

A Defense-in-Depth Architecture for Protecting AI Agent Intellectual Property in Multi-Tenant Cloud Environments

David Petrocelli¹, Juan Manuel Fernández^{1,2}[0000–0001–9291–3066]

¹ Universidad Nacional de Luján (UNLu), Buenos Aires, Argentina

² Buenos Aires Institute of Technology (ITBA), Argentina
dpetrocelli@unlu.edu.ar, jmfernandez@{unlu,itba}.edu.ar

Abstract. Large language models (LLMs) have shifted a substantial portion of system logic from executable code to natural-language specifications, particularly through the use of system prompts that encode behavior, domain expertise, and safety constraints. In multi-tenant cloud deployments, this paradigm introduces a fundamental tension: AI agents must often operate within customer-controlled environments due to data residency requirements, while the prompts that define their functionality must remain confidential as proprietary intellectual property.

In this work, we address this problem from a cross-domain trust perspective and propose a defense-in-depth architecture for protecting prompt-based intellectual property in such settings. The approach composes four software-based layers—cryptographic prompt delivery, OWASP-aligned input validation, output filtering with steganographic watermarking, and cross-account monitoring—into a unified trust chain that achieves practical prompt confidentiality, output integrity, and forensic traceability without relying on Trusted Execution Environments or specialized hardware.

We validate the proposed architecture through controlled penetration testing and a live two-account deployment. The results show that the system achieves 90.5% attack detection with no false positives on legitimate queries, while introducing a latency overhead of 68.3 ms at P99. The architecture incurs negligible infrastructure overhead, demonstrating that the approach is both effective and practically deployable.

These findings suggest that composing multiple complementary defenses across trust boundaries provides a practical alternative to hardware-based isolation, enabling secure and scalable AI agent architectures in real-world multi-tenant environments.

Keywords: AI security · prompt protection · defense-in-depth · multi-tenant cloud · watermarking · cryptographic authentication

1 Introduction

In recent years, large language models (LLMs) have introduced a fundamental shift in how software systems encode and operationalize knowledge [7,6]. In traditional

systems, intellectual value is primarily embedded in executable code, where system behavior is explicitly implemented. In contrast, AI agents increasingly rely on *system prompts*—natural-language specifications that define behavior, domain expertise, and safety constraints—as a primary control interface, effectively externalizing system logic into interpretable representations [33,27,31]. This shift relocates a substantial portion of system logic from compiled artifacts to natural-language specifications, raising new challenges for protecting intellectual property, whose practical relevance is amplified by the rapid growth of the *AI agent as a service* market [20].

This transformation becomes particularly problematic in multi-tenant cloud environments, where deployment and data ownership are separated across administrative boundaries. In such settings, AI agents are often required to operate within the customer’s cloud account to satisfy data residency constraints and regulatory requirements such as the General Data Protection Regulation (GDPR), the Health Insurance Portability and Accountability Act (HIPAA), and Service Organization Control 2 (SOC 2) [13]. At the same time, system prompts must be protected as proprietary knowledge, since they encode the differentiated capabilities that justify the service’s commercial value. These requirements are fundamentally at odds: executing an agent within a customer-controlled environment introduces an exposure surface in which prompt configurations become accessible to privileged actors, undermining their confidentiality [3,29].

This tension is not merely theoretical. Commercial AI agent platforms—such as Amazon Bedrock Agents, Azure AI Agent Service, and Vertex AI Agents—store agent configurations within account-level resources, often accessible to administrative principals. Prompt extraction has been shown to be practical even without privileged access [15,1], while prompt injection attacks [11,23], identified as the primary risk in the Open Worldwide Application Security Project (OWASP) LLM Top-10, further expand the attack surface by enabling indirect manipulation of model behavior.

Existing approaches address this problem only partially and typically in isolation. Cryptographic mechanisms protect prompts during transmission or controlled access [32], while software-based defenses target specific attack vectors such as filtering, obfuscation, or leakage detection [34,8,9,2,17]. Hardware-based solutions rely on Trusted Execution Environments (TEEs), which provide strong isolation guarantees but require specialized hardware, incur measurable overhead [10], and lack support for forensic attribution [14,5]. More broadly, recent surveys on agentic security [29] highlight the breadth of the threat landscape, yet identify no deployed architecture that systematically addresses prompt protection across distinct administrative domains.

Prompt protection in multi-tenant LLM systems can thus be understood as a cross-domain trust problem rather than an isolated security mechanism design task [26,25,29]. Addressing this problem requires reasoning about how trust is distributed across vendor and customer environments, how adversaries exploit these boundaries, and how multiple defensive mechanisms can be composed to preserve security guarantees even under partial compromise.

We propose a defense-in-depth architecture that composes standard cloud services into a four-layer trust chain, integrating cryptographic prompt delivery, OWASP-aligned input validation, output filtering with steganographic watermarking, and cross-account monitoring into a unified design. This approach enables prompt confidentiality, output integrity, and forensic traceability while preserving deployment flexibility in multi-tenant environments. By avoiding reliance on specialized hardware and composing defenses across independent layers, the architecture remains portable across cloud providers and resilient to partial compromise.

