



A Trusted Provenance and TEE-Based Reference Architecture for Intent-Execution Binding in LLM Agents

Haewon Byeon 

Department of Future Technology, Korea University of Technology and Education (KOREA TECH), Cheonan 31253, South Korea

Corresponding Author Email: bhwpuma@naver.com

Copyright: ©2026 The author. This article is published by IIETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/ijssse.160810>

ABSTRACT

Received: 21 April 2026

Revised: 2 July 2026

Accepted: 26 August 2026

Available online: 31 August 2026

Keywords:

Large Language Model agent security, Trusted Execution Environment, provenance, complete mediation, indirect prompt injection

The rapid expansion of autonomous Large Language Model (LLM) agents introduces critical security risks, particularly the confused deputy problem caused by orchestrator compromise or indirect prompt injection (IPI). Traditional defenses rely on in-band filtering, which fails to prevent a subverted orchestrator from bypassing security checks. We propose Intent-Execution Integrity Binding (I-EIB), a reference architecture designed to enforce complete mediation by isolating execution capabilities within a Trusted Execution Environment (TEE). By leveraging Amazon Web Services (AWS) Nitro Enclaves and Key Management Service (KMS)-based capability sealing, I-EIB ensures that sensitive API keys and session tokens remain inaccessible to the host. These credentials are only released when a TEE-based verifier confirms that the processing provenance matches a user-signed domain-specific language (DSL) policy and a predefined chain layout. We formalize the system's security invariants and provide a structured proof sketch demonstrating how I-EIB maintains intent-execution integrity even under full host compromise. Compared with a host-mediated baseline that performs policy checking without enclave isolation or attestation-gated capability release, the proposed architecture introduces a mean additional overhead of 14.2 ms. Experimental results show a 100% detection rate for bounded step-omission attacks and a false positive rate below 0.1% after normalization. A workflow-oriented case study and computational-path analysis further indicate that the architecture is feasible for high-stakes domains such as finance and healthcare, while still carrying deployment constraints associated with TEE availability, policy precision, and wrapper management.

1. INTRODUCTION

The rapid integration of Large Language Models (LLMs) into autonomous agent frameworks marks a transformative shift in artificial intelligence [1]. AI no longer merely summarizes text or answers queries; it executes code, manages financial transactions, and interacts with complex third-party APIs [2]. These “agentic” capabilities rely on an orchestrator—the central component responsible for planning, tool selection, and execution [3]. However, this increased autonomy introduces a fundamental security gap [4]. As agents consume data from untrusted external sources, they become vulnerable to indirect prompt injection (IPI) [5]. In such scenarios, an attacker embeds malicious instructions within a web page or a document, tricking the agent into performing unauthorized actions that deviate from the user's original intent [6].

This vulnerability often manifests as the “Confused Deputy” problem [7]. Here, the agent possesses high-level privileges but lacks the internal logic to distinguish between a legitimate user command and a subverted instruction hidden in the context [8]. Traditional defense strategies, such as prompt engineering or output filtering, offer only a thin layer of

protection [9]. We argue that these methods are insufficient because they operate “in-band”—meaning the security signals are part of the same text stream that an attacker can manipulate [10]. If a malicious input can override the system prompt, it can just as easily bypass a filter or a watermark [11].

Existing research has proposed mechanisms like Signed-Prompt or Prompt Fencing to create trust boundaries [12]. While these are steps in the right direction, they frequently fail when the orchestrator itself is compromised [13]. If an attacker gains control of the host environment or the orchestrator process, they can simply skip the validation steps or forge the logs [14]. Current architectures lack “complete mediation”—the security principle that every access to a sensitive resource must be checked for authority [15]. More specifically, the unresolved limitation in current prompt-level or Trusted Execution Environment (TEE)-adjacent defenses is that they still do not fully bind user-approved intent to the final privileged execution path once the orchestrator or host is subverted. Without a hardware-anchored mechanism that enforces this binding at execution time, the link between what the user intended and what the agent actually does remains fragile and easily severed [16].

A concrete example helps clarify how this failure unfolds in

practice. A user may authorize an agent to summarize a financial report or prepare a healthcare workflow recommendation, but a malicious instruction embedded in retrieved content may try to redirect the agent toward a credential-bearing API call, an unauthorized transfer, or another unsafe downstream action. In a conventional architecture, the orchestrator may still invoke the privileged tool because it already holds the required token or key. In Intent-Execution Integrity Binding (I-EIB), such an attempt is blocked because the TEE-side verifier checks both whether the final action remains within the user-signed domain-specific language (DSL) policy and whether all required intermediate provenance steps were completed before any capability is released.

We propose I-EIB to address these systemic weaknesses [17]. I-EIB moves away from passive detection and toward hardware-enforced mediation [18]. By utilizing TEE, specifically Amazon Web Services (AWS) Nitro Enclaves, we isolate the execution logic and the underlying credentials from the potentially compromised host [19]. The core of our approach lies in “Capability Sealing.” We ensure that the API keys and session tokens required for tool use are cryptographically locked [20]. They are only released if a verifiable provenance chain—a digital record of every processing step—matches a pre-defined, user-signed policy

[21].

This paper provides three primary contributions [22]. First, we define a reference architecture that establishes a “Closed System Boundary” through the use of Tool Wrappers and mutual Transport Layer Security (mTLS) pinning, preventing host-level bypasses [23]. Second, we formalize the security invariants of the system, providing a proof sketch that demonstrates why unauthorized execution is impossible under our defined threat model [24]. Finally, we implement a prototype on AWS Nitro Enclaves and evaluate its performance [25]. In addition to latency and attack-detection results, we also interpret the main computational path and present a workflow-oriented case study to make the operational meaning of the architecture more concrete. Our results indicate that I-EIB detects 100% of step-omission attacks while introducing a mean additional latency overhead of approximately 14.2 ms [26]. We suggest that this architecture provides a practical and robust foundation for deploying autonomous agents in high-stakes environments like finance and healthcare, where accountability and intent-integrity are non-negotiable [27]. Figure 1 also explicitly distinguishes the trusted and untrusted zones of the architecture, showing that all privileged execution must pass through the enclave-side verification path before capability release.

