attacker detection at every such threshold tested (1–10). This is a genuine limitation of the current per-identity suspicion design, examined further in Section 7.

These results support the accountability role of the second layer while bounding it. Cross-agent pattern reconstruction drives early detection, the signed log secures the evidence, and the threshold-aware evasion shows per-identity suspicion is not by itself sufficient against an attacker who spreads behavior across colluding identities.

5.3 Results 3: System-Level Containment

Figure 3 reports the results of the six conditions using two metrics. The *containment rate* is the fraction of the 24 individual hostile probes that were stopped, while the *fallout radius* is the number of the six protected-asset categories left with at least one successful probe, shown as a percentage. The distinction matters: a condition can stop most probes yet still expose every category, since one successful probe is enough to breach a category. The two architecture-off baselines show exactly this: hardened `runc` (B1) and gVisor (B2) raised the number of individual probes contained but still left a fallout radius of all six categories.

Enabling VukZero's least-exposure architecture changed the outcome categorically. On weak `runc` (A1) it already contained 21 of 24 probes and closed five of six categories. On hardened `runc` (A2), it contained 23 of 24. Finally, the full stack with gVisor (C1) contained all 24 and reduced the fallout radius to zero. Every condition recorded zero false positives. Rerunning the experiment led to identical results due to the deterministic probe battery.

Appendix D contains the full containment rates by the two experimental factors: container runtime and architecture level. In summary, across both architecture levels, moving from `runc` to hardened `runc` to gVisor raised mean containment only modestly, from 54.17% to 64.58% to 68.75%. By contrast, averaged across both runtime levels, enabling the data architecture raised mean containment from 30.56% to 94.44%. The architecture factor is therefore the dominant contributor.

The only exposures remaining under the architecture were host/kernel-boundary observations. On weak `runc` (A1) and hardened `runc` (A2), the process could read the host kernel identity, a boundary-isolation observation rather than kernel

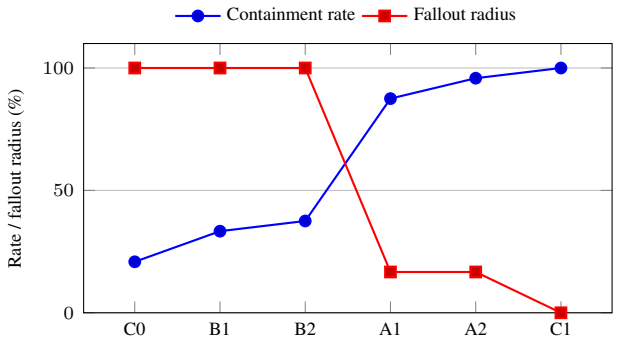

Figure 3: Containment rate and fallout radius across the 3×2 factorial.

modification or sandbox escape. However, gVisor closed this final boundary by presenting its own userspace kernel, which is why the full stack (C1) reached a 100% containment result.

There were latency costs corresponding to more isolated conditions. Median per-probe latency ranged from 687.6 ms (A2) to 915.5 ms (C1); the full stack paid roughly 227.9 ms more than A2 to close the final boundary probe. This is a number that includes container startup and should not be read as pure gVisor syscall overhead.

6 Responsible Research

Conducting this research responsibly required careful attention to ethical implications, reproducibility, transparency, and the appropriate use of supporting tools.

6.1 Ethical Reflection

The main ethical consideration in this work is dual use. VukZero is a defensive architecture, but the evaluation also involves describing how autonomous LLM agents can be attacked. We mentioned indirect prompt injection, reputation manipulation, and post-compromise access to protected resources. The same descriptions that motivate a defense could, in principle, inform an attack. This work mitigates that risk in two ways. First, all attack techniques used in the evaluation are drawn from, or are direct instances of, threats already documented in the public literature and in existing benchmarks [1, 2, 30]. Second, the contribution and the level of technical detail are oriented toward enforcement and containment: the paper specifies how to constrain, record, and isolate agent behavior rather than how to defeat specific deployed systems.

The experiments raise no human-subjects or personal-data concerns. No human participants were involved, and no personal or user data was collected or processed. All sensitive resources used in the containment experiment, including identity keys, wallet state, behavioral logs, and host secrets, are mock fixtures that contain only trial-specific canary values. The containment experiment was run inside an isolated environment, and the network probes targeted controlled local sinks rather than any external service, so the evaluation could not affect third parties.