The effectiveness of the proposed approach is evaluated along three complementary dimensions: (i) security effectiveness, in terms of resistance to prompt extraction and adversarial manipulation; (ii) performance overhead, measured as the latency introduced by the defense layers; and (iii) practical deployability, including portability across cloud providers and feasibility under realistic operational constraints.

We formalize the problem as a cross-boundary trust challenge, define a corresponding threat model, and validate the architecture through controlled penetration testing and a real-world two-account deployment.

The remainder of this paper is organized as follows. Section 2 reviews related work, Section 3 introduces the threat model, Sections 4 and 5 present the proposed system, Section 6 provides the experimental evaluation, and Section 7 discusses limitations and concludes the paper.

2 Related Work

Recent work has highlighted the vulnerability of large language model (LLM) applications to prompt extraction and manipulation attacks. PLeak [15] formalizes prompt extraction as an adversarial task, achieving near-complete reconstruction of system prompts through carefully crafted queries. Multi-turn leakage attacks further exploit model sycophancy to progressively recover hidden instructions, reaching success rates as high as 86.2% [1]. Complementary analyses of prompt injection attacks classify a wide range of direct, indirect, and composed attack vectors, demonstrating that prompt confidentiality alone is insufficient to guarantee secure behavior [11].

In response to these threats, a growing body of work has proposed software-based defenses that target specific stages of the interaction pipeline. Approaches such as ProxyPrompt [34] introduce proxy layers to mitigate prompt extraction, while SysVec [8] encodes prompts into non-invertible vector representations. Other methods focus on filtering and validation mechanisms: PromptGuard [2] applies structured filtering to mitigate injection attacks, Encrypted Prompt [9] restricts prompt access through permission-based gating, and PromptKeeper [17] detects leakage via hypothesis testing. Attention-based techniques, such as Attention Tracker [16], analyze model internals to identify anomalous patterns associated with adversarial inputs. While effective within their scope, these approaches

typically operate within a single trust boundary and address individual attack vectors in isolation.

Beyond point defenses, recent work has explored multi-layered strategies for improving robustness against adversarial behavior. TRYLOCK [30] composes multiple jailbreak defenses to reduce attack success rates, while Countermind [28] augments inference pipelines with audit and monitoring capabilities. Similarly, serverless security analyses highlight the expanded attack surface introduced by cloud-native machine learning deployments [24]. These approaches emphasize the importance of layered protection, but remain primarily focused on inference-time robustness rather than cross-domain deployment scenarios.

At the infrastructure level, hardware-based approaches aim to provide confidentiality guarantees through Trusted Execution Environments (TEEs). Systems such as Petridish [14] and Omega [5] execute LLM inference within secure enclaves, preventing direct access to model inputs and prompts. Empirical evaluations show that such approaches can achieve modest overheads in controlled settings [10]. However, they require specialized hardware, introduce deployment constraints, and do not inherently provide mechanisms for forensic attribution or cross-system monitoring.

Finally, watermarking techniques address the complementary problem of output provenance and attribution. Early work introduced token-level watermarking schemes for LLM outputs [18], while subsequent approaches such as Waterfall [19] and SynthID [12] demonstrate scalable, production-ready solutions for content traceability. Alternative approaches based on zero-width character (ZWC) steganography enable covert embedding of signals into generated text [4], although recent work has shown that such mechanisms may be vulnerable to steganographic collusion attacks [21]. These methods focus on output integrity and attribution, rather than protecting the confidentiality of system prompts.

Taken together, these lines of work illustrate a fragmented landscape in which different aspects of the problem—prompt confidentiality, input validation, output filtering, and attribution—are addressed independently, often within a single trust boundary. This fragmentation becomes particularly limiting in multi-tenant settings, where the separation between deployment and ownership introduces additional constraints on how security mechanisms can be applied.

In this context, the present work approaches prompt protection as a cross-domain trust problem and investigates how multiple defensive mechanisms can be systematically composed across layers and administrative boundaries. Rather than focusing on individual attack vectors or isolated defenses, we explore an architectural perspective that integrates cryptographic protection, validation, filtering, watermarking, and monitoring into a unified framework, enabling secure deployment in realistic cloud-based environments.

3 Threat Model

To operationalize the cross-domain trust perspective introduced in the previous section, we formalize the threat model for multi-tenant LLM deployments. We

consider a setting in which deployment and ownership are separated across administrative boundaries, and analyze how different adversaries interact with the system under this constraint. The following subsections define the system model, characterize adversary capabilities, and specify the security goals and their relation to relevant attack vectors.

3.1 System Model

We define a *two-account boundary*: a *Vendor Account* (trusted, controlled by the Software-as-a-Service (SaaS) vendor, hosting prompts, keys, registry, and monitoring) and a *Customer Account* (semi-trusted, hosting the deployed agent, an Application Programming Interface (API) gateway, and event forwarding). This generalizes across providers: Amazon Web Services (AWS) account boundaries, Azure subscription/tenant boundaries, or Google Cloud Platform (GCP) project boundaries.

3.2 Adversary Classes

Three adversaries with increasing capabilities: (1) **Customer Administrator** ($\mathcal{A}_1$) with full account privileges—can inspect containers, read environment variables, intercept traffic; (2) **External Attacker** ($\mathcal{A}_2$) with access only to the public API; (3) **Malicious Insider** ($\mathcal{A}_3$) with partial access to either organization.

