Advances in Knowledge-Based Systems, Data Science, and Cybersecurity Received 11-01-2026; Accepted 01-04-2026; Published 27-04-2026

Security Challenges of LLM Integration in Multi-Tenant SaaS: Threats, Vulnerabilities, and Mitigations

Nazarii Romankiv ayzrian@gmail.com
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
ORCID: 0009-0004-9893-6823

Dmytro Sytnikov ayzrian@gmail.com
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
ORCID: 0000-0003-1240-7900

Corresponding Author: Nazarii Romankiv

Abstract

The rapid pace of Large Language Model (LLM) adoption and incorporation into existing multi-tenant Software-as-a-Service (SaaS) suites brings new security risks that challenges application security paradigms. Retrieval-Augmented Generation (RAG), AI agents with tool access, and newly articulated patterns for context orchestration all provide attack surfaces specific to Generative AI (GenAI) systems. Multi-tenancy creates a further layer of risk through cross-tenant data leakage, shared inference infrastructure, and the difficulty in enforcing tenant isolation when AI models are processing chunks of tokens from multiple tenants in the same infrastructure. In this work, we surfed a wave of content created by AI researchers and researchers studying AI, such as the Open Worldwide Application Security Project (OWASP) Top 10 for LLM Applications (2025), National Institute of Standards and Technology (NIST) AI 600-1, and other more recent academic sources to get the lay of the land regarding security threats residing specifically in multi-tenant SaaS setups with LLM integration hung off the side. We derive a taxonomy of threats categorized by five classes of attack surface vulnerability, and map each threat to specific defenses in a defense-in-depth style framework. The analysis is rooted in real-world SaaS architectures like Context Orchestration Layer (COL), RAG pipelines, and Model Context Protocol (MCP). We prove that 12 out of the 18 discovered vulnerabilities experience greater amplification from multi-tenancy than single-tenant deployments. Cross-tenant data exfiltration and knowledge base poisoning had the highest amplification factors. Defending LLMs in SaaS requires a composite approach to safeguarding input, enforcing tenant isolation, sanitizing output, hardening the supply chain, and monitoring over time. We provide a mitigation matrix tying all our discovered threats to specific defenses, along with guidance for SaaS architects on implementing them.

Keywords: Llm Supply Chain, Llm Security, Data Leakage, Guardrails, Model Context Protocol, Multi-Tenant SAAS, Prompt Injection

Citation: Nazarii Romankiv and Dmytro Sytnikov. Security Challenges of LLM Integration in Multi-Tenant SaaS. **Advances in Knowledge-Based Systems, Data Science, and Cybersecurity; Review**. 2026;3(1).21.

Page number:443-465.

1. INTRODUCTION

1.1 Context and Motivation

The enterprise software industry is undergoing a significant change since most of known SaaS applications are being integrated with Large Language Models (LLMs), such as Jira, ClickUp, Notion, etc. This trend can be seen in various business domains, such as CRM (customer relationship management), CMS (Content Management Systems), business intelligence tools, and even Integrated Development Environment (IDE) tools that now embed code assistance. These all use LLM capabilities to perform content summarization, build conversational interfaces, enable intelligent search, generate insights, and many other tasks [1, 2]. This upward trend started in early 2023 as soon as most major cloud providers started to offer LLM Application Programming Interfaces (APIs) as managed services, at the same time open-source models enabled self-hosted deployment, which addresses enterprise privacy concerns [3, 4].

Patterns for developing thematically consistent SaaS consisting of LLM-based capability have arisen based on solving common requirements of the SaaS model. Examples include: Retrieval-Augmented Generation (RAG) pipelines that provide domain-specific knowledge to model-based capabilities from vector database-stored documents [5], context orchestration layers that create tenant-specific context through dynamic aggregation of memory types (semantic, operational, and episodic) that mitigate lost-in-the-middle [6], and maximize response relevance [7], tool-use capabilities available to AI agents through protocols such as the Model Context Protocol (MCP) that allow AI agents to perform actions across multiple external systems [8, 9], and LLM-based systems for moderating user-generated content to ensure that they comply with organizational policies [10]. Although these patterns provide significant functional benefits, they also introduce a variety of security concerns that do not exist in traditional application security associated with web applications.

LLM integration presents a security paradox in which the very qualities that provide the key LLM capabilities (the ability to perform contextual reasoning on disparate data, the ability to execute tools, and the ability to generate outputs) are also creating new attack vectors. LLMs interpret natural language instructions, which are inherently ambiguous by nature. Due to this fact, the boundary between trusted and untrusted (malicious) instructions becomes fundamentally vague [11, 12].

1.2 The Multi-Tenancy Amplifier

The predominant architectural model of SaaS is that of multi-tenancy. This allows for infrastructure cost efficiencies through the use of shared infrastructure while still providing logical data separation between tenant organizations. Traditional multi-tenant security has focused primarily on three things: controlling access to the database at a logical level; controlling access at an API level; and separating tenants on different networks (through the use of firewalls). The introduction of shared non-tenant resources via LLMs provides an opportunity for the creation of new methods of isolation not previously available.

A typical multi-tenant LLM-powered SaaS application involves the following shared resources: (1) shared inference infrastructure (i.e., Graphics Processing Unit (GPU) compute and key-value (KV) caches), (2) document embeddings from multiple tenants stored together in the same vector

database, (3) shared model weights serving tenant-specific context through common endpoints, and (4) the context window, which acts as a transient shared space while assembling a prompt. These shared resources clearly create risks of potential leakage and interference of information across tenants.

Li et al. [13], demonstrated that, within multimodal LLM-serving, KV-cache sharing allows an attacker to recreate the prompts of another tenant via the use of timing side channels (PROMPT-PEEK). Chen et al. [14], proposed a Secure Multi-Tenant Architecture that employs a Burn After Use approach to achieve 92% defense success, while still acknowledging residual risk from credential misconfigurations (e.g., limited access to only those who have proper authentication credentials) and observability pipelines. Therefore, it is clear that standard multi-tenant isolation (an important aspect of LLM integration) is a necessary, yet not sufficient, element of LLM integrated systems.

1.3 Problem Statement and Contributions

Despite growing awareness of LLM security risks, no comprehensive threat model exists specifically for multi-tenant LLM-powered SaaS. Existing surveys cover LLM security generally [2–4], but do not analyze how multi-tenancy changes the threat landscape. The OWASP Top 10 for LLM Applications [1], provides a valuable baseline taxonomy but does not differentiate between single-tenant and multi-tenant deployment scenarios.