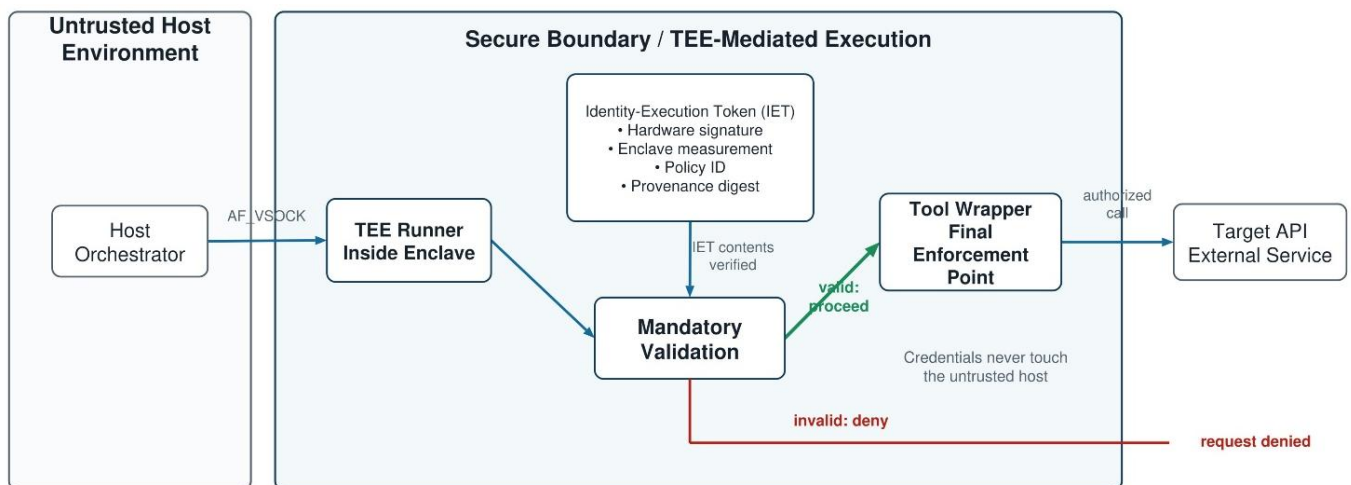


Figure 1. Secure Identity-Execution Token (IET)-based interaction acting as the final enforcement point

2. SYSTEM BOUNDARIES AND TRUST MODEL

Establishing a robust security posture for autonomous agents requires an explicit definition of trust perimeters [28]. In conventional LLM deployments, the orchestrator typically operates with unfettered access to the host environment, managing sensitive credentials in either plain text or volatile memory [29]. We argue that this design is fundamentally fragile; a single compromise at the host level grants an adversary full control over the agent's capability set [30]. I-EIB addresses this vulnerability by defining a closed system boundary where the TEE runner acts as the sole, non-bypassable gateway for all high-privilege operations [1].

Within this model, the host operating system, the orchestrator process, retrieved external content, and downstream third-party APIs are treated as untrusted. By contrast, the enclave verifier, attestation logic, nonce store, capability-release mechanism, and enclave-held ephemeral

keys are treated as trusted. Tool Wrappers operate at the security perimeter: they are not part of the host trust base, but they enforce externally visible access decisions only after successful enclave-backed verification.

2.1 Capability taxonomy and isolation

We categorize the execution capabilities (C) required by the agent into two distinct tiers based on their cryptographic lifecycle and binding mechanisms [29]. This classification, detailed in Table 1, allows us to apply the principle of least privilege while maintaining the operational flexibility required for complex agentic workflows [3].

The first tier involves Key Management Service (KMS)-Sealed Secrets [6]. These represent long-lived assets stored as encrypted ciphertext within a cloud-managed KMS [4]. We utilize the KMS RecipientAttestation ImageSha384 condition key to bind these secrets to the Enclave's identity [27]. This

ensures that the KMS provider only releases the decrypted secret to a caller that presents a valid Nitro attestation document matching the expected Platform Configuration Register (PCR) values [27]. Notably, this mechanism prevents

even a root-level host administrator from retrieving the secrets, as they cannot replicate the cryptographic identity provided by the Nitro hypervisor [10].

Table 1. Taxonomy of agent execution capabilities (C)

Feature	Tier 1: Key Management Service (KMS)-Sealed Secrets	Tier 2: Attestation-Aware Tokens
Primary examples	Static API keys, mutual Transport Layer Security (mTLS) private keys	OAuth 2.0/OpenID Connect (OIDC) session tokens
Storage mechanism	Cloud KMS (encrypted blobs)	Trusted Execution Environment (TEE) secure memory (ephemeral)
Access logic	KMS RecipientAttestation ImageSha384	Identity Provider (IdP)-verified attestation document
Binding level	Platform Configuration Register (PCR)-based hardware identity	Dynamic enclave measurement
Revocation	Policy update/key rotation	Token expiry/session revocation

The second tier comprises Attestation-Aware Short-Lived Tokens [27]. For modern services utilizing dynamic authentication, the TEE runner generates an ephemeral key pair upon startup [12]. It then submits the public key along with its attestation document to an attestation-aware Identity Provider (IdP) [27]. The IdP verifies the Enclave’s integrity and issues a token cryptographically bound to that specific Enclave instance [27]. We suggest that this dual-tier approach effectively mitigates credential theft; even if a token is extracted from host memory, it remains useless without the associated private key isolated within the TEE [15].

This two-tier split is sufficient for the reference architecture because privileged execution reduces operationally to either long-lived sealed trust material or short-lived delegated access credentials. However, cross-tier interaction still introduces risk if not controlled carefully. For example, a weak Tier 1 policy could permit issuance of a valid Tier 2 token under an outdated measurement, or a Tier 2 token might remain usable briefly after a Tier 1 revocation event. To mitigate these cross-tier risks, token issuance is made dependent on successful enclave-side verification at the time of use, and policy updates are designed to prevent future token minting after revocation.