3.3 Security Goals and Attack Mapping

Three guarantees: *prompt confidentiality* (prompt inaccessible in customer account at rest or in transit), *output integrity* (tamper-detectable responses), and *forensic traceability* (invisible per-request fingerprint). Table 1 maps vectors to adversaries and defenses.

The table uses the following abbreviations: Hash-based Message Authentication Code (HMAC), man-in-the-middle (MITM), and zero-width character (ZWC).

Table 1. Attack vectors, adversary capabilities, and defense layer mapping.

Attack vector	Adv.	Goal	Defense
Console/API inspection	$\mathcal{A}_1$	Conf.	L1: Cross-acct isolation
Container reverse eng.	$\mathcal{A}_1$	Conf.	L1: Multi-stage build
Direct gatekeeper call	$\mathcal{A}_1$	Conf.	L1: HMAC + account binding
Request replay	$\mathcal{A}_1, \mathcal{A}_2$	Conf.	L1: Timestamp + nonce
MITM tampering	$\mathcal{A}_1$	Integ.	L1: Body hash in HMAC
Prompt injection / jailbreak	$\mathcal{A}_2$	Conf.	L2: Risk scoring
Multi-turn extraction	$\mathcal{A}_2$	Conf.	L2+L3: Score + filter
Output exfiltration	$\mathcal{A}_2, \mathcal{A}_3$	Conf.	L3: Canary + detection
Unauthorized redistribution	$\mathcal{A}_3$	Trace.	L3: ZWC fingerprint
Encoding evasion	$\mathcal{A}_2$	Conf.	L2: Encoding detectors
Typoglycemia attacks	$\mathcal{A}_2$	Conf.	L2: Normalization
Monitoring evasion	$\mathcal{A}_1$	All	L4: Vendor-side logs

Trust assumptions. (1) Vendor Identity and Access Management (IAM) is correctly configured; (2) Transport Layer Security (TLS) 1.2+ is used on all cross-account communication; (3) container runtime provides process-level memory isolation; (4) the LLM API does not persist prompts beyond session scope. Assumption (1) is weaker than TEE hardware attestation but universally available.

4 Architecture

The proposed defense-in-depth architecture composes four defense layers into a sequential trust chain, analogous to network security pipelines composed of firewalls, Intrusion Detection Systems (IDS), Web Application Firewalls (WAF), and Security Information and Event Management (SIEM) systems. Figure 1 illustrates the two-account deployment model, separating trusted vendor-side components from customer-controlled infrastructure.

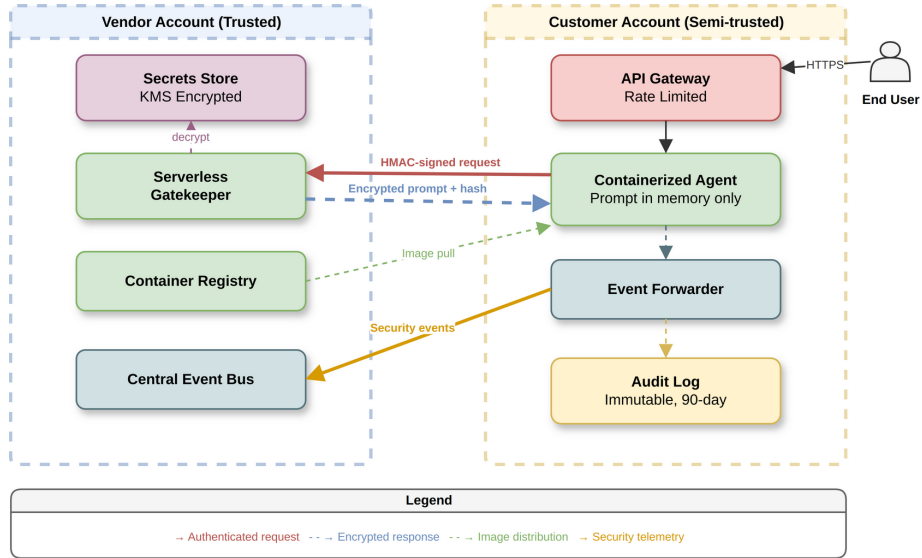


Fig. 1. Proposed two-account architecture for cross-domain prompt protection.

The *Vendor Account* hosts the trusted components of the system, including a secrets store with Key Management Service (KMS)-encrypted prompts, a serverless gatekeeper responsible for authentication and prompt delivery, a container registry, and a centralized event bus. In contrast, the *Customer Account* hosts the deployed components, including the containerized agent, an API gateway with rate limiting, and an event forwarding mechanism. The agent retrieves its prompt at runtime and maintains it only in process memory.

This separation enforces a critical invariant: the system prompt never persists in customer-controlled storage and exists only transiently during inference.

While Figure 1 describes where system components are deployed, the security of the architecture also depends on how requests are processed across these components. In particular, protecting prompt confidentiality in a cross-account setting requires a sequence of coordinated defenses applied at different stages of the interaction.

Figure 2 illustrates the defense-in-depth pipeline as four sequential layers, whose composition provides end-to-end protection across the request lifecycle.