This paper addresses this gap with four contributions:

1. A **threat taxonomy** specific to multi-tenant LLM-powered SaaS, organized into 5 categories and 18 vulnerability classes.
2. A **multi-tenancy amplification analysis** demonstrating that 12 of 18 vulnerabilities are amplified in multi-tenant versus single-tenant deployments.
3. A **defense-in-depth mitigation framework** with five layers mapping threats to specific countermeasures with implementation guidance.
4. A **security evaluation** of concrete SaaS architectures—COL [7], RAG pipelines [15], content moderation [10], and MCP integration [9]—identifying both existing protections and remaining gaps.

1.4 Paper Organization

Section 2 reviews the related work on LLM security, multi-tenant isolation, and emerging protocol-level threats. Section 3 describes the research methodology, reference architecture, and threat modeling approach. Section 4 presents the threat taxonomy, amplification analysis, and mitigation framework. Section 5 discusses implications, case study analysis, and limitations. Section 6 concludes with recommendations for the SaaS industry.

2. RELATED WORK

2.1 LLM Security Surveys

Several comprehensive surveys have mapped the LLM security landscape. Das et al. [3], catalogued security concerns across the LLM lifecycle, covering training-time poisoning, inference-time prompt attacks, and output-related risks, but treated deployment context as uniform. Sun et al. [4], surveyed threats and mitigation strategies with emphasis on privacy-preserving techniques, noting that multi-user deployments remain underexplored. Yao et al. [2], organized LLM risks into beneficial, malicious, and unintentional categories, providing broad coverage but without differentiating single-tenant from multi-tenant threat surfaces. These surveys establish the vocabulary and attack taxonomy used in the field, yet none analyze how shared infrastructure changes the severity or exploitability of the identified threats.

2.2 Prompt Injection and Jailbreaking

Prompt injection has emerged as the most studied LLM application-layer threat. Greshake et al. [11] , established the indirect prompt injection paradigm, demonstrating real-world attacks against Bing Chat and code completion systems. Liu et al. [16], formalized prompt injection attacks and benchmarked five attack types against ten defenses across ten LLMs, concluding that no single defense achieves both high security and high utility. The HouYi framework [17], found 86% of 36 real-world LLM applications vulnerable to prompt injection. Liu et al. [12], provided a comprehensive review of prompt injection in LLM-integrated applications, documenting how compositional injections exploit multi-turn context. On the jailbreaking front, Zou et al. [18], introduced the GCG attack producing universal adversarial suffixes, while Xu et al. [19], benchmarked jailbreak techniques showing 90–99% success on open-weight models. Chu et al. [20], documented the evolutionary trajectory of jailbreaking methods through the JailbreakRadar framework.

2.3 Data Privacy and Leakage in LLMs

Carlini et al. [21], demonstrated verbatim extraction of training data from GPT-2, including PII, establishing that memorization scales with model size. Li et al. [22], created the LLM-PBE framework for systematically assessing privacy risks. Wang et al. [23], identified five distinct PII leakage pathways in LLM agents, while Zhang et al. [24], distinguished memorization-based from inference-based PII leakage. Staab et al. [25], compared privacy guardrail deployments across applications and found significant variation in effectiveness. These works address privacy in general LLM usage but do not consider the compounded risk when PII from multiple tenant organizations coexists within shared retrieval and inference infrastructure.

2.4 RAG Security and Knowledge Base Attacks

RAG systems [5], introduce a distinct attack surface at the retrieval layer. Zou et al. [26], demonstrated that PoisonedRAG achieves 97% attack success with only 5 malicious documents injected

into million-document knowledge bases. Shafran et al. [27], showed that "blocker" documents can suppress retrieval of legitimate content. Xiang et al. [28], formalized the RAG threat model and attack surface, while Chen et al. [29], proposed RevPRAG as a detection mechanism for poisoned retrieval. Morris et al. [30], demonstrated that text embeddings can be inverted to recover 60–80% of original content, threatening the confidentiality of vector-stored documents. These attacks become qualitatively different in multi-tenant settings where a single poisoned document can influence responses served to all tenants on the platform.

2.5 Multi-Tenant LLM Security

Research specifically addressing multi-tenant LLM deployments is limited. Li et al. [13], introduced the PROMPTPEEK attack, demonstrating that KV-cache sharing between tenants enables prompt reconstruction through timing side-channels. Chen et al. [14], proposed the SMTA with a BAU approach, achieving 92% defense success while acknowledging residual risks from credential misconfigurations and observability pipelines. The OWASP Top 10 for LLM Applications (2025) [1], provides a widely adopted baseline taxonomy, and NIST AI 600-1 [31], identifies 12 generative AI-specific risks with a governance framework. However, neither differentiates between single-tenant and multi-tenant deployment scenarios, leaving a gap in understanding how shared infrastructure amplifies the documented threats.

2.6 Supply Chain and Protocol Security

The Model Context Protocol [8], standardizes tool-use for LLMs but introduces novel supply chain risks. Doynikova et al. [32], documented attacks spanning prompt injection and protocol exploitation, including tool description manipulation and rug pull mutations in MCP ecosystems. Mohanty et al. [33], proposed governance frameworks for MCP security, including server verification and sandboxing. Inan et al. [34], introduced Llama Guard for input–output safety classification, and Kumar et al. [35], demonstrated that multi-stage guardrail pipelines reduce attack success rates. The EU AI Act [36], imposes transparency and safety obligations on AI providers, adding regulatory urgency to these technical challenges.

2.7 Positioning of This Work

The existing literature addresses LLM security threats individually but does not provide an integrated threat model for multi-tenant LLM-powered SaaS. General LLM security surveys [2–4], do not analyze deployment-context effects. RAG security research [26–29], does not consider cross-tenant retrieval contamination. Multi-tenant LLM studies [13, 14], focus on specific isolation mechanisms without comprehensive threat categorization. This paper bridges these gaps by synthesizing findings across all four domains into a unified taxonomy, analyzing how multi-tenancy amplifies each threat class, and providing a defense-in-depth framework with specific mitigation mappings.

3. MATERIALS AND METHODS

3.1 Research Methodology

This study employs a systematic literature review combined with architecture-based threat modeling. The approach is analytical rather than experimental: we synthesize empirical findings from the existing literature into a unified threat taxonomy and mitigation framework, rather than conducting new attack experiments. This analytical approach is appropriate for foundational taxonomy work, where the goal is to organize and contextualize a rapidly expanding body of empirical research before targeted experimental validation.

Search strategy. We searched IEEE Xplore, ACM Digital Library, arXiv, Scopus, and Google Scholar between January and March 2026 using queries combining terms from three domains: LLM security ("prompt injection," "jailbreak," "LLM attack," "AI safety"), multi-tenant systems ("multi-tenant," "SaaS security," "tenant isolation"), and specific technologies ("RAG security," "MCP vulnerability," "vector database attack"). The search was restricted to publications from 2021–2026, since these are the years when LLM trends became dominant in the SaaS industry.