2.2 Tool Wrappers and non-bypassable gateways

A primary hurdle in securing LLM agents is the inherent heterogeneity of third-party APIs [16]. Many legacy or SaaS-based tools lack the infrastructure to natively verify TEE-specific Identity-Execution Tokens (IETs) [29]. To address this without compromising the closed boundary, we introduce Tool Wrappers as security-hardened proxies [18].

We design these wrappers to reside between the TEE and the target API, acting as the final enforcement point [19]. The wrapper holds the actual downstream credentials, ensuring they never reside on the untrusted host [29]. To execute a tool, the TEE runner initiates a request via Address Family Virtual Socket (AF_VSOCK) [21]. This request must include an IET—a hardware-signed token containing the Enclave’s measurement, the current policy ID, and a digest of the provenance chain [28]. The Tool Wrapper performs a mandatory validation of the IET’s signature and expiration [23]. If the validation fails or the measurement is unrecognized, the wrapper denies the request [24]. In deployment, the Tool Wrapper operates as a hardened proxy service placed between the enclave and the downstream API endpoint. It validates the IET, terminates the enclave-authenticated channel, and forwards requests only after attestation and policy checks succeed. Although the wrapper is a critical enforcement component, it need not be a single point of failure: it may be

replicated as a stateless service behind ordinary service-redundancy mechanisms, while downstream credentials remain inaccessible to the untrusted host. This interaction is illustrated in Figure 1.

2.3 Adversary model and root of trust

Our model assumes a “Hostile Host” scenario, where an adversary has achieved full root access to the host operating system [27]. This allows the attacker to inspect network traffic, manipulate Virtual Socket (VSOCK) packets, and modify any orchestrator logic residing outside the Enclave [1]. We also assume the adversary may attempt “omission attacks” by skipping mandatory security steps in the agent’s planning phase [29].

Despite these capabilities, we define our Root of Trust based on three immutable pillars [2]:

- (1) Hardware isolation: The AWS Nitro Enclaves provide a side-channel-resistant, isolated memory space [1]. We consider the hypervisor-level attestation mechanism tamper-proof [27].
- (2) User-Signed Intent: The user’s private key, used to sign the initial DSL policy, remains isolated from the agent’s execution environment (e.g., on the user’s local device) [29].
- (3) KMS Integrity: The cloud provider’s KMS correctly enforces attestation-based access control policies [27].

We propose that by anchoring the agent’s execution in these pillars, we create a system where the orchestrator’s behavior is irrelevant to the security outcome. The structural constraints of I-EIB ensure that no execution occurs unless the Enclave’s internal verifier confirms adherence to the user-signed intent and the predefined chain layout [29].

To make the evaluation section easier to interpret, it should be noted here that the simulated omission, replay-style misuse, and parameter-tampering attacks later reported in Section 5 are derived directly from this threat model. Omission attacks correspond to orchestrator-level skipping of required provenance stages, while parameter tampering represents host-side modification of chain inputs, policy-constrained values, or downstream tool arguments under hostile-host control.

3. FORMAL SECURITY MODEL AND PROOF

The efficacy of I-EIB rests on its ability to transform abstract security goals into enforceable cryptographic invariants [7]. We move beyond heuristic-based defenses by

formalizing the interaction between the user’s intent and the agent’s execution path [29]. This section defines the mathematical underpinnings of our “complete mediation” model and provides a proof sketch demonstrating why unauthorized executions remain impossible under our defined threat model [29].

3.1 Formal specification of the chain layout

To prevent omission attacks, where a compromised orchestrator might bypass intermediate safety filters, we define a Chain Layout (L) [10]. We formalize L as a 4-tuple that dictates the permissible structure and participants of an agentic workflow [11]. The components of this layout are presented in Table 2 [12].

Table 2. Formal definitions of chain layout elements (L)

Element	Symbol	Description	Formal Constraint
Steps	S	The ordered sequence of processing nodes	$S = \{s_1, s_2, \dots, s_n\}$
Required	R	Mandatory security-critical steps	$R \subseteq S$
Authorized workers	W	Whitelisted Trusted Execution Environment (TEE) measurements (M) for each step	$\forall s_i \in S, \text{"worker"}(s_i) \in W$
Transitions	T_{in}	JavaScript Object Notation (JSON) schema-based input constraints	"Validate" ($input_i, Schema_i$) $\rightarrow$ "True"

3.2 Core security invariants

The security of I-EIB is anchored in three fundamental invariants [18]. These properties ensure that the system boundary remains closed even when the host environment is hostile [19].

- (1) Invariant 1 (Capability Sealing): An execution capability C is retrievable if and only if the environment measurement M satisfies the KMS policy Π_{KMS} [29]. This ensures that long-lived secrets are physically bound to the TEE and are inaccessible to the host [21].
- (2) Invariant 2 (Chain Completeness): An execution E is authorized if and only if the presented provenance chain L contains all mandatory steps $s \in R$ and each step is signed by an authorized worker $w \in W$ [28].
- (3) Invariant 3 (Atomic Verification): The release of C and the generation of the Identity-Execution Token (IET) occur atomically within the TEE if and only if $Verify(L, P, Layout)$ returns Success [29].

3.3 Theorem of Intent-Execution Integrity

Based on these invariants, we propose the following theorem regarding the system's resilience against orchestrator subversion [24].

Theorem 1 [25]: In an I-EIB-compliant tool environment, where all execution capabilities C are governed by the TEE or an attestation-aware issuer, an execution E violating user policy P or chain layout L cannot occur within the defined system boundary [27].

Lemma 1 (Capability Non-Release): A privileged capability cannot be released unless enclave attestation satisfies the applicable sealing policy.

Lemma 2 (Gateway Non-Bypassability): A downstream tool invocation cannot succeed unless a valid IET is accepted by the Tool Wrapper.

To improve readability, the principal symbols used throughout the formal model are consolidated here. In particular, S denotes ordered workflow steps, R denotes required security-critical steps, W denotes authorized workers or enclave measurements, L denotes the provenance chain or chain layout, C denotes an execution capability, M denotes the enclave measurement, P denotes the user-signed DSL policy, and T denotes schema-based transition constraints.