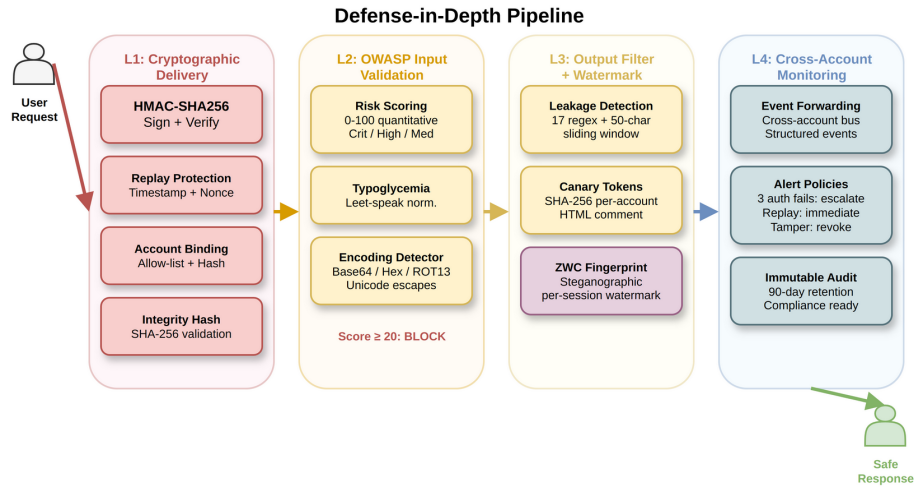


Fig. 2. Defense-in-depth pipeline composed of four sequential layers.

Each layer addresses a distinct aspect of the attack surface: Layer 1 ensures secure prompt delivery through cryptographic request validation; Layer 2 applies OWASP-aligned input validation to detect adversarial inputs; Layer 3 performs output filtering and embeds forensic fingerprints for leakage prevention and attribution; and Layer 4 centralizes cross-account monitoring to enable anomaly detection and coordinated response. The following subsections describe each layer in detail.

4.1 Layer 1: Cryptographic Prompt Delivery

This layer ensures that only authenticated agents can retrieve prompts and that requests cannot be tampered with or replayed. This is achieved through a cryptographic binding between the request payload, the originating account, and temporal information.

Each request is authenticated using an HMAC with SHA-256 (HMAC-SHA256) signature computed over a structured message:

$$m = ts \parallel nonce \parallel account_id \parallel \text{SHA256}(body) \quad (1)$$

The signature is defined as $\sigma = \text{HMAC-SHA256}(k, m)$, where k is a pre-shared secret key. This construction ensures integrity of the request body, binds the request to a specific account, and introduces temporal freshness through the timestamp and nonce.

The gatekeeper validates four properties before releasing the prompt: (1) the timestamp falls within a bounded time window to prevent replay; (2) the nonce has not been previously observed, enforced through a short-lived cache; (3) the account identifier is present in an allow-list; and (4) the request body matches the signed hash. All comparisons are performed using constant-time operations to mitigate timing attacks.

This mechanism provides three key guarantees at the application layer: request authenticity, payload integrity, and replay protection. Together, these properties ensure that prompt delivery remains secure even in the presence of adversaries with visibility into the execution environment.

4.2 Layer 2: OWASP-Aligned Input Validation

This layer mitigates prompt injection and adversarial manipulation by filtering user inputs before they reach the model. The approach is based on a quantitative risk assessment aligned with the OWASP LLM Top 10 [23] and complementary taxonomies from the National Institute of Standards and Technology (NIST) [22,11].

Each incoming message is assigned a risk score based on the presence of patterns associated with known attack strategies, including instruction override attempts, jailbreak prompts, prompt extraction queries, and structural injection markers. These patterns are grouped into severity tiers, reflecting their potential impact on system behavior.

A predefined threshold (set to 20 in our implementation) is used to block high-risk inputs while allowing benign interactions to proceed. This design balances security and usability by preventing clear attack attempts without overly restricting legitimate technical queries. In particular, medium-risk patterns (e.g., references to encoding schemes) are tolerated in isolation, while combinations of such signals may trigger blocking. The threshold follows from the severity weights (critical +30, high +20, medium +10, low +5): a score of 20 corresponds to a single high-severity pattern or two independent medium-severity signals. A sensitivity analysis over the evaluation corpus supports this operating point: coverage of the score-based detector is stable between thresholds 15 and 20 (20/42 attacks) and drops sharply at 25 (12/42), while no legitimate query scores above 5, leaving a four-fold margin below the blocking threshold; the remaining attack categories are covered by the specialized detectors rather than the generic scorer. The detector set was additionally calibrated iteratively against a preliminary live

run, raising overall detection from 59.5% to the final 90.5% (section 6) while keeping false positives at zero.

To extend coverage beyond explicit patterns, the system incorporates additional detectors targeting obfuscation strategies. These include a typoglycemia detector, which identifies deliberate misspellings of critical keywords, and an encoding detector, which flags potentially malicious payloads hidden through transformations such as base64, hexadecimal encoding, or Unicode escapes.