Inclusion and exclusion criteria. Papers were included if they: (1) addressed application-layer security of LLM-integrated systems, (2) presented empirical attack demonstrations, formal threat models, or defense evaluations, and (3) were published in peer-reviewed venues or established preprint servers with demonstrable community impact. Papers were excluded if they: (1) focused exclusively on model training-time attacks without application-layer relevance, (2) addressed only infrastructure-level security (network, OS, cloud) without LLM-specific considerations, or (3) were position papers without technical contributions.

Selection process. From an initial corpus of over 200 papers, title and abstract screening reduced the set to 78 candidates. Full-text review yielded 32 external sources selected for relevance to LLM application-layer security, methodological rigor, and citation impact. We supplemented these with 4 self-authored publications describing concrete SaaS architectures that serve as case studies for security analysis [7, 9, 10, 15]. The OWASP Top 10 for LLM Applications (2025 edition) [1] was used as the baseline taxonomy, which we extended to account for multi-tenancy-specific vulnerability classes.

Taxonomy derivation. The threat taxonomy was derived through inductive coding of attack mechanisms reported in the selected literature. We first extracted all distinct attack types and grouped them by the component of the LLM interaction lifecycle they target (input, inference, output, storage, integration). Within each group, we consolidated overlapping attack descriptions into distinct vulnerability classes. The resulting five categories and 18 vulnerability classes were validated against the OWASP Top 10 for LLM Applications [1], to ensure completeness, which revealed three vulnerability classes not captured by the OWASP framework.

Amplification assessment. Multi-tenancy amplification was assessed for each vulnerability class using a four-level scale. *Critical* indicates that multi-tenancy either creates a fundamentally new threat (e.g., cross-tenant data leakage) or elevates severity from High to Critical by enabling platform-wide impact. *High* indicates that shared infrastructure measurably increases the blast radius or exploitability compared to single-tenant deployment. *Medium* indicates a modest increase in risk.

None indicates that the vulnerability operates at the model level and is independent of deployment architecture. Each rating is justified by specific technical mechanisms documented in the literature, as presented in TABLE 1.

3.2 Reference Architecture

To ground our threat analysis in practical systems, we define a reference architecture for a multi-tenant LLM-powered SaaS platform (FIGURE 1). This architecture synthesizes patterns from the authors' prior work and industry practice.

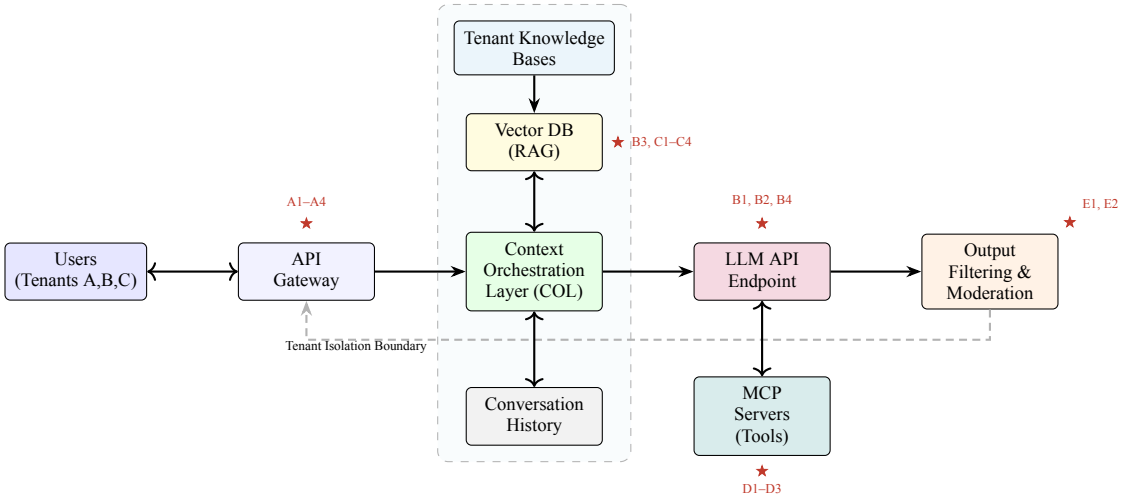

Figure 1: Reference architecture of a multi-tenant LLM-powered SaaS platform. Red stars indicate attack surface points mapped to vulnerability classes from the proposed taxonomy.

The architecture comprises the following components. The **API Gateway** authenticates users and routes requests with tenant context. The **Context Orchestration Layer** [7] assembles the prompt from three memory types: Semantic Memory (retrieved via RAG from the vector database), Operational Memory (live state via API calls), and Episodic Memory (conversation history). Metadata filtering enforces tenant isolation during retrieval through Access Control List (ACL) and Role-Based Access Control (RBAC) middleware [7]. The **Vector Database** stores document embeddings with tenant metadata, enabling similarity search for the RAG pipeline [15]. The **LLM API Endpoint** processes the assembled prompt and generates a response. **MCP Servers** provide tool-use capabilities through the standardized Model Context Protocol [8, 9]. The **Output Filtering** layer applies content moderation [10], and Personally Identifiable Information (PII) redaction before returning responses to users.

3.3 Threat Modeling Approach

We adapt the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) threat modeling framework for LLM-specific threats by decomposing

the attack surface along the LLM interaction lifecycle: (1) input processing, encompassing prompt construction and retrieval; (2) model inference, including shared compute and caching; (3) output processing, covering response generation and tool execution; (4) data storage, encompassing vector databases, conversation histories, and tenant data; and (5) external integration via MCP servers, APIs, and plugins.

We consider three adversary models: an **external attacker** operating as a malicious tenant with standard API access; an **insider threat** with elevated access to platform infrastructure; and a **compromised supply chain** where third-party models, tools, or plugins contain malicious payloads. Our analysis focuses on application-layer threats, excluding infrastructure-level attacks (network, OS) that are addressed by standard cloud security practices.

3.4 Classification Framework

Each vulnerability class is characterized along four dimensions: (1) *description* of the attack mechanism; (2) *multi-tenancy amplification*, rated as Critical, High, Medium, or None depending on whether the shared infrastructure exacerbates the vulnerability; (3) *attack vector* specifying how the vulnerability is exploited; and (4) *OWASP mapping* to the corresponding LLM Top 10 entry where applicable. Severity is assessed as the product of impact, likelihood, and multi-tenancy amplification factor.

4. RESULTS

4.1 Proposed Threat Taxonomy

We organize the identified threats into five categories (FIGURE 2) encompassing 18 vulnerability classes. Each category corresponds to a distinct segment of the LLM interaction lifecycle within the reference architecture.