We model the processing sequence as a linear chain [15]. Each link in the chain must point to the preceding link’s hash, creating an immutable sequence [16]. By defining R as a subset of S that must be present in every valid provenance chain, we ensure that an orchestrator cannot skip a Safety_Check or a Planner_Validation step without breaking the integrity of the chain [27].

Lemma 3 (Chain Incompleteness Rejection): Any provenance chain that omits a required step, includes an unauthorized worker, or violates the user-signed DSL policy fails enclave verification.

Proof Sketch: We prove this by contradiction [28]. Suppose an execution occurs that violates policy (e.g., an unauthorized bank transfer) [29].

- (1) Access Failure: For E to execute, it requires capability C [30]. By Invariant 1, C is only available inside the TEE [1].
- (2) Bypass Failure: For E to trigger at the tool endpoint, it must present a valid IET [2]. Since we utilize a Tool Wrapper that mandates IET verification, the host cannot bypass the TEE [3].
- (3) Verification Failure: Within the TEE, the verifier checks the provenance chain L [28]. If the orchestrator modified the intent to initiate E , the chain would fail the match against P [5]. If the orchestrator omitted a security filter to hide the malicious intent, the chain would fail the completeness check against R (Invariant 2) [6].
- (4) Conclusion: Since the TEE only releases and signs the IET upon a successful Verify call (Invariant 3), no valid capability C or token can be generated for E [7]. Thus, E cannot occur [8].

This logic is visually summarized in the cryptographic sequence shown in Figure 2 [9]. By coupling the hardware-backed isolation of credentials with the strict schema-based validation of the processing steps, I-EIB creates an environment where “complete mediation” is a structural property of the architecture rather than a configuration option [10].

This argument remains construction-based rather than a fully mechanized proof in ProVerif, Tamarin, or Burrows-Abadi-Needham (BAN) logic. The theorem therefore establishes architectural and cryptographic resistance under the stated model, not universal formal completeness.

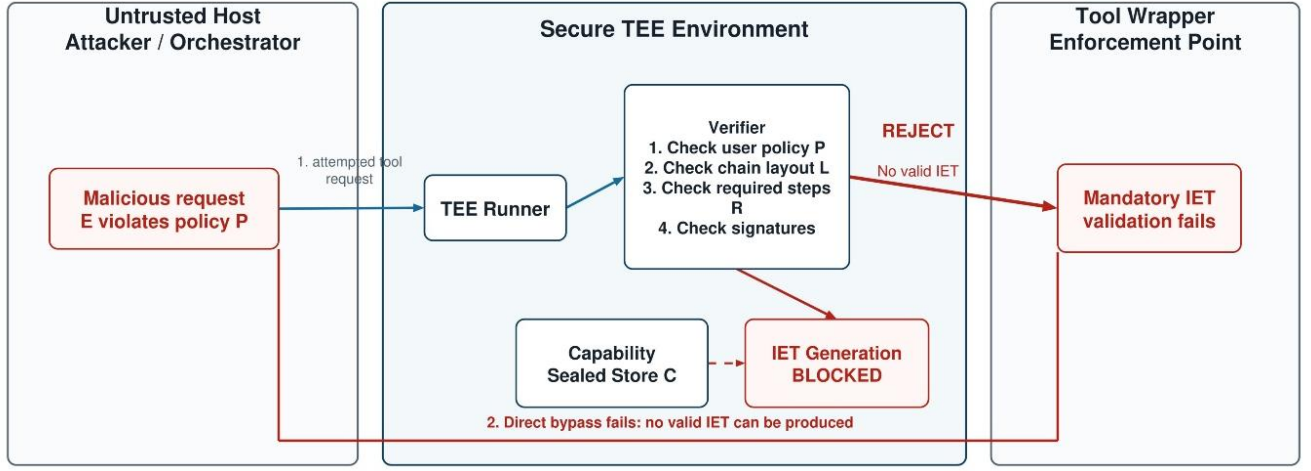


Figure 2. The visually summarized cryptographic sequence

4. ARCHITECTURE IMPLEMENTATION AND OPERATION

We transitioned the theoretical framework of I-EIB into a functional prototype using the Rust programming language, chosen for its memory safety guarantees and low runtime overhead [14]. Our implementation targets AWS Nitro Enclaves, utilizing the AF_VSOCK protocol for secure communication between the untrusted host and the isolated enclave [29]. By minimizing the Trusted Computing Base (TCB) through the use of the NFA-based regex crate and the serde library for canonical JSON handling, we reduced the attack surface of the internal policy verifier [2]. To better reflect the engineering-critical aspects of the implementation, the runtime discussion here focuses less on language choice and more on the actual execution path and dominant computational stages.

4.1 Cryptographic payload and link binding

The integrity of the provenance chain relies on the rigorous definition of the cryptographic payload generated at each processing step [27]. Every worker w_i participating in the agentic workflow must produce a signed link L_i [18]. The structure of this signature payload, PL_i , is designed to bind the step's identity to its specific hardware execution context and the preceding state of the chain [29]. Recent studies on prompt-injection formalization, tool-integrated agent benchmarks, behavioral risk assessment, and task-alignment defenses further support the need for explicit payload fields and validation logic [30-37]. We define PL_i as follows [20]:

$$PL_i = \left\langle StepID_i, H(Input_i), H(Output_i), PrevLinkHash, M_i, P_{id}, Timestamp \right\rangle \quad (1)$$

In this construction, M_i represents the hardware measurement (PCR values) of the TEE, while $PrevLinkHash$ serves as the pointer to the entirety of the previous link L_{i-1} , rather than just the prior output [22]. This deep binding ensures that any attempt to reorder steps or substitute a worker with an unauthorized measurement results in an immediate hash mismatch [23]. The specific fields of the link metadata are presented in Table 3.