Together, these mechanisms provide a first line of defense against adversarial inputs, reducing the likelihood that malicious instructions reach the model while maintaining flexibility for legitimate use cases.

4.3 Layer 3: Output Filtering and Watermarking

This layer prevents unintended disclosure of sensitive information while enabling forensic attribution of generated content. It operates on every model response through three complementary mechanisms: leakage detection, active signaling, and covert fingerprinting.

First, responses are analyzed for potential prompt leakage using pattern-based detection and sliding-window similarity checks against the original prompt. Outputs exhibiting high overlap are blocked, reducing the risk of inadvertent disclosure through model generation.

Second, the system incorporates canary tokens embedded within the prompt. These tokens remain invisible during normal operation but trigger alerts if reproduced in the output, providing a direct signal of leakage or model misbehavior.

Third, each response is augmented with a steganographic fingerprint based on zero-width Unicode characters (ZWC) [4,19]. The fingerprint encodes a hash derived from the account, session, and request identifiers and is distributed across the generated text to remain imperceptible while resisting naive removal strategies. For completeness, Algorithm 1 shows a reference implementation of the watermarking mechanism.

Algorithm 1 ZWC-based watermark embedding

```

1: function WATERMARK(text, account, session, request)
2:   payload  $\leftarrow$  account||session||request
3:   h  $\leftarrow$  SHA256(payload)
4:   w  $\leftarrow$  EncodeZWC(h)
5:   if |text| > 4 tokens then
6:     pos  $\leftarrow$  random index in text
7:     insert w at position pos in text
8:     return modified text
9:   else
10:    return w||text
11:  end if
12: end function

```

Together, these mechanisms provide two complementary guarantees: prevention of prompt leakage at generation time and post hoc attribution in the event of content exfiltration. This dual capability strengthens the overall security posture by combining proactive filtering with reactive traceability.

4.4 Layer 4: Cross-Account Monitoring

This layer provides system-wide observability and enables coordinated detection of security events across trust boundaries. By centralizing monitoring in the vendor-controlled environment, the architecture ensures that security-relevant signals cannot be suppressed or manipulated by adversaries operating within the customer account.

Security events generated by all layers are forwarded to a centralized event bus, where they are aggregated and analyzed to detect anomalous behavior. These events include authentication failures, replay attempts, input validation triggers, and output filtering signals, allowing the system to correlate activity across different stages of the request lifecycle.

Based on this aggregated view, the system enforces alerting and response policies that escalate suspicious activity and enable timely intervention. In addition, all events are recorded in an immutable audit log, ensuring that security incidents can be retrospectively analyzed even in the presence of partial system compromise.

Together, these mechanisms provide a final layer of defense that complements the preceding stages by enabling detection, accountability, and coordinated response across administrative domains.

5 Reference Implementation

We provide a reference implementation of the proposed architecture using serverless cloud services, enabling automatic scaling and cost-efficient deployment. The system is defined entirely as Infrastructure-as-Code (IaC) using Terraform³, and can be deployed across major cloud providers, with direct equivalents on Azure and Google Cloud, as discussed in section 4.

The implementation follows the two-account model described in Section 4, separating trusted vendor-side components from customer-controlled infrastructure. The vendor account hosts the gatekeeper, encrypted prompt storage, and centralized monitoring services, while the customer account hosts the deployed agent and API interface.

The deployment proceeds in three phases. First, during the *build* phase, the agent container is constructed with secure handling of cryptographic material, ensuring that sensitive keys are not persisted in the resulting image. Second, during the *provisioning* phase, infrastructure resources are instantiated in the

³ The reference implementation and experimental artifacts are available at <https://github.com/dpetrocelli/defense-in-depth-agent-ip>.

customer account through cross-account access policies. Finally, in the *activation* phase, the agent retrieves its prompt from the gatekeeper via an authenticated channel and maintains it only in process memory during execution.

All infrastructure modules, deployment scripts, and configuration files are available in the reference implementation repository, enabling full reproducibility of the system.

6 Validation

6.1 Penetration Testing

We designed 63 test cases across 12 categories aligned with the OWASP LLM Top 10 [23] and the National Institute of Standards and Technology (NIST) taxonomy [22]. Each test specifies: attack vector, targeted goal, expected defense layer, payload, and pass/fail criterion. Table 2 presents the breakdown, covering representative attack patterns aligned with established taxonomies rather than exhaustive adversarial exploration.

Table 2. Penetration test results (63 tests, 100% pass rate).

Category	OWASP	Tests	Pass	Layer
Direct extraction	LLM01/06	8	8	L2+L3
Multi-turn extraction	LLM01	5	5	L2+L3
Jailbreak / Do Anything Now (DAN)	LLM01	7	7	L2
Instruction override	LLM01	6	6	L2
Prompt injection	LLM01	6	6	L2
Encoding evasion	LLM01	4	4	L2
Typoglycemia evasion	LLM01	3	3	L2
Social engineering	LLM01	5	5	L2
Output leakage	LLM02/06	6	6	L3
Personally Identifiable Information (PII) exfiltration	LLM06	4	4	L3
Response safety	LLM02	5	5	L3
Replay / auth bypass	—	4	4	L1
Total		63	63	

Key findings indicate that all authentication bypass attempts (Layer 1), including expired timestamps, duplicate nonces, forged signatures, and missing headers, were correctly rejected with Hypertext Transfer Protocol (HTTP) 403 responses and no information leakage. Input validation (Layer 2) effectively handled 44 input-facing tests, ranging from naive extraction attempts to sophisticated multi-turn social engineering and encoding evasion, with the typoglycemia detector successfully blocking obfuscated variants that bypass naive regex-based approaches. Output filtering (Layer 3) demonstrated complementary protection, as the sliding-window similarity check detected partial leakage (50+ character fragments) not captured by pattern-based methods alone. Finally, cross-layer interactions were validated through 8 tests targeting boundary conditions, showing that attacks reducing Layer 2 scores were still mitigated by Layer 3 mechanisms.