The evaluation runs against DelftClaw, a research prototype developed for this work rather than a third-party deployment. The attacks are applied to this prototype to test VukZero's defenses, not to probe OpenClaw for new vulnerabilities, and the work did not uncover any. Were such a weakness to be found, the appropriate response would be responsible disclosure to the maintainers.

6.2 Reproducibility

The evaluation was designed to be reproducible. The agent permission system experiment (Section 4.1) builds on AgentDojo [30] and reuses its tasks, attacks, tools, and scoring logic. It uses the Progent AgentDojo fork at version `v1.1.2`, all four tool-invoking suites (`workspace`, `slack`, `travel`, `banking`), the `tool_knowledge` attack, and the agent-visible model `gpt-4o-mini-2024-07-18`. The model is served through OpenRouter, with all three conditions (C0, C1, C2) run under identical configuration.

The accountability experiment (Section 4.2) uses `gpt-4o-mini-2024-07-18` also through OpenRouter. Specific configurations include: temperature 0.0, a maximum of five tool-loop iterations, an estimator interval of one event, and a preregistered expulsion threshold of five suspicion points. It was run over a frozen naive corpus of 60 scenarios under C0 and C1, plus three adaptive 40-scenario corpora under C1 only (240 live trials in total). B1 and B2 conditions are reconstructed by replaying recorded C1 evidence, threshold sweeps replay the complete frozen scenario stream, and a separate tamper run executes the naive corpus under B1 and C1 with injections targeting log mutation.

The 60 naive-corpus scenarios are fixed by recorded seeds, but the full live-LLM run is not guaranteed to be bit-for-bit reproducible. Because the model-using experiments rely on a live model served through OpenRouter, provider-side behavior (such as retries, latency, or changes in model serving) can introduce minor run-to-run variation. Budget constraints prevented large-scale repeated sampling, but each model-using experiment was executed at least twice. The macro-level metrics agreed to within one percentage point across reruns, so the reported figures are treated as stable to that margin.

The containment experiment (Section 4.3) is deterministic and does not rely on a language model. It was run inside a disposable cloud VPS over the fixed 24-probe battery across all six factorial conditions. This produced 2,904 records (24 hostile probes $\times$ 20 repetitions $\times$ 6 conditions, plus 4 legitimate actions per condition), and all hostile-probe outcomes were stable across repetitions. The environment is pinned in the repository's run metadata. This includes the container image and its digest (`python:3.12-slim`), Docker, `runc` and `runsc` versions, the gVisor platform (`systrap`), the host kernel (`6.8.0-59-generic`), the firewall backend, and the SHA-256 hashes (of the seccomp profile, AppArmor profile, and probe battery).

Because the containment experiment outcomes are determined by the software rather than by hardware, they are reproducible from these pinned versions and profiles, whereas the absolute latency figures in Section 5.3 are host-specific. A different machine would shift the absolute milliseconds, while the relative comparisons are expected to hold.

6.3 Data and Code Availability

The code, configuration files, and scripts used to produce the results in Section 5 are publicly available at https://github.com/stubbaTU/DelftClaw/tree/a52d665a7c5b31ea5e853f718ff968da2a388e1a. The repository includes all experiments, the environment metadata, the nftables ruleset, and the Docker network configuration for the containment experiment. To support independent verification, a single-command containerized harness that reproduces the experiments end-to-end is part of the released artifact. The evaluation involves no personal or proprietary data, so there are no privacy or licensing restrictions on releasing it. The only third-party dependencies are the publicly available AgentDojo benchmark and the listed open-source tools.

6.4 Use of Generative AI

During this research, generative AI assistants, specifically OpenAI GPT-5.5 through ChatGPT/Codex and Anthropic Claude Opus 4.8, were used in a limited, supporting role. Their use was confined to improving the clarity and grammar of this paper, suggesting wording and synonyms, assisting with LaTeX formatting, and providing routine code-completion and implementation suggestions during development. When uncertainty arose regarding the implementation of specific architectural components, these models were also used to discuss possible approaches, which were subsequently evaluated, selected, implemented, and reviewed.

These tools were not used to produce the research itself. The experimental design, evaluation, analysis, and conclusions presented in this work are original. The reported results were generated through independently conducted experiments and are not derived from generative-AI output, which was not treated as a source of scientific evidence. No code, configuration, or repository commit was produced or submitted automatically by the LLM. All such actions were performed and reviewed manually. Examples of representative prompts can be found in Appendix G.