Table 3. Specification of the Provenance Link Payload (PL_i)

Field	Description	Security Function
$StepID$	Identifier of the current workflow stage	Ensures adherence to the linear sequence S
$H(Input)$	SHA-384 hash of the data received	Prevents "mid-flight" data manipulation
$H(Output)$	SHA-384 hash of the produced result	Fixes the state for the subsequent step
$PrevLinkHash$	Cryptographic pointer to L_{i-1}	Enforces the immutable order of the chain
M_i	Trusted Execution Environment (TEE) measurement ($PCR0$)	Verifies the hardware-backed identity of the worker
P_{id}	Current policy ID	Prevents mixing links from different user sessions

4.2 The Verification Protocol

Crucially, the TEE runner does not release execution capabilities until the Verify algorithm successfully validates the presented provenance chain against the user-signed policy and the system layout L [28]. We implemented a strict verification logic that prioritizes schema validation for each transition [28]. Unlike heuristic approaches, our verifier mandates that the input of each link L_i conforms to the JSON Schema defined in the transition rules T_i in [29]. This prevents semantic bypasses where the orchestrator might provide technically valid but logically malicious inputs to a tool [30]. The critical execution path consists of five stages: (1) attestation generation, (2) enclave-side policy and schema verification, (3) provenance-link signature verification, (4) attestation-gated capability release through KMS or attestation-aware token issuance, and (5) wrapper-mediated downstream invocation. The dominant cost lies in attestation-backed KMS decryption and provenance verification rather than in ordinary application processing.

The core verification logic, as executed within the enclave, is presented below:

```
Python
def verify_provenance(chain_L, user_policy_P,
```

```

system_layout_L:
# 1. Validation of Mandatory Steps (Omission Detection)
executed_steps = [link.step_id for link in chain_L]
if not
system_layout_L.required_steps.issubset(set(executed_steps)
):
return REJECT("Mandatory security steps missing from
chain")

# 2. Sequential Integrity and Schema Verification
prev_link_hash = SEED_HASH

for link in chain_L:
# (a) Identity and Measurement Verification (W)
if not is_authorized(link.worker_id, link.measurement,
system_layout_L.W):
return REJECT("Worker identity or measurement
unauthorized")

# (b) Hash Chain Consistency Check
if link.prev_link_hash != prev_link_hash:
return REJECT("Provenance chain discontinuity detected")

# (c) Input Schema Enforcement (Tin)
# We verify that the input for this step adheres to the formal
layout
if not validate_schema(link.input,
system_layout_L.T_in[link.step_id]):

```

```

return REJECT(f"Schema violation at step: {link.step_id}")

# (d) Cryptographic Signature Validation
if not crypt_verify(link.sig, link.payload, link.pubkey):
return REJECT("Cryptographic signature invalid")
# Update link pointer for the next iteration
prev_link_hash = hash(link)

```

```

# 3. Final Policy and User Intent Alignment
return match_dsl_policy(user_policy_P,
chain_L.last().final_request)

```

The connection between the DSL policy and the chain layout can be stated more explicitly as follows: the DSL policy constrains what final action is allowed, while the chain layout constrains how that action must be reached. During verification, the enclave first checks chain completeness and worker legitimacy, then validates schema-constrained transitions, and only afterward confirms that the final request remains within the user-signed DSL policy.

4.3 Operational resilience and lifecycle management

We designed the I-EIB operational model to handle the realities of distributed cloud environments, specifically addressing replay attacks and identity rotation [4]. The communication architecture between the host and the enclave is illustrated in Figure 3.

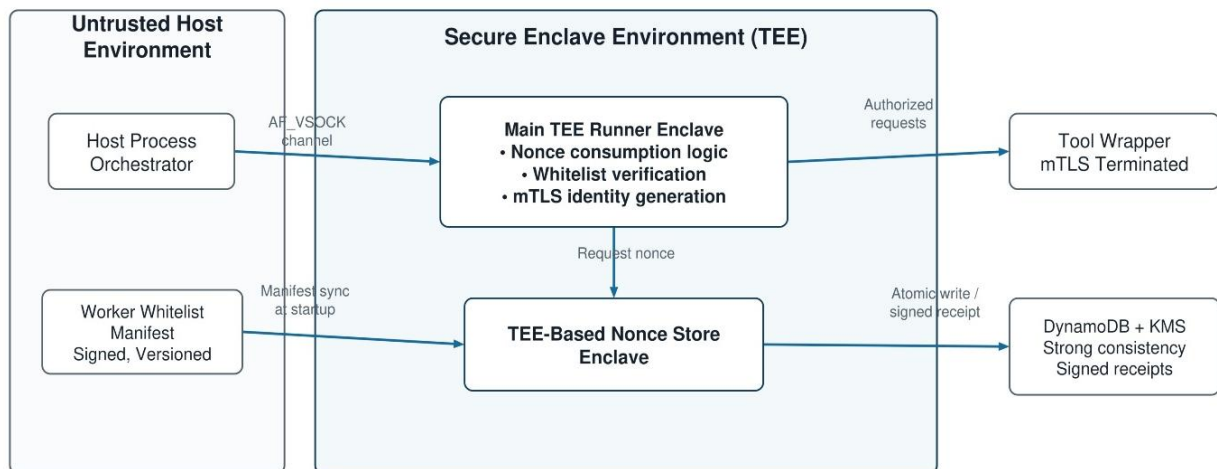


Figure 3. The communication architecture between the host and the enclave

To prevent the reuse of previously valid provenance chains, we implemented an anti-replay mechanism using a TEE-based Nonce Store [28]. By hosting the store within its own enclave, we ensure that the signing keys for nonce receipts are protected from host-level exfiltration [1]. We use DynamoDB with strong consistency for the underlying storage, employing conditional writes (attribute_not_exists) to guarantee the atomic consumption of nonces [4]. When a TEE runner consumes a nonce, it receives a receipt signed by the Nonce Store’s KMS-backed asymmetric key [6].

Furthermore, we manage the lifecycle of the worker whitelist (W) through a versioned, signed manifest [7]. When a new worker image is deployed, its measurement is added to the authorized list, while compromised or deprecated versions are moved to a revocation list [8]. This manifest is synchronized with the enclave upon startup, ensuring that the is authorized check in the verification algorithm remains

current [1]. We also manage mTLS identities dynamically; client certificates are generated within the enclave, and the associated private keys never touch the host’s persistent storage, thereby maintaining a high-fidelity trust chain from the TEE to the Tool Wrapper [1].