4.1.1 Category A: Input manipulation attacks

A1. Direct Prompt Injection. A malicious actor can generate an input for a Large Language Model (LLM) that attempts to bypass its operational instructions and access private data, manipulate the model's output, or otherwise compromise its integrity. Liu et al. [16], formalized attacks based on prompt manipulation and benchmarked five different types of prompt manipulation attacks against ten forms of defensive measures across ten different LLMs, concluding that none of the existing forms of prompt manipulation was found to have acceptable security as well as acceptable ability to meet business needs. The HouYi attacks [17], found that of the thirty-six applications utilizing LLMs that had been used in real-world scenarios, thirty-one (eighty-six percent) were vulnerable to prompt manipulation attacks. In a multi-tenant SaaS application environment, the extent of direct prompt manipulation attacks will be similar in nature to that of a single-tenant environment because prompt injection attacks target how LLMs are expected to respond to instructions rather than compromising the infrastructure utilized by multiple tenants. *OWASP mapping:* LLM01 (Prompt Injection) [1].

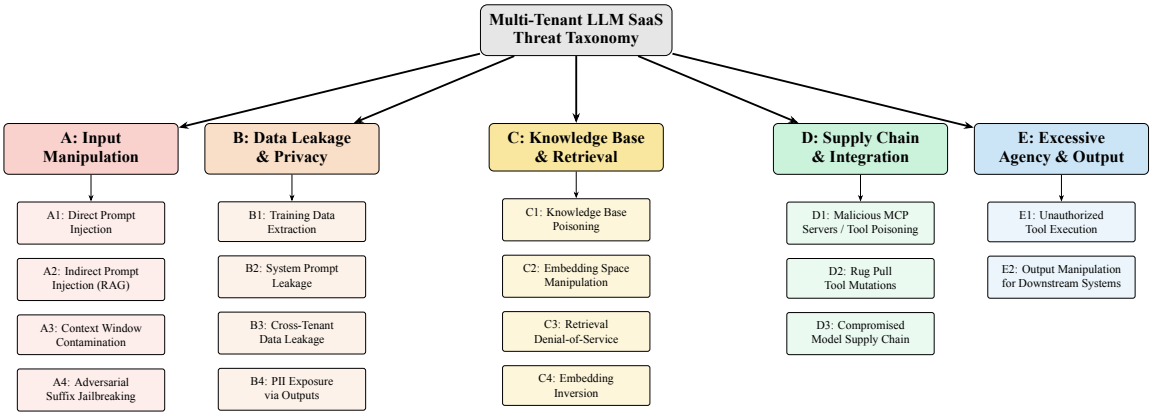

Figure 2: Threat taxonomy for multi-tenant LLM-powered SaaS: 5 categories and 18 vulnerability classes.

A2. Indirect Prompt Injection via Retrieved Content. Embedded documents, knowledge base entries, or code can contain malevolent commands that the RAG pipeline accesses and deliver to the context of the model. Greshake et al. [11], established this threat category as a robust foundation by showing how attack methods carried out against Bing Chat and code completion systems, redirected the behaviour of the model through adversarial data in the retrieved content. In multi-tenant SaaS, this vulnerability is *critically amplified*: if metadata filtering fails or is bypassed, poisoned documents uploaded by one tenant can be retrieved during another tenant's queries. The COL architecture [7] mitigates this through metadata filtering during retrieval, but the filtering mechanism itself becomes a single point of failure—if an attacker can manipulate document metadata, cross-tenant injection becomes possible. *OWASP mapping:* LLM01 [1].

A3. Context Window Contamination. Adversarial content injected into conversation history (Episodic Memory) persists across turns and influences subsequent model responses. In the COL architecture [7], conversation history is compressed using a sliding window with summarization, meaning malicious content injected early in a conversation could be condensed into summary representations that persist indefinitely. Multi-tenancy amplifies this if conversation histories are improperly isolated, enabling contamination to propagate across tenant boundaries. Liu et al. [12], and Doynikova et al. [32], documented how compositional prompt injections exploit multi-turn context to progressively hijack agent behavior. *OWASP mapping:* LLM01 (Prompt Injection) [1].

A4. Jailbreaking Through Adversarial Suffixes. Automated techniques generate adversarial token sequences that bypass model alignment. Zou et al. [18] introduced the Greedy Coordinate Gradient (GCG) attack, producing universal adversarial suffixes transferable across both open-source and commercial LLMs including ChatGPT, Claude, and LLaMA-2. Comprehensive benchmarking [19], shows that advanced automated attacks achieve 90–99% success rates on open-weight models and 80–94% on proprietary models. Adversarial suffixes are a way of bypassing content moderation systems [10], by embedding them in tenant uploaded documents which get retrieved from the chunks through RAG pipelines. This is a continuation of the ongoing arms race that is

demonstrated through the evolutionary history of jail-breaking techniques documented by Cu et al. [20]. *OWASP mapping:* LLM01 [1].

4.1.2 Category B: Data leakage and privacy violations

B1. Training Data Extraction. LLMs store parts of the data they are trained on. By creating appropriate prompts, these can be extracted. Carlini and colleagues [21], have shown the ability to extract hundreds of pieces of verbatim text from GPT-2, which includes personally identifiable information and shows that the amount of memorisation increases with the size of the model used. Li and colleagues [22], have created a systematic framework to evaluate the privacy threat associated with LLMs. And the use of multi-tenancy results in an increased threat to privacy when fine-tuning LLMs with multi-tenant data, since one tenant can extract proprietary data from another tenant. *OWASP mapping:* LLM02 (Sensitive Information Disclosure) [1].

B2. System Prompt Leakage. Attackers capture system prompt data that includes tenant-specific configurations, business logic, API keys, and access patterns. With multi-tenant SaaS applications, system prompts often contain tenant-specific behavior defined using parameterized templates, and leaking system prompts exposes the platform's customization architecture potentially allowing attackers to perform similar attacks against other tenants in the system since other tenants' configurations often follow the same general customization patterns as yours do. *OWASP mapping:* LLM07 (System Prompt Leakage) [1].

B3. Cross-Tenant Data Leakage via Shared Infrastructure. This vulnerability class is *unique to multi-tenant deployments* and represents the highest-severity threat in our taxonomy. Li et al. [13], introduced the PROMPTPEEK attack, demonstrating that KV-cache sharing between tenants enables prompt reconstruction through cache timing side-channels. Beyond cache-based leakage, shared embedding spaces in vector databases create channels where one tenant's queries may retrieve semantically similar documents from another tenant's corpus if metadata filtering is insufficiently strict [14]. GPU memory residuals from prior inference requests represent an additional leakage channel at the infrastructure level. *No OWASP equivalent*—this is a novel vulnerability class specific to multi-tenant LLM deployments.