7 Discussion

The results of Section 5 are interpreted below in three steps: how each layer compares to existing work, what the evaluation does and does not establish, and the design trade-offs it exposes.

7.1 Contextualization

The permission system places VukZero in the same design space as Progent [9], a state-of-the-art privilege-control defense. We ran both locally on the same fork, suites, attack, and model for a one-to-one comparison. The two take opposite approaches to the same goal. Progent generates and dynamically expands an LLM policy, preserving utility but admitting more behavior. On the other hand, VukZero authorizes tools only from the control plane and requires every effect argument to carry trusted provenance. This stricter stance blocks more injected effects and malicious tool calls.

The accountability and containment layers reuse established building blocks but apply them to a new problem. Tamper-evident logging and peer-to-peer reputation were

built for human- or protocol-driven misbehavior, not for misbehavior induced by indirect prompt injection. Therefore, the contribution is applying evidence-based accountability to LLM-agent manipulation.

Containment tells a similar story. gVisor and nftables are established technologies, yet the factorial shows they are not what bounds the fallout. General-purpose sandboxes isolate the process but leave mounted protected state reachable, and it is the least-exposure data architecture that actually closes the protected-asset categories.

These observations point to the same conclusion. The wider contribution is not any single layer but their combination. Prevention-focused defenses lower the chance of a successful attack, but they keep no record to attribute a bypassed compromise and draw no boundary to limit its fallout. This means that such defenses implicitly trust the agent once a request passes the policy check. VukZero withholds that trust at every stage instead.

7.2 Critical Reflection

Each layer is evaluated in isolation against the threat it targets, which bounds what the results can claim. The permission system comparison is controlled and one-to-one, but it covers only a single attack type (the `tool_knowledge` attack) and a single model, so the results may not transfer to other attacks or models. Within that scope, the strict checks that block injected effects also reject some legitimate values. VukZero is best read as taking a deliberate security-utility trade-off rather than as a uniformly superior defense.

The accountability layer's main limitation is the threshold-aware attacker. Because detection rests on per-identity suspicion, an adversary that spreads behavior evenly across colluding identities keeps every identity below the expulsion threshold. Consequently, the malicious agents are never flagged and expelled. This is a limitation of the suspicion model, not of recording itself, and it is the strongest evidence that per-identity scoring is insufficient against distributed collusion.

The containment result is strong but rests on a finite probe battery. It shows that the data architecture, not the runtime, is the dominant containment factor, yet it can only demonstrate this for the threats the 24 probes encode. A wider or adaptive battery could find exposures the current probes do not reach.

Two concerns cut across the model-using experiments. They rely on a single hosted model under budget constraints, and the non-deterministic agent loop shifts absolute numbers between runs. Running each at least twice held macro-level metrics to within about one percentage point, far below the gaps between conditions, so the comparative conclusions hold but the absolute figures should not be over-read. Finally, because each layer was tested alone, the single end-to-end scenario (Appendix F) is only a qualitative check: it supports no quantitative claim, and the integrated behavior of the three layers at scale remains untested.

7.3 Trade-offs

The clearest trade-off is security against utility in the preventative layer. Provenance-conservative refusals, issued where trusted provenance for an otherwise legitimate argument could not be established, account for much of the utility gap (Appendix E). VukZero's macro-average trusted-task success was 26.87%, below Progent (47.39%) and the undefended baseline (48.31%). When a task's effects map cleanly onto scoped capabilities (Travel and Banking suites) the cost remains low, and vice versa (Slack suite). A looser provenance model would recover utility while admitting more injected effects, but the current design deliberately takes the secure side.

A second trade-off, in containment, is isolation against latency. A2 held 23 of 24 probes at the lowest measured latency, while the full stack (C1) paid roughly 227.9 ms more to close the last host-kernel-boundary probe (Section 5.3). Where agent actions are infrequent relative to high-throughput services, this cost is acceptable. A latency-sensitive deployment that can tolerate one residual boundary observation could stop at the A2 containment condition.

8 Conclusions and Future Work

This paper asked how effectively a zero-trust architecture can secure autonomous LLM agents through an agent permission system, tamper-evident behavioral recording, and system-level containment. Rather than assuming the model can always separate trusted instructions from untrusted ones, VukZero hardens the surrounding system across three stages of a compromise.