To make the operational meaning of the architecture more concrete, we also consider a simple workflow case. A user authorizes an agent to review a healthcare billing document and submit a reimbursement request only after a safety-validation stage and a planner-validation stage. In the benign case, the chain includes retrieval, normalization, safety check, planner validation, and wrapper-mediated submission. In a hostile-host case, the orchestrator attempts to omit the safety step and directly generate the downstream submission request. The enclave-side verifier rejects the chain because a required step in R is missing, and no capability or IET is released. This illustrates that I-EIB constrains not only what action may be

taken, but also how that action must be reached.

5. PERFORMANCE AND SECURITY EVALUATION

To assess the practical viability of the I-EIB architecture, we conducted an empirical evaluation focused on two primary metrics: computational overhead and defensive efficacy [14]. We deployed our prototype on an AWS m5.xlarge instance (4 vCPUs, 16GB RAM) utilizing AWS Nitro Enclaves CLI 1.3.1. All measurements represent the aggregate of 1,000 independent runs to ensure statistical significance. We utilized a standard agentic workload consisting of a five-step provenance chain and a 12KB JSON-based DSL policy. To make the evaluation more interpretable from a computational and engineering perspective, we also compare the system against a host-mediated baseline that performs policy checking without enclave isolation or attestation-gated capability release.

5.1 Latency analysis and scaling

A critical concern in TEE-based security is the latency introduced by hardware attestation and cryptographic verification. We found that the I-EIB framework adds a mean end-to-end latency of 14.2ms to the agent's execution cycle. While any security layer introduces delay, this overhead remains negligible—typically representing less than 2% of the total inference time for modern LLMs like GPT-4 or Claude 3.5.

Compared with the host-mediated baseline, the mean additional overhead attributable specifically to enclave isolation and attestation-gated capability release was 14.2 ms.

This comparison indicates that the main cost of stronger complete mediation arises from enclave-backed secret release, provenance verification, and VSOCK context switching rather than from ordinary application execution.

The breakdown of this latency, as presented in Table 4, reveals that the attestation-backed KMS decryption is the most significant contributor.

Notably, the provenance verification time scales linearly with the number of steps in the agentic workflow. Our data indicates an incremental cost of approximately 0.9ms per additional link. This suggests that even for complex, long-running agents involving twenty or more steps, the total security-induced delay would likely remain below 30ms, maintaining a high level of responsiveness for real-time applications. This trend is illustrated in Figure 4.

From a computational viewpoint, provenance verification scales as $O(|L|)$ in the number of links in the chain, with per-link cost dominated by schema validation, hash-chain checking, and signature verification. The principal fixed-cost component remains attestation-gated capability release.

Table 4. Latency breakdown of Intent-Execution Integrity Binding (I-EIB) security operations (N = 1,000)

Metric	Mean (ms)	P95 (ms)	P99 (ms)
Key Management Service (KMS) decrypt (attestation-gated)	7.9	9.2	11.5
Provenance chain verification	4.5	5.3	6.8
Virtual Socket (VSOCK) context switching	1.8	2.5	3.2
Total security overhead	14.2	18.5	23.4

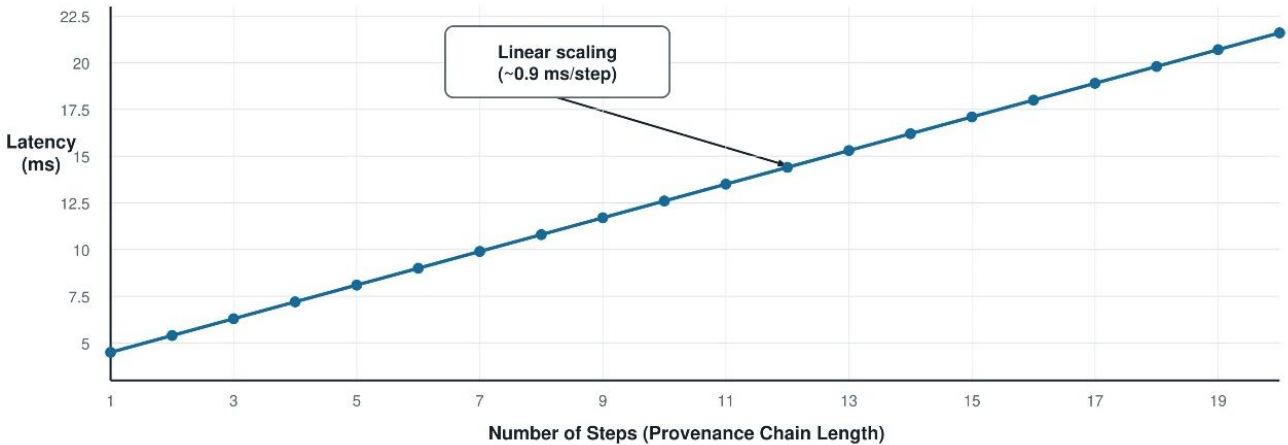


Figure 4. Latency scaling vs. provenance chain length

5.2 Security effectiveness and taxonomy of attacks

We evaluated I-EIB against a corpus of 500 malicious scenarios designed to simulate sophisticated orchestrator compromises. These were categorized into 200 “Omission Attacks”—where an attacker attempts to skip mandatory security filters—and 300 “Parameter Tamper” attempts.

The attack corpus was derived directly from the hostile-host adversary model defined in Section 2.3. The omission attacks were generated by removing one or more mandatory steps from otherwise valid workflows, while the parameter-tampering cases altered policy-constrained fields, encoded

values, or downstream tool arguments at different points in the chain. The reported 100% detection rate should therefore be understood as applying to this bounded scenario set rather than to all imaginable attack variants.

Our results, summarized in Figure 5, show that I-EIB achieved a 100% detection rate for omission attacks. This perfect score is a direct result of the mandatory Required_Steps constraint in our Chain Layout (L). Because the TEE runner refuses to release capabilities unless every mandatory link is cryptographically present and valid, an attacker cannot bypass a safety check without severing the chain.