B4. PII Exposure Through LLM Outputs. LLMs may generate responses containing PII from retrieved context or memorized training data. Wang et al. [23], identified five distinct PII leakage pathways: direct chat leakage, indirect context leakage via agents, attribute inference, and attribute aggregation. Zhang et al. [24], distinguished between memorization-based and inference-based PII leakage. Multi-tenancy critically amplifies this because the PII of multiple tenants' users exists within the shared retrieval infrastructure, and a failure in output filtering exposes PII from any tenant. Staab et al. [25], compared privacy guardrail implementations and found significant variation in effectiveness across deployment scenarios. *OWASP mapping:* LLM02 [1].

4.1.3 Category C: Knowledge base and retrieval attacks

C1. RAG Knowledge Base Poisoning. Adversarial documents are injected into the vector database to manipulate retrieval results. Zou et al. [26], demonstrated that PoisonedRAG achieves 97% attack

success on Natural Questions, 99% on HotpotQA, and 91% on MS-MARCO with only 5 malicious texts injected into knowledge bases containing millions of documents. The fundamental vulnerability is that RAG systems treat all documents in the knowledge base as equally trustworthy. In multi-tenant SaaS, this is critically amplified: a single poisoned document uploaded by a malicious tenant can influence responses for all tenants if metadata filtering does not perfectly isolate retrieval scopes. Xiang et al. [28], formalized this threat model for RAG systems, while Chen et al. [29], proposed RevPRAG as a detection mechanism. *OWASP mapping:* LLM04 (Data and Model Poisoning) [1].

C2. Embedding Space Manipulation. Adversarial embeddings crafted to be semantically close to target queries but containing malicious content can cross tenant boundaries in shared vector databases [1]. The OWASP entry for Vector and Embedding Weaknesses [1] identifies adversarial embeddings that match arbitrary queries as a key threat vector. Multi-tenancy amplifies this because shared embedding spaces mean that an adversary's carefully crafted embeddings coexist with legitimate tenant data. *OWASP mapping:* LLM08 (Vector and Embedding Weaknesses) [1].

C3. Retrieval Denial-of-Service. Shafran et al. [27], demonstrated that "blocker" documents can be crafted to suppress retrieval of legitimate content through adversarial manipulation of document embeddings. In multi-tenant SaaS, this enables targeted suppression of a competitor tenant's knowledge base, effectively degrading their service quality without affecting the attacker's own operations. *No direct OWASP equivalent.*

C4. Embedding Inversion. Morris et al. [30], demonstrated that text embeddings can be inverted to recover 60–80% of original text content for sentence-level embeddings. If vector databases store embeddings without strict tenant isolation, an adversary with access to the embedding space can reconstruct other tenants' proprietary documents. *OWASP mapping:* LLM08 (Vector and Embedding Weaknesses) [1].

4.1.4 Category D: Supply chain and integration attacks

D1. Malicious MCP Servers and Tool Poisoning. The Model Context Protocol [8], enables LLMs to interact with external tools through a standardized client-server architecture [9]. However, malicious MCP servers can exploit this trust relationship. Documented attacks include: silent exfiltration of a user's entire WhatsApp history through tool poisoning combined with legitimate MCP servers; prompt injection via GitHub issues against the official GitHub MCP server to hijack AI assistants and leak private repository data; and the Postmark MCP server compromise enabling silent BCC email exfiltration [32]. Doynikova et al. [32], documented how tool description manipulation enables attacks that span prompt injection and protocol exploitation. Mohanty et al. [33], proposed governance frameworks for MCP security. Multi-tenancy critically amplifies this: a compromised tool used by any tenant could exfiltrate data from all tenants served by the platform. *OWASP mapping:* LLM03 (Supply Chain) [1].

D2. "Rug Pull" Tool Mutations. Tools used in your MCP can be redefined by MCP's owner after they are installed - a tool that was previously approved as "safe" may suddenly be "outfitted" in such a way that it now will steal your data or perform some sort of malicious act [32]. Because of the ability for MCPs to dynamically discover tools, changes made to their definitions take effect immediately, preventing you from redeploying those tools as is commonly done with traditional supply

chain attacks. Examples include unauthenticated remote execution in Anthropic MCP Inspector and CVE-2025-6514 command injection in mcp-remote [32]. Damage to your SaaS service from a rug pull mutation will affect all of your tenants at the same time. *No direct OWASP equivalent*—this is a novel vulnerability class specific to MCP ecosystems.

D3. Compromised Model Supply Chain. Models that have been pre-trained can contain backdoor implants, whereas backdoor attacks on malicious model updates are by their very nature exploited by all tenants that use the compromised model (regardless of whether or not tenant use is evenly proportioned). While multi-tenancy does not exponentiate the vulnerability, the *blast radius* of this vulnerability will be determined by the number of tenants using the model. *OWASP mapping:* LLM03 [1].

4.1.5 Category E: Excessive agency and output risks

E1. Unauthorized Tool Execution. Prompt manipulation can lead to the execution by LLMs of Tools or Functions that aren't the original intent of the design (Creating a Cross-Tenant Operation). For example, an LLM could modify the data of another Tenant by calling a Tool intended for the LLM to interact with its own data. As part of the COL Architecture, the Operational Memory [7], contains function definitions that allow real-time access to data, so if an attacker were able to manipulate which Tools the LLM can select, then unauthorized operations could cross Tenant boundaries. *OWASP mapping:* LLM06 (Excessive Agency) [1].

E2. Output Manipulation for Downstream Systems. LLMs may generate malicious outputs—SQL injection, cross-site scripting, or command injection payloads—consumed by downstream SaaS components. In multi-tenant SaaS, shared backend services may process LLM-generated payloads without adequate sanitization, enabling attacks that propagate across the platform. *OWASP mapping:* LLM05 (Improper Output Handling) [1].

4.2 Multi-Tenancy Amplification Analysis

TABLE 1, presents the Multi-Tenancy Amplification Matrix, analyzing how each vulnerability class is affected by the transition from single-tenant to multi-tenant deployment. We identify 12 of 18 vulnerability classes that exhibit amplification, with 4 exhibiting Critical amplification, 8 exhibiting High amplification, and 6 exhibiting None (severity remains unchanged).

The key findings from this analysis are as follows. First, **cross-tenant data leakage (B3) is unique to multi-tenancy**—this vulnerability class does not exist in single-tenant deployments and represents a fundamentally new threat introduced by the shared infrastructure model. Second, **RAG poisoning (C1) exhibits the highest practical amplification factor** because a single poisoned document from one malicious tenant can influence responses served to all other tenants, achieving what PoisonedRAG [26], demonstrated at 97% success with 5 adversarial documents. Third, **MCP tool poisoning (D1) has high amplification** because tools are typically shared across tenants—a single compromised MCP server can exfiltrate data from the entire platform. Fourth, **metadata filtering as implemented in the COL architecture** [7], reduces amplification for retrieval-

Table 1: Multi-Tenancy Amplification Matrix: assessment of how multi-tenant deployment affects each vulnerability class.