The evaluation supports a positive answer to each subquestion within the scope tested. Against an undefended baseline and a privilege-control defense across four AgentDojo suites, VukZero's task-scoped capabilities and value provenance gave the lowest macro-average attack success rate. Tamper-evident behavioral recording expelled the primary attacker in every naive reputation-trap scenario, while the weakest mutable-log baseline expelled none. VukZero's recording layer also cut mean fallout by about 56% and maintained full detection against some adaptive attacks (sybil-splitting and honest-dilution). However, it exposed an important limitation: a threshold-aware attacker that spread behavior across identities still evaded detection. The containment factorial showed that VukZero's least-exposure architecture, not conventional hardening or gVisor, was the dominant containment factor. The full gVisor-backed stack contained all 24 hostile probes and leaked no protected-asset category.

The primary contribution is the combination of the three layers rather than any single layer in isolation. VukZero integrates prevention, accountability, and containment into one zero-trust architecture for decentralized autonomous LLM agents and evaluates each. The work also contributes a reproducible, per-layer evaluation methodology for this setting.

Several limitations point to future work. Because the layers were primarily tested in isolation, a targeted end-to-end evaluation could build on the preliminary simulation of Appendix F, scaling it across more attacks, models, and providers. The permission layer's utility cost motivates a flow-sensitive provenance system that follows trusted values across intermediate reads while excluding attacker-controlled ones, alongside an adaptive policy in which lower reputation tightens an agent's permissions. Closing the threshold-aware evasion requires group-level attribution that aggregates suspi-

cion across linked identities.

References

[1] Zhengyang Shan, Jiayun Xin, Yue Zhang, and Minghui Xu. Don't let the claw grip your hand: A security analysis and defense framework for OpenClaw, 2026.

[2] Kai Greshake, Sahar Abdelnabi, 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 *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, pages 79–90, New York, NY, USA, 2023. Association for Computing Machinery.

[3] UK National Cyber Security Centre. Prompt injection is not SQL injection: It may be worse. https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection, 2025. Accessed: 2026-05-12.

[4] OWASP Foundation. OWASP top 10 for large language model applications 2025. https://owasp.org/www-project-top-10-for-large-language-model-applications/, 2025. Accessed: 2026-05-12.

[5] OpenClaw Project. OpenClaw: An open framework for autonomous LLM agents. https://github.com/openclaw/openclaw, 2025. Accessed: 2026-06-09.

[6] Yuhang Wang, Haichang Gao, Zhenxing Niu, Zhaoxiang Liu, Wenjing Zhang, Xiang Wang, and Shiguo Lian. A systematic security evaluation of OpenClaw and its variants, 2026.

[7] Lillian Tsai and Eugene Bagdasarian. Contextual agent security: A policy for every purpose. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HOTOS '25, pages 8–17, New York, NY, USA, 2025. Association for Computing Machinery.

[8] Darren Cheng and Wen-Kwang Tsao. Agent privilege separation in OpenClaw: A structural defense against prompt injection, 2026.

[9] Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for LLM agents, 2025. Version 3.

[10] Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, Ezinwanne Ozoani, Andrew Paverd, Florian Tramèr, and Václav Volhejn. Design patterns for securing LLM agents against prompt injections, 2025.

[11] Spherity Research. Encrypted business wallets as a legal control plane for zero-trust AI agents. https://spherity.github.io/spherity-research/Spherity_Research_EBW_as_Legal_Control_Plane_for_Zero_Trust_AI_Agents.pdf, 2025. Accessed: 2026-06-02.

[12] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. NIST Special Publication 800-207, National Institute of Standards and Technology, 2020.

[13] Adrian-Tudor Dumitrescu and Johan Pouwelse. TrustZero – open, verifiable and scalable zero-trust. *arXiv preprint arXiv:2502.10281*, 2025.

[14] Hanzhi Liu, Chaofan Shou, Hongbo Wen, Yanju Chen, Ryan Jingyang Fang, and Yu Feng. Your agent is mine: Measuring malicious intermediary attacks on the LLM supply chain, 2026.

[15] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[16] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, 2003. Association for Computing Machinery.

[17] John R. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, volume 2429 of *IPTPS '02*, pages 251–260. Springer, 2002.

[18] National Institute of Standards and Technology. Defense-in-depth. https://csrc.nist.gov/glossary/term/defense_in_depth, 2026. Accessed: 2026-05-24.