In the parameter tampering category, I-EIB blocked 98.4% of attempts (295 out of 300). We performed a failure analysis on the remaining 1.6% (5 cases) and identified two primary causes for these misses. Three cases involved “Schema Laxity,” where the DSL’s regex constraints were defined too broadly, allowing subtle Unicode encoding variants to pass through. The other two cases stemmed from normalization edge cases in tool outputs that bypassed the policy verifier’s current canonicalization rules.

More specifically, the first failure mode occurred when regular-expression constraints were permissive enough to admit semantically altered but syntactically valid values, including Unicode variants. The second occurred when legacy tool outputs were canonicalized inconsistently before hashing. These results suggest that the residual risk in deployment lies less in the core enclave architecture than in schema precision and normalization discipline.

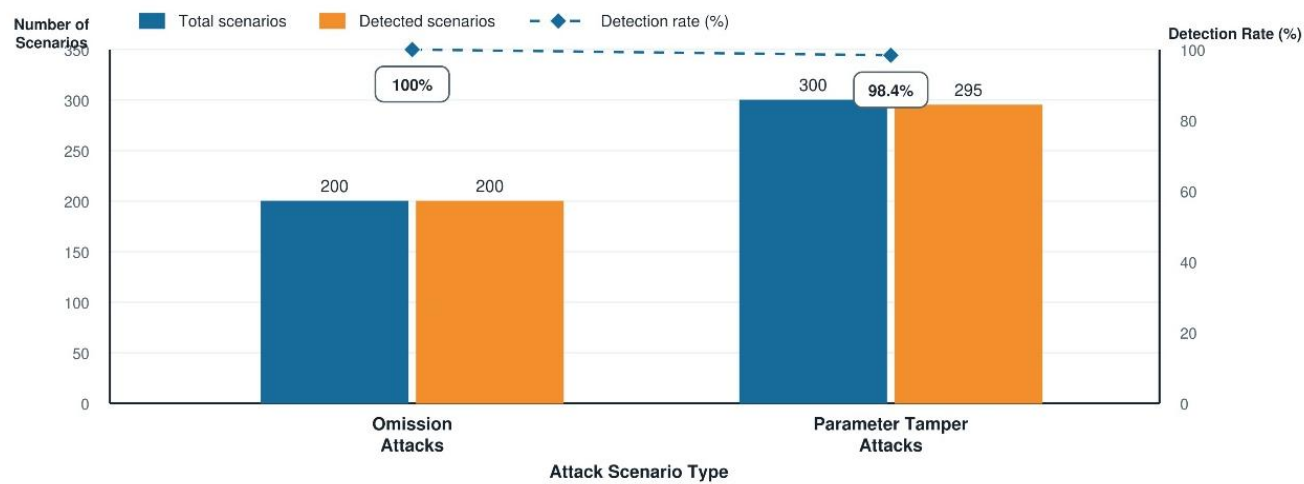


Figure 5. Attack detection rates by scenario type

5.3 False positive mitigation via normalization

Strict security enforcement often risks high False Positive Rates (FPR), particularly in LLM environments where text outputs are non-deterministic. Initially, we observed an FPR of 0.6%, primarily caused by minor encoding discrepancies (e.g., NFC vs. NFD normalization) in legacy tool outputs.

To address this, we implemented a Canonicalizer Adapter within the TEE. This adapter forces all incoming and outgoing data into a standard UTF-8 NFC format and strips non-essential whitespace before hashing. Following the introduction of this adapter, the FPR dropped to below 0.1%. This indicates that a rigorous approach to data canonicalization can effectively resolve the tension between strict integrity binding and operational availability.

For clarity, the canonicalization procedure can be summarized as follows:

Algorithm: Canonicalize(x); First, decode x as UTF-8; Second, normalize Unicode to NFC; Third, collapse non-semantic whitespace; Fourth, sort canonical JSON keys if structured; Fifth, serialize in deterministic form; Sixth, return canonicalized output for hashing and schema validation.

This shows that the reduction in false positives was achieved through a deterministic canonicalization path rather than ad hoc adjustment.

We suggest that I-EIB offers a highly practical tradeoff: it provides near-absolute protection against orchestrator subversion for a performance cost that is almost imperceptible to the end-user.

6. CONCLUSION

The intent-execution gap in autonomous LLM agents represents a critical failure point. We addressed this

vulnerability through I-EIB, a hardware-anchored architecture that binds user intent directly to execution. By coupling KMS-sealed capabilities with formalized provenance chains, we ensure that even a total orchestrator compromise cannot result in unauthorized tool execution. Our empirical results on AWS Nitro Enclaves show that this security layer is remarkably efficient, adding just 14.2 ms of mean latency. This negligible performance cost, paired with a 100% detection rate in omission attack simulations, suggests that I-EIB successfully enforces the principle of complete mediation within its defined boundaries.

Several limitations must also be acknowledged. I-EIB depends on the availability and correct configuration of TEE infrastructure, attestation-aware secret release, accurately specified DSL policies, and correctly deployed Tool Wrappers. In heterogeneous or multi-cloud settings, additional engineering work may be required to preserve the same closed-boundary property across provider-specific attestation and credential systems.

As agents assume higher-stakes roles in finance and healthcare, I-EIB offers the verifiable foundation necessary for maintaining intent-integrity. Future work should therefore extend this architecture through broader numerical simulation, richer workflow-based evaluations, stronger policy-authoring support, mechanized formal verification, and cross-cloud deployment support.

ACKNOWLEDGMENT

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-RS-2023-00237287, 2026S1A5A01004224).

DATA AVAILABILITY

The data used to support the findings of this study are available from the corresponding author upon request.