ID	Vulnerability Class	Single	Multi	Amp.	Root Cause of Amplification
A1	Direct Prompt Injection	High	High	None	Model-level; independent of tenancy
A2	Indirect Prompt Injection	High	Critical	**Yes**	Cross-tenant document retrieval
A3	Context Window Contam.	Med	High	**Yes**	Shared conversation infrastructure
A4	Adversarial Suffixes	High	High	None	Model-level; independent of tenancy
B1	Training Data Extraction	Med	High	**Yes**	Fine-tuned models contain multi-tenant data
B2	System Prompt Leakage	Med	High	**Yes**	Prompts contain tenant-specific configs
B3	Cross-Tenant Leakage	N/A	Critical	**Unique**	Exists *only* in multi-tenant deployments
B4	PII Exposure	High	Critical	**Yes**	PII from multiple tenants in shared context
C1	RAG Poisoning	High	Critical	**Yes**	One tenant poisons; all tenants affected
C2	Embedding Manipulation	Med	High	**Yes**	Shared embedding space
C3	Retrieval DoS	Med	High	**Yes**	Targeted suppression of competitor
C4	Embedding Inversion	Med	High	**Yes**	Shared vector DB stores all tenants' data
D1	MCP Tool Poisoning	High	Critical	**Yes**	Shared tools serve all tenants
D2	Rug Pull Mutations	Med	High	**Yes**	Tool change affects all tenants
D3	Supply Chain Compromise	High	High	None	Model-level; blast radius scales
E1	Unauthorized Tool Exec.	High	Critical	**Yes**	Cross-tenant actions possible
E2	Output Manipulation	Med	Med	None	Same risk regardless of tenancy

Amplification: 4 Critical (unique or maximum increase), 8 High, 6 None. Total amplified: 12/18.

dependent vulnerabilities (A2, C1–C4), but this creates a concentrated dependency on the filtering mechanism's correctness and bypass resistance.

4.3 Defense-in-Depth Mitigation Framework

We propose a five-layer defense-in-depth architecture (FIGURE 3) that maps each layer to the threat categories it addresses. TABLE 2 provides the detailed threat-to-mitigation mapping.

Layer 1: Input Defense and Prompt Hardening. This layer intercepts malicious input before it reaches the LLM. Perplexity-based detection [36] identifies adversarial suffixes that exhibit anomalous token distributions. Classifier-based approaches, such as Llama Guard [34] deployed as input screening middleware, filter prompts matching known attack patterns. Input parameterization separates user data from system instructions through structured prompt templates, reducing the effectiveness of injection attacks. Kumar et al. [35] demonstrated that multi-stage guardrail pipelines significantly reduce attack success rates. Systematic security benchmarking [16] provides methodology for evaluating input defense effectiveness.

Layer 2: Tenant Isolation Enforcement. This is the most critical layer for multi-tenant deployments. Strict metadata filtering with cryptographic verification extends the COL architecture's [7], basic metadata filtering by ensuring that tenant identifiers cannot be spoofed or bypassed. Separate

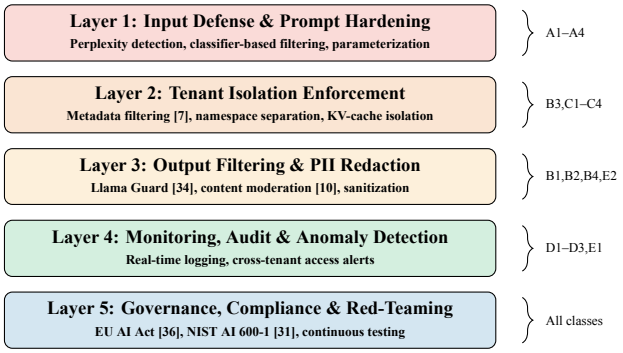

Figure 3: Defense-in-depth mitigation architecture with five layers. Braces indicate primary threat categories addressed by each layer.

vector namespaces per tenant eliminate the shared embedding space that enables C1–C4 attacks, at the cost of increased storage requirements. KV-cache isolation prevents the PROMPTPEEK attack [13] by ensuring that no cache entries are shared between tenants, with an estimated performance overhead of 15–30% [14]. Chen et al. [14] demonstrated that their Secure Multi-Tenant Architecture (SMTA) achieves 92% defense success rate through rigorous semantic isolation.

Layer 3: Output Filtering and PII Redaction. Safety classifiers such as Llama Guard [34] screen both input and output for unsafe content. PII redaction engines scan generated responses for personally identifiable information before delivery. Response grounding verification extends the COL architecture's "Grounding via Context" principle [7], by adding security checks that verify responses are derived solely from authorized context. Content moderation [10], provides multi-category classification of output content.

Layer 4: Supply Chain Security and Monitoring. MCP server verification and sandboxing ensure that external tools operate within defined security boundaries [33]. Tool definition pinning prevents rug pull mutations by cryptographically signing tool definitions at installation time and rejecting runtime modifications [32]. Real-time prompt and response logging with privacy-preserving storage enables forensic analysis. Anomaly detection systems flag unusual cross-tenant access patterns, sudden changes in query distributions, or tool invocations outside normal parameters.

Layer 5: Governance, Compliance, and Red-Teaming. The EU AI Act [36], imposes transparency and safety testing obligations on providers of general-purpose AI models, with penalties up to 7% of global turnover. NIST AI 600-1 [31], identifies 12 generative AI-specific risks and provides a governance framework for risk management. Automated red-teaming through continuous security testing validates the effectiveness of Layers 1–4. Compliance reporting ensures that tenant data handling meets General Data Protection Regulation (GDPR) requirements and data residency constraints.

Table 2: Threat-to-mitigation mapping matrix: defense layers addressing each vulnerability class with key implementation mechanisms.