6.2 Live Two-Account Deployment

To validate the architecture under realistic conditions, we deployed the full four-layer stack across two separate cloud accounts (vendor and customer) and executed 42 attack queries across 8 sophistication categories plus 23 legitimate queries. Table 3 reports the results.

Table 3. Live deployment: attack detection by category (42 attacks, 23 legitimate). FP = false positive.

Attack category	Blocked	Rate	Defense
Override commands	5/5	100%	L2 (risk score)
Jailbreak / DAN	7/7	100%	L2 (risk score)
Encoded payloads	4/4	100%	L2 (encoding det.)
Typoglycemia	4/4	100%	L2 (typo det.)
Social engineering	5/5	100%	L2+L3
Indirect injection	4/4	100%	L2 (structural)
Naive extraction	6/8	75%	L2+L3+defensive prompt
Multi-turn	3/5	60%	L2+L3 (escalation)
Total attacks	38/42	90.5%	
Legit (FP)	0/23	0.0%	

The unblocked queries correspond to reconnaissance probes that do not extract prompt content. The architecture therefore allows ambiguous interactions while preventing disclosure of sensitive information. Multi-turn escalation shows the lowest detection rate because input validation is stateless: each request is scored independently, and escalation sessions distribute intent across individually benign turns—only one of the five sessions contains a turn that reaches the blocking threshold on its own. The blocked sessions were intercepted by the combination of escalation heuristics and output-side leakage filtering (L2+L3) once responses began approaching protected content. Session-level cumulative risk scoring across conversation turns is a natural extension to close this gap. Concurrency testing showed stable throughput up to 50 concurrent requests (13.8 requests per second), with degradation at higher concurrency due to infrastructure limits.

To enable experimentation without requiring cloud accounts, we provide a Docker Compose environment combining LocalStack for cloud service emulation and Ollama for local LLM inference (Llama 3.2, Gemma 3 and 4, Phi-4 Mini). The setup reproduces the two-account deployment model using two isolated LocalStack instances: a central account hosting the gatekeeper function behind an API endpoint, and a client account operating under a separate identity. The gatekeeper executes the same production code as in the cloud deployment, enabling end-to-end validation without code modifications. This setup preserves the cross-domain trust boundary under controlled conditions, although mechanisms dependent on model behavior remain sensitive to the underlying LLM.

6.3 Performance

Table 4 reports per-layer latency overhead ($n = 1,000$ requests, AWS reference implementation).

Table 4. Latency overhead per defense layer (ms, $n = 1,000$).

Layer	P50	P95	P99
L1: HMAC validation	2.1	3.8	5.2
L2: Risk scoring	8.3	14.7	19.1
L3: Output filter + ZWC	12.5	22.3	31.6
L4: Event forwarding	4.1	8.2	12.4
Total	27.0	49.0	68.3

Total P99 overhead is 68.3 ms—well below the 100–200 ms threshold imperceptible in conversational AI. Cold-start latency averages 1.2 s for container-based functions; provisioned concurrency eliminates this at ~\$4.50/month per instance (not included in the base infrastructure cost reported below).

6.4 Cost Analysis

Table 5 reports the infrastructure cost of the defense layers across three cloud providers. In this context, Function-as-a-Service (FaaS) refers to serverless function execution used to implement the gatekeeper component. Notably, this cost is *independent of model inference*: the LLM invocation cost remains unchanged, as the defense mechanisms operate at the transport and application layers without modifying inference calls.

Table 5. Monthly infrastructure cost of defense layers by provider (50K requests).

Component	Service primitive	AWS	Azure	GCP
Secrets + Encryption	Managed secrets + KMS key	\$1.40	\$1.60	\$1.06
Gatekeeper + API	FaaS function + HTTP endpoint	\$0.05	\$0.00	\$0.00
Event Bus	Cross-account event routing	\$0.00	\$0.00	\$0.00
Audit Storage	Immutable log storage (90-day)	\$0.25	\$0.25	\$0.25
Monitoring	Log ingestion + retention	\$0.03	\$0.10	\$0.05
Container Registry	Image storage (~100 MB)	\$0.01	\$5.01	\$0.01
Total		\$1.74	\$6.96	\$1.37

The results show that the proposed architecture introduces minimal infrastructure overhead, remaining negligible relative to model inference costs. This demonstrates that the approach is economically feasible for real-world deployments, even at moderate scale.