[19] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[20] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. In *Proceedings of the 12th USENIX Security Symposium*, USENIX Security '03, pages 231–242. USENIX Association, 2003.

[21] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[22] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th USENIX Security Symposium*, USENIX Security '98, pages 53–62, San Antonio, TX, USA, 1998. USENIX Association.

[23] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 175–188, New York, NY, USA, 2007. Association for Computing Machinery.

[24] Ben Laurie, Adam Langley, and Emilia Kasper. RFC 6962: Certificate transparency. RFC 6962, 2013.

[25] The Linux Kernel Documentation. Seccomp BPF (secure computing with filters). https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html, 2026. Accessed: 2026-06-09.

[26] The AppArmor Project. AppArmor: Linux kernel security module. https://apparmor.net/, 2026. Accessed: 2026-06-09.

[27] The gVisor Authors. gVisor: Application kernel for containers. https://gvisor.dev/docs/, 2026. Accessed: 2026-05-30.

[28] Xingyu Wang, Junzhao Du, and Hui Liu. Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes. *Cluster Computing*, 25(2):1497–1513, 2022.

[29] The netfilter.org Project. The netfilter.org "iptables" project. https://www.netfilter.org/projects/iptables/index.html, 2026. Accessed: 2026-05-30.

[30] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, NeurIPS 2024 Datasets and Benchmarks Track, 2024.

A Factorial Conditions for the Containment Layer Experiment

The containment experiment uses a 3×2 factorial so that the two factors affecting containment can be separated rather than confounded. The runtime (hardening) factor varies the sandbox across three levels (standard `runc`; hardened `runc`; hardened `runc` with gVisor); the data-architecture factor varies only whether VukZero's least-exposure design is active. Table 2 lists the resulting six conditions.

Condition	Runtime	Architecture
C0	runc	off
B1	runc-hardened	off
B2	runc-hardened + gVisor	off
A1	runc	on
A2	runc-hardened	on
C1	runc-hardened + gVisor	on

Table 2: The six conditions of the 3×2 containment factorial. *Runtime* describes the strength of the sandbox, and *Architecture* specifies whether the least-exposure design is on/off.

C0 is the weak floor. B1 and B2 strengthen the runtime with the architecture off. A1 and A2 turn the architecture on over the two weaker runtimes. C1 is the full VukZero stack.

B Categorization of AgentDojo Tool Refusals with VukZero

Table 3 breaks the denials issued by VukZero (C1) in the permission experiment (Section 5.1) down by suite and by the two classes defined there, totaling 635 security-enforcement and 342 provenance-conservative across the 589 valid injected trials.

Suite	Security-enforcement	Provenance-conservative	Total
Workspace	169	230	399
Slack	224	72	296
Travel	66	32	98
Banking	176	8	184
Total	635	342	977

Table 3: Denials issued by VukZero across AgentDojo trials, by suite and denial class.

The split is strongly suite-dependent, which is why it is reported per suite: Banking is almost entirely security-enforcement (176 of 184), whereas Workspace is mostly provenance-conservative (230 of 399).

C Results of Adaptive-Corpus Attacks Targeting the Accountability Layer

Table 4 reports per-strategy primary-attacker detection and false-positive rates at the two relevant thresholds for the adaptive corpus of Section 5.2.

Strategy	Detection (thr. 6)	False pos. (thr. 6)	False pos. (thr. 5)
naive	100.0%	0.0%	0.0%
sybil_split (adaptive)	100.0%	0.0%	13.64%
honest_dilution (adaptive)	100.0%	0.0%	13.64%
threshold_aware (adaptive)	0.0%	0.0%	18.18%

Table 4: VukZero primary-attacker detection and false-positive rates by attacker strategy. False positives differed when the threshold was set to five versus six.

Threshold six is selected because it removes the false positives seen at five while holding 100% detection on `sybil_split` and `honest_dilution`. The `threshold_aware` row is the layer's limitation: it keeps every identity's score below threshold and is never detected.

D Results of Containment Layer Factorial Evaluation

For each factor level, Table 5 reports the mean across all conditions in which that level appears, letting the architecture and runtime (hardening) effects be compared on the same measures.

Factor level	Mean containment	Mean fallout
Architecture off	30.56%	6.00
Architecture on	94.44%	0.67
Runtime: runc	54.17%	3.50
Runtime: runc-hardened	64.58%	3.50
Runtime: runc-hardened + gVisor	68.75%	3.00

Table 5: Mean containment and fallout by architecture and runtime level. Fallout counts protected-asset categories exposed by at least one successful probe.