ID	Vulnerability	L1	L2	L3	L4	L5	Key Mitigation Mechanisms
A1	Direct Prompt Injection	•		•	•		Perplexity filter, Llama Guard, prompt parameterization
A2	Indirect Prompt Injection	•	•	•	•		Metadata filtering, document integrity verification
A3	Context Contamination	•	•		•		History isolation, summarization sanitization
A4	Adversarial Suffixes	•		•	•	•	Perplexity detection, red-team testing, model updates
B1	Training Data Extraction			•	•	•	Output filtering, access logging, differential privacy
B2	System Prompt Leakage	•	•	•	•		Prompt encryption, output screening, access control
B3	Cross-Tenant Leakage		•		•	•	KV-cache isolation, namespace separation, SMTA [14]
B4	PII Exposure			•	•	•	PII redaction, privacy guardrails [25], GDPR compliance
C1	RAG Poisoning	•	•		•		Document integrity, RevPRAG [29], anomaly detection
C2	Embedding Manipulation		•		•		Namespace separation, embedding validation
C3	Retrieval DoS		•		•		Rate limiting, retrieval quality monitoring
C4	Embedding Inversion		•			•	Encryption at rest, namespace isolation, access audit
D1	MCP Tool Poisoning				•	•	Server verification, sandboxing, allowlisting [33]
D2	Rug Pull Mutations				•	•	Definition pinning, cryptographic signing
D3	Supply Chain Compromise				•	•	Model provenance, dependency auditing
E1	Unauthorized Tool Exec.	•	•		•		Least-privilege tool access, human-in-the-loop
E2	Output Manipulation			•	•		Output sanitization, injection prevention

5. DISCUSSION

5.1 Comparison with OWASP Top 10 for LLM Applications

TABLE 3 compares our taxonomy with the OWASP Top 10 for LLM Applications (2025) [1], highlighting both the alignment and the three novel vulnerability classes that our taxonomy introduces.

Table 3: Comparison of the proposed taxonomy with OWASP Top 10 for LLM Applications (2025).

OWASP	OWASP Name	Our ID	Extension
LLM01	Prompt Injection	A1–A4	Multi-tenant RAG injection
LLM02	Sensitive Info Disclosure	B1,B2,B4	Cross-tenant PII
LLM03	Supply Chain	D1–D3	MCP rug pull
LLM04	Data/Model Poisoning	C1, C2	Cross-tenant RAG
LLM05	Improper Output	E2	Shared backends
LLM06	Excessive Agency	E1	Cross-tenant actions
LLM07	System Prompt Leak	B2	Tenant configs
LLM08	Vector/Embedding	C2, C4	Shared vector DB
Novel classes not in OWASP:			
–	Cross-Tenant Leakage	B3	Multi-tenant unique
–	Retrieval DoS	C3	RAG-specific
–	Rug Pull Mutations	D2	MCP-specific

The OWASP Top 10 [1] provides a broadly applicable taxonomy of LLM risks but treats each vulnerability in isolation from the deployment context. Our analysis demonstrates that multi-tenancy fundamentally changes the threat landscape in three ways.

First, multi-tenancy introduces an entirely new threat class—**cross-tenant data leakage (B3)**—that has no single-tenant equivalent. The PROMPTPEEK attack [13] and embedding inversion techniques [30] exploit shared infrastructure in ways that are architecturally impossible in single-tenant deployments.

Second, multi-tenancy **amplifies the blast radius** of existing vulnerabilities. RAG poisoning (C1) in a single-tenant context affects only the poisoning tenant; in a multi-tenant system, it can affect every tenant on the platform. Similarly, MCP tool poisoning (D1) in a multi-tenant context exposes the entire platform's data rather than a single organization's.

Third, multi-tenancy creates **adversarial dynamics between tenants** that do not exist in single-tenant systems. Retrieval denial-of-service (C3) can be weaponized as a competitive tool, where a malicious tenant deliberately degrades a competitor's service quality.

5.2 Security Analysis of Prior Architectures

We evaluate four concrete architectures from the authors' prior work through the lens of the proposed threat taxonomy.

Context Orchestration Layer (COL) [7]. *Strengths:* The COL architecture provides baseline tenant isolation through metadata filtering during retrieval. The PII verification step in the context assembly pipeline addresses B4 by screening retrieved chunks for sensitive information before they enter the model's context. The "Grounding via Context" principle constrains the model to respond based on provided context, limiting hallucination-based leakage. The intent classifier (informational, transactional, analytical) provides an implicit security benefit by limiting the scope of context loaded for each query type.

Weaknesses: Metadata filtering is vulnerable to bypass if an attacker can manipulate document metadata (A2 amplification). There is no explicit protection against adversarial content within retrieved chunks—the re-ranking module optimizes for relevance, not security, meaning adversarial documents that are semantically relevant to a query will be prioritized. The sliding window history compression could propagate injected content through summarized representations (A3).

Recommended enhancements: Cryptographic tenant tagging for metadata integrity, adversarial content detection in the retrieval pipeline, and input/output guardrails [34], integrated as COL middleware.

RAG Pipeline for Codebase Q&A [15]. *Strengths:* The OpenSearch vector storage provides native ACL capabilities, and the code-specific domain limits the applicability of some generic attacks. The ingestion pipeline's use of Claude Haiku for summarization creates an intermediate representation that may dilute adversarial content.

Weaknesses: No adversarial document detection (C1), meaning the ingestion pipeline will index poisoned code files without screening. Embedding inversion risk (C4) exists because text-embedding-3-large embeddings stored in OpenSearch could be inverted to recover source code. The external API calls to Claude for both summarization and Q&A transmit potentially sensitive code over the network, creating data-in-transit exposure.

Recommended enhancements: Document integrity verification before indexing, embedding encryption at rest, and evaluation of self-hosted LLM options for processing sensitive code.

LLM Content Moderation [10]. *Strengths:* Multi-category classification across hate speech, PII, advertising, and other categories provides broad coverage. The structured JSON output format with action labels enables deterministic downstream handling.

Weaknesses: The moderator itself is an LLM susceptible to jailbreaking (A4)—adversarial inputs designed to bypass moderation can exploit the same GCG and AutoDAN techniques documented in [18, 19]. The original work [10], acknowledged that users employ symbol substitution, synonyms, and euphemisms to evade detection, and these evasion techniques continue to evolve. There is no protection against indirect prompt injection in moderated content (A2).

Recommended enhancements: Ensemble moderation combining LLM-based analysis with rule-based filters, adversarial robustness testing using established benchmarks [16, 19], and regular red-teaming against the moderation pipeline.

MCP Integration [9]. *Strengths:* The standardized protocol enables consistent security policy enforcement across all tool integrations. The client-server architecture provides a natural point for intercepting and validating tool interactions.

Weaknesses: No built-in tool verification mechanism (D1)—the protocol trusts server-provided tool definitions. The rug pull risk (D2) is inherent in the dynamic tool discovery mechanism. Bidirectional communication enables server-initiated attacks through the Sampling and Elicitation primitives, where a malicious server can manipulate the host's LLM or extract information from the user.

Recommended enhancements: Tool definition pinning with cryptographic signatures, sandboxed execution environments for tool operations, human-in-the-loop approval for sensitive operations, and an allowlisting framework for approved MCP servers [33].

5.3 Trade-offs and Practical Considerations

Implementing the proposed defense-in-depth framework involves significant trade-offs that SaaS architects must navigate.