7 Discussion and Conclusions

This work presented a defense-in-depth architecture for protecting AI agent intellectual property in multi-tenant cloud deployments. By composing cryptographic prompt delivery, OWASP-aligned input validation, output filtering with steganographic watermarking, and cross-account monitoring into a sequential trust chain, the proposed approach achieves prompt confidentiality, output integrity, and forensic traceability using standard cloud services, without requiring TEEs or specialized hardware. The architecture is cloud-agnostic and demonstrated through a serverless reference implementation deployed across two separate accounts, with experimental results demonstrating strong security effectiveness against realistic attack scenarios, low performance overhead, and practical deployability in real-world settings.

From a systems perspective, the proposed trust model trades hardware-backed guarantees for broader deployability. The root of trust relies on correctly configured cloud IAM, which is weaker than hardware attestation but universally available. This limitation is mitigated through the defense-in-depth design: even if a single layer is compromised (e.g., due to IAM misconfiguration), the remaining layers continue to enforce protection. In this sense, the proposed architecture and TEE-based approaches should be viewed as complementary rather than competing. Recent benchmarks indicate that TEE overhead has decreased significantly, with CPU overhead below 10% and GPU overhead below 7% [10], suggesting that high-assurance deployments may benefit from combining both approaches.

The architecture generalizes naturally across cloud providers, as each component maps to equivalent services in Azure and GCP environments, enabling portability without altering the underlying design principles. However, several limitations remain. First, HMAC key rotation currently requires container rebuilds. Second, risk scoring thresholds may require per-deployment tuning. Third, some pipelines may strip zero-width watermarking signals. Additionally, the system has not been evaluated against adversarial fine-tuning scenarios, and cold-start latency in serverless environments may require mitigation through provisioned concurrency. Moreover, the adversarial evaluation is static: we did not evaluate runtime memory scraping by a privileged customer administrator ($\mathcal{A}_1$), who could extract the decrypted prompt from process memory, nor adaptive adversaries that iteratively refine inputs based on observed blocking behavior; per-account watermarked prompt variants make such leaks attributable rather than preventable, and adaptive red-teaming remains future work. Recent findings on steganographic collusion [21] also suggest potential risks in covert signaling mechanisms, motivating further monitoring strategies.

From an ethical standpoint, the proposed watermarking mechanism is designed to identify accounts and sessions rather than individual users. Its use should be transparently disclosed and restricted to intellectual property protection and forensic analysis, in line with standard privacy practices.

Future work will focus on extending the architecture in several directions, including optional integration with TEE-based inference, automated HMAC key rotation with zero-downtime updates, formal verification of security properties,

and evaluation against adaptive adversaries. Additional directions include redundant watermarking strategies combining ZWC with token-based methods [18,12], multi-model evaluation across different LLM families, and the incorporation of stateful detection mechanisms for multi-turn interactions.

References

1. Agarwal, D., Fabbri, A.R., Risher, B., Laban, P., Joty, S., Wu, C.S.: Prompt leakage effect and mitigation strategies for multi-turn LLM applications. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP), Industry Track (2024)
2. Alzahrani, A.: PromptGuard: A structured framework for injection-resilient language models. *Scientific Reports* (2026). <https://doi.org/10.1038/s41598-025-31086-y>, <https://www.nature.com/articles/s41598-025-31086-y>
3. Amazon Web Services: Shared responsibility model. Tech. rep., Amazon Web Services (2023), <https://aws.amazon.com/compliance/shared-responsibility-model/>
4. Bashir, H.M., Li, Q., Hou, J.: A high capacity text steganography utilizing unicode zero-width characters. In: 2020 International conferences on internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics). pp. 668–675. IEEE (2020)
5. Bodea, T., Misono, M., Pritzi, J., Sabanic, P., Sommer, T., Unnibhavi, H., Schall, D., Santos, N., Stavrakakis, D., Bhatotia, P.: Trusted AI agents in the cloud. arXiv preprint arXiv:2512.05951 (2025). <https://doi.org/10.48550/arXiv.2512.05951>, <https://arxiv.org/abs/2512.05951>
6. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., et al.: On the opportunities and risks of foundation models (2021), <https://arxiv.org/abs/2108.07258>, stanford CRFM report
7. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., et al.: Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)* **33** (2020)
8. Cao, B., Li, C., Cao, Y., Ge, Y., Wang, T., Chen, J.: You can’t steal nothing: Mitigating prompt leakages in LLMs via system vectors. In: Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. pp. 4423–4437. CCS ’25, Association for Computing Machinery (2025). <https://doi.org/10.1145/3719027.3765124>, <https://doi.org/10.1145/3719027.3765124>
9. Chan, S.H.: Encrypted prompt: Securing LLM applications against unauthorized actions. arXiv preprint arXiv:2503.23250 (2025). <https://doi.org/10.48550/arXiv.2503.23250>, <https://arxiv.org/abs/2503.23250>
10. Chrapek, M., Copik, M., Mettaz, E., Hoefler, T.: Confidential LLM inference: Performance and cost across CPU and GPU TEEs. In: Proceedings of the 2025 IEEE International Symposium on Workload Characterization (IISWC). pp. 84–98. IEEE (2025). <https://doi.org/10.1109/IISWC66894.2025.00017>, <https://doi.org/10.1109/IISWC66894.2025.00017>
11. Das, B.C., Amini, M.H., Wu, Y.: Prompt injection attacks in large language models and AI agent systems: A comprehensive review. *Information* **17**(1), 54 (2026)