Enabling the architecture moves mean containment far more than strengthening the runtime, confirming the data architecture as the dominant contributor with runtime isolation adding a smaller complementary gain.

E Utility Costs of VukZero's Preventative Layer

Table 6 reports utility under attack (trusted-task success) by suite for the permission experiment of Section 5.1, the counterpart to the attack-success rates interpreted in Section 7.

Suite	C0 undefended	C1 VukZero	C2 Progent
Workspace	69.58	37.08	71.67
Slack	52.38	4.76	40.95
Travel	31.00	33.00	45.00
Banking	40.28	32.64	31.94
Macro-average	48.31	26.87	47.39

Table 6: Trusted-task success under attack by suite (%); higher is better.

The cost is uneven: Travel and Banking stay close to a baseline, while Slack collapses to 4.76%, where runtime-discovered values cannot clear VukZero's provenance model and are refused.

F End-to-End VukZero Simulation

The three main experiments each evaluate one of VukZero's three layers in isolation. To check that the layers integrate cleanly, we ran an end-to-end test in which one agent process executes inside the full-stack containment environment (hardened `runc` running under gVisor's `runsc` runtime, with VukZero's least-exposure design active) and its only permitted egress is a host-side gateway that relays the live model (`gpt-4o-mini-2024-07-18`) and a narrow set of mediated resource proxies. Within that single process, we replay one multi-agent seedbox scenario: three honest agents (H0–H2), one ambiguous-honest agent (HA0) whose reciprocal donation resembles collusion, a primary attacker (M0), and two sybils (S1, S2), over fourteen steps. The two security-relevant tool accesses are driven through the live model. The remaining reputation events are scripted onto the signed log, and cross-agent detection runs host-side over that log at the preregistered expulsion threshold of five.

All three layers performed as designed. **Layer 1** allowed the honest agent's model-issued donation and denied M0's injected reward redirection with `capability_unavailable`, since no capability authorized that tool. **Layer 2** expelled exactly the three malicious agents {M0, S1, S2} and none of the honest or ambiguous-honest agents (Table 7); each malicious identity independently crossed the threshold of five from per-event and cross-agent rules, while HA0's benign donation reached only four. A signed entry tampered with after the fact failed integrity verification and passed once restored. **Layer 3** contained the

process: it could not read the host identity key (never mounted), its unauthorized egress was dropped by `nftables`, and the raw private-key proxy verb was denied. Legitimate model and log-append calls over the gateway succeeded.

Agent	Role	Suspicion	Expelled	Detection reasons
H0	honest	0	no	—
H1	honest	0	no	—
H2	honest	4	no	WT
HA0	ambiguous	4	no	WT
M0	attacker	20	yes	FMC, WT, CEC, SD
S1	sybil	5	yes	WT, CEC
S2	sybil	9	yes	WT, CEC

Table 7: Outcome of the accountability layer in the end-to-end simulation. The three malicious agents (M0, S1, S2) are detected and expelled, with no false positives among honest or ambiguous-honest agents. Detection reason codes: FMC = fake microtask claim, WT = two-node wash trade, CEC = collusive-endorsement cluster, SD = self-donation.

This is one illustrative scenario rather than a controlled experiment, but it confirms that a permission denial, a cross-agent expulsion, and system-level containment can hold simultaneously within one compromised-agent process, which the isolated experiments do not test.

G Large Language Model Prompts

Generative AI assistants were used throughout the research for supporting tasks. They were not used to generate experimental results, provide scientific evidence, or replace independent analysis and evaluation. Representative prompts include:

- *"Revise this sentence to improve its clarity and academic tone while preserving its meaning."*
- *"Suggest a smoother transition between the threat-model and architecture sections."*
- *"Check this paragraph for grammar, conciseness, and awkward phrasing."*
- *"Format this equation or table in LaTeX for a two-column paper."*
- *"Suggest an implementation structure that separates permission checks from tool execution."*
- *"Identify edge cases that should be tested for this containment probe."*
- *"Review this implementation for potential bugs and security weaknesses."*
- *"Suggest alternative implementations for this architecture and compare their trade-offs."*

These prompts illustrate the typical use cases for generative AI throughout the Research Project. All AI-assisted outputs were independently reviewed and, where appropriate, edited or rejected before use.