REFERENCES

- [1] Fei, S., Yan, Z., Ding, W., Xie, H. (2021). Security vulnerabilities of SGX and countermeasures: A survey. *ACM Computing Surveys*, 54(6): 1-36. <https://doi.org/10.1145/3456631>
- [2] Pinto, S., Santos, N. (2019). Demystifying Arm TrustZone: A comprehensive survey. *ACM Computing Surveys*, 51(6): 1-36. <https://doi.org/10.1145/3291047>
- [3] Denning, D.E. (1976). A lattice model of secure information flow. *Communications of the ACM*, 19(5): 236-243. <https://doi.org/10.1145/360051.360056>
- [4] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7): 558-565. <https://doi.org/10.1145/359545.359563>
- [5] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Available online: <https://bitcoin.org/bitcoin.pdf>.
- [6] Shacham, H., Waters, B. (2013). Compact proofs of retrievability. *Journal of Cryptology*, 26(3): 442-483. <https://doi.org/10.1007/s00145-012-9129-2>
- [7] Ateniese, G., Burns, R., Curtmola, R., et al. (2007). Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA, pp. 598-609. <https://doi.org/10.1145/1315245.1315318>
- [8] Boneh, D., Franklin, M. (2003). Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3): 586-615. <https://doi.org/10.1137/S0097539701398521>
- [9] Baumann, A., Peinado, M., Hunt, G. (2015). Shielding applications from an untrusted cloud with Haven. *ACM Transactions on Computer Systems*, 33(3): 1-26. <https://doi.org/10.1145/2799647>
- [10] Merkle, R.C. (1988). A digital signature based on a conventional encryption function. In: *Advances in Cryptology—CRYPTO 1987*. Berlin, Heidelberg: Springer, 369-378. https://doi.org/10.1007/3-540-48184-2_32
- [11] Cheney, J., Chiticariu, L., Tan, W.C. (2009). Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4): 379-474. <https://doi.org/10.1561/19000000006>
- [12] Moreau, L., Clifford, B., Freire, J., et al. (2011). The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27(6): 743-756. <https://doi.org/10.1016/j.future.2010.07.005>
- [13] Joshi, A.P., Han, M., Wang, Y. (2018). A survey on security and privacy issues of blockchain technology. *Mathematical Foundations of Computing*, 1(2): 121-147. <https://doi.org/10.3934/mfc.2018007>
- [14] Schneier, B., Kelsey, J. (1999). Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2): 159-176. <https://doi.org/10.1145/317087.317089>
- [15] Ali, M., Nelson, J., Shea, R., Freedman, M. (2016). Blockstack: A global naming and storage system secured by blockchains. In *Proceedings of the 2016 USENIX Annual Technical Conference*, Denver, USA, pp. 181-194.
- [16] Pasquier, T., Singh, J., Bacon, J., Evers, D. (2018). Data provenance to audit compliance with privacy policy in the Internet of Things. *Personal and Ubiquitous Computing*, 22(2): 333-344. <https://doi.org/10.1007/s00779-017-1067-4>
- [17] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, Bethesda, MD, USA, pp. 169-178. <https://doi.org/10.1145/1536414.1536440>
- [18] Papernot, N., McDaniel, P., Goodfellow, I. (2016). Transferability in machine learning: From phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*. <https://doi.org/10.48550/arXiv.1605.07277>
- [19] Haber, S., Stornetta, W.S. (1991). How to time-stamp a digital document. In *Advances in Cryptology—CRYPTO 1990*, Berlin, Heidelberg: Springer, pp. 437-455. https://doi.org/10.1007/3-540-38424-3_32
- [20] Schuster, F., Costa, M., Fournet, C., et al. (2015). VC3: Trustworthy data analytics in the cloud. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 38-54. <https://doi.org/10.1109/SP.2015.10>
- [21] Ménétrey, J., Göttel, C., Khurshid, A., et al. (2022). Attestation mechanisms for trusted execution environments demystified. In *Distributed Applications and Interoperable Systems*, Cham: Springer, pp. 95-113. https://doi.org/10.1007/978-3-031-16092-9_7
- [22] Castro, M., Liskov, B. (2002). Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4): 398-461. <https://doi.org/10.1145/571637.571640>
- [23] Reiter, M.K. (1996). A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1): 31-42. <https://doi.org/10.1109/32.481515>
- [24] Douceur, J.R. (2002). The Sybil attack. In *Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, MA, USA: Springer, pp. 251-260. https://doi.org/10.1007/3-540-45748-8_24
- [25] Shamir, A. (1984). Identity-based cryptosystems and signature schemes. In *Advances in Cryptology—CRYPTO 1984*, Berlin, Heidelberg: Springer, pp. 47-53. https://doi.org/10.1007/3-540-39568-7_5
- [26] Juels, A., Kaliski, B.S.Jr. (2007). PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA, pp. 584-597. <https://doi.org/10.1145/1315245.1315317>
- [27] Shi, E., Chan, H.T.H., Rieffel, E., Chow, R., Song, D. (2011). Privacy-preserving aggregation of time-series data. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA.
- [28] Abadi, M., Burrows, M., Lampson, B., Plotkin, G. (1993). A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4): 706-734. <https://doi.org/10.1145/155183.155225>
- [29] Rivest, R., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):

- 120-126. <https://doi.org/10.1145/359340.359342>
- [30] Liu, Y., Jia, Y., Geng, R., Jia, J., Gong, N.Z. (2024). Formalizing and benchmarking prompt injection attacks and defenses. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, USA, pp. 1831-1847.
- [31] Zhan, Q., Liang, Z., Ying, Z., Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand, pp. 10471-10506. <https://doi.org/10.18653/v1/2024.findings-acl.624>
- [32] Debenedetti, E., Zhang, J., Balunovic, M., Beurer-Kellner, L., Fischer, M., Tramèr, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems 37*, Vancouver, Canada, pp. 82895-82920. <https://doi.org/10.52202/079017-2636>
- [33] Yuan, T., He, Z., Dong, L., et al. (2024). R-Judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Miami, Florida, USA, pp. 1467-1490. <https://doi.org/10.18653/v1/2024.findings-emnlp.79>
- [34] Jia, F., Wu, T., Qin, X., Squicciarini, A. (2025). The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria, pp. 29680-29697. <https://doi.org/10.18653/v1/2025.acl-long.1435>
- [35] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *AISeC '23: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pp. 79-90. <https://doi.org/10.1145/3605764.3623985>
- [36] Ruan, Y., Dong, H., Wang, A., et al. (2024). Identifying the risks of LM agents with an LM-emulated sandbox. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*.
- [37] Liu, X., Yu, H., Zhang, H., et al. (2024). AgentBench: Evaluating LLMs as agents. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*.