Security versus performance. KV-cache isolation eliminates the PROMPTPEEK attack [13], but reduces inference throughput by an estimated 15–30% due to loss of cache-sharing benefits [14]. Separate vector namespaces per tenant increase storage requirements proportionally to the number of tenants. Input and output filtering add latency to every request.

Security versus cost. The most secure configuration is to have each tenant have their own separate instance of a model, which eliminates the majority of multi-tenancy amplification but will not make the SaaS business model economically viable to most providers. A shared infrastructure model with multiple layers of guardrails offers a pragmatic approach to finding a middle ground, as there will always be some level of residual risk, and provides economic viability to service providers.

Security versus functionality. If an input filter is too strict then it results in producing false positives that impact on the level of satisfaction of the user. Liu et al. [16], found that there exists no one defence mechanism which provides both high utility and high security at the same time i.e., there is an inherent trade-off between the two. The Moderation of User Content work [10], documented the challenges associated with achieving a balance of false positive rates versus the coverage of user detection especially when considering different Linguistic and Cultural environments.

Regulatory compliance. The EU AI Act [36], imposes transparency and safety obligations with penalties up to 7% of global turnover. NIST AI 600-1 [31], provides guidance for managing generative AI risks. SaaS providers serving regulated industries face compounding requirements from GDPR (data protection), sector-specific regulations (healthcare, finance), and emerging AI-specific regulation. These compliance requirements add urgency to adopting the defense framework pro-

posed in this paper, as regulatory penalties for cross-tenant data leakage may be assessed per affected tenant rather than per incident.

5.4 Limitations

This study has several limitations that should be acknowledged. First, the threat taxonomy is derived from systematic literature review and architectural analysis rather than large-scale empirical attack testing in production multi-tenant environments. This analytical approach is a deliberate methodological choice: foundational taxonomy work necessarily precedes targeted experimentation, and the individual vulnerability classes in our taxonomy are each grounded in peer-reviewed empirical evidence (e.g., PoisonedRAG [26] achieving 97% attack success, PROMPTPEEK [13] demonstrating cross-tenant prompt reconstruction, GCG [18] achieving 90–99% jailbreak success rates). However, the interaction effects between multiple simultaneous attacks in multi-tenant settings have not been empirically studied, and quantifying these compound effects in a controlled multi-tenant testbed represents the natural next step for this research.

Second, the mitigation effectiveness assessments are qualitative rather than quantitative. The proposed defense layers are grounded in published research (Llama Guard [34], SMTA/BAU [14], RevPRAG [29]) but their combined effectiveness in a fully integrated multi-tenant deployment requires empirical validation. Measuring detection and prevention rates for each layer individually and in combination would enable evidence-based security architecture decisions.

Third, the LLM security landscape evolves rapidly. New attack techniques and defense mechanisms emerge continuously [20], and the threat taxonomy presented here represents a snapshot as of early 2026. The framework is designed to be extensible, with new vulnerability classes addable within the existing categorical structure.

Fourth, this analysis focuses on text-based LLM interactions. The emerging field of multimodal LLM security, where attacks may leverage image, audio, or video inputs, is out of scope but represents an important future direction.

5.5 Future Work

Several directions emerge from this analysis. *Empirical validation* through prompt injection testing against a multi-tenant RAG system with metadata filtering would quantify the actual bypass rates and amplification factors. *Quantitative benchmarking* of the defense-in-depth framework's effectiveness, measuring the detection and prevention rates of each layer individually and in combination, would enable evidence-based security architecture decisions. *Extension to multimodal LLM security* is increasingly relevant as SaaS platforms incorporate vision and audio processing capabilities. *Automated security testing frameworks* specifically designed for LLM-powered SaaS would accelerate adoption of security best practices across the industry. Finally, *formal verification* of tenant isolation properties in context orchestration architectures could provide mathematical guarantees rather than heuristic assurances.

6. CONCLUSIONS

This paper has presented a systematic analysis of security challenges specific to multi-tenant LLM-powered SaaS platforms. The central finding is that multi-tenancy does not merely scale existing LLM security risks—it qualitatively transforms them. Shared inference infrastructure, shared retrieval stores, and shared tool ecosystems create cross-tenant attack pathways that have no equivalent in single-tenant deployments. Our amplification analysis confirmed that two-thirds of the identified vulnerability classes exhibit increased severity under multi-tenancy, with cross-tenant data leakage, RAG knowledge base poisoning, and MCP tool poisoning presenting the highest risk.

The proposed threat taxonomy and defense-in-depth framework offer two practical contributions for the SaaS industry. First, the taxonomy provides a structured vocabulary for security architects to reason about LLM-specific threats in the context of their deployment model, extending the OWASP Top 10 for LLM Applications [1] with three vulnerability classes unique to multi-tenant and protocol-level contexts. Second, the threat-to-mitigation matrix enables risk-prioritized implementation, allowing organizations to allocate security investment where multi-tenancy amplification is greatest rather than treating all threats equally.

A key implication of our analysis is that traditional SaaS security controls—network segmentation, database access control, and API authentication—remain necessary but are insufficient for LLM-integrated platforms. The probabilistic, context-driven nature of LLM behavior demands security mechanisms that operate at the semantic level: detecting adversarial content in retrieved documents, verifying that outputs do not leak cross-tenant data, and ensuring that tool invocations remain within authorized boundaries. Metadata filtering, as implemented in the COL architecture [7], provides an effective baseline for tenant isolation but should not be relied upon as a single point of defense without cryptographic hardening and complementary layers.

The regulatory landscape, including the EU AI Act [36], and NIST AI 600-1 [31], adds urgency to these technical measures. SaaS providers that fail to address LLM-specific security risks face not only technical exposure but potential regulatory penalties assessed on a per-tenant basis, amplifying the financial consequences of cross-tenant breaches.

Looking ahead, empirical validation of the proposed framework—measuring actual bypass rates, amplification factors, and defense-layer effectiveness in controlled multi-tenant testbeds—is the critical next step. We invite collaboration across the SaaS industry, security research community, and standards bodies to build on the taxonomy and mitigation framework presented here toward practical, validated security standards for LLM-integrated multi-tenant systems.

Conflict of Interest

The authors declare no conflict of interest.

References

[1] https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/.

[2] Yao Y, Duan J, Xu K, Cai Y, Sun Z, et al. A Survey on Large Language Model (Llm) Security and Privacy: The Good, the Bad, and the Ugly. High-Confidence Computing. 2024;4:100211.

[3] Li MQ, Fung BC. Security Concerns for Large Language Models: A Survey. J Inf Secur Appl. 2025;95:104284.

[4] Berini AD, Jamil N, Benrazek AE, Lakas A, Ismail L, et al. Security and