12. Dathathri, S., See, A., Ghaisas, S., Huang, P.S., McAdam, R., Welbl, J., et al.: Scalable watermarking for identifying large language model outputs. *Nature* **634**, 818–823 (2024)
13. European Union: General data protection regulation (gdpr). Tech. rep., European Union (2016), regulation (EU) 2016/679
14. Gim, I., Li, C., Zhong, L.: Confidential prompting: Protecting user prompts from cloud LLM providers. arXiv preprint arXiv:2409.19134 (2024). <https://doi.org/10.48550/arXiv.2409.19134>, <https://arxiv.org/abs/2409.19134>
15. Hui, B., Yang, H., Yuan, H., Gong, N.Z., Lyu, L.: PLeak: Prompt leaking attacks against large language model applications. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS) (2024)
16. Hung, K.H., Ko, C.Y., Rawat, A., Chung, I.H., Hsu, W.H., Chen, P.Y.: Attention tracker: Detecting prompt injection attacks in LLMs. In: Findings of ACL: NAACL 2025. pp. 2309–2322 (2025)
17. Jiang, Z., Jin, Z., He, G.: PromptKeeper: Safeguarding system prompts for LLMs. In: Findings of the Association for Computational Linguistics: EMNLP 2025. pp. 2712–2728. Association for Computational Linguistics (2025). <https://doi.org/10.18653/v1/2025.findings-emnlp.147>, <https://aclanthology.org/2025.findings-emnlp.147/>
18. Kirchenbauer, J., Geiping, J., Wen, Y., Katz, J., Miers, I., Goldstein, T.: A watermark for large language models. In: Proceedings of the 40th International Conference on Machine Learning (ICML) (2023)
19. Lau, G.K.R., Niu, X., Dao, H., Chen, J., Foo, C.S., Low, B.K.H.: Waterfall: Framework for robust and scalable text watermarking and provenance for LLMs. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 20432–20466 (2024)
20. MarketsandMarkets: AI agents market—global forecast to 2030. Tech. rep., MarketsandMarkets Research (2024), report ID: TC-9261. Available at <https://www.marketsandmarkets.com/Market-Reports/ai-agents-market-15761548.html>
21. Mathew, Y., Matthews, O., McCarthy, R., Velja, J., Schroeder de Witt, C., Cope, D., Schoots, N.: Hidden in plain text: Emergence and mitigation of steganographic collusion in LLMs. In: Proceedings of IJCNLP-AAACL (2025)
22. National Institute of Standards and Technology: Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. Tech. Rep. AI 100-2e, NIST (2025)
23. OWASP Foundation: OWASP top 10 for LLM applications 2025. Tech. rep., Open Web Application Security Project (2025)
24. Pathade, C., Dhimam, V., Ahmad, S., Lareb, I.: Serverless AI security: Attack surface analysis and runtime protection mechanisms for FaaS-based machine learning. arXiv preprint arXiv:2601.11664 (2026). <https://doi.org/10.48550/arXiv.2601.11664>, <https://arxiv.org/abs/2601.11664>
25. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero trust architecture. NIST special publication **800**(207), 1–52 (2020)
26. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. *Proceedings of the IEEE* **63**(9), 1278–1308 (1975)
27. Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761 (2023)
28. Schwarz, D.: Countermind: A multi-layered security architecture for large language models. arXiv preprint arXiv:2510.11837 (2025). <https://doi.org/10.48550/arXiv.2510.11837>, <https://arxiv.org/abs/2510.11837>

29. Shahriar, A., Rahman, M.N., Ahmed, S., Sadeque, F., Parvez, M.R.: A survey on agentic security: Applications, threats and defenses. arXiv preprint arXiv:2510.06445 (2025). <https://doi.org/10.48550/arXiv.2510.06445>, <https://arxiv.org/abs/2510.06445>
30. Thornton, S.: TRYLOCK: Defense-in-depth against LLM jailbreaks via layered preference and representation engineering. arXiv preprint arXiv:2601.03300 (2026). <https://doi.org/10.48550/arXiv.2601.03300>, <https://arxiv.org/abs/2601.03300>
31. Wang, L., Ma, C., Chen, X., Yang, Q., Zhang, Q., et al.: A survey on large language model based autonomous agents. arXiv preprint arXiv:2308.11432 (2024)
32. van Wyk, M.A., Bekker, M., Richards, X.L., Nixon, K.J.: Protect your prompts: Protocols for IP protection in LLM applications. arXiv preprint arXiv:2306.06297 (2023). <https://doi.org/10.48550/arXiv.2306.06297>, <https://arxiv.org/abs/2306.06297>
33. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2023)
34. Zhuang, Z., Nicolae, M.I., Wang, H.P., Fritz, M.: ProxyPrompt: Securing system prompts against prompt extraction attacks. arXiv preprint arXiv:2505.11459 (2025). <https://doi.org/10.48550/arXiv.2505.11459>, <https://arxiv.org/abs/2505.11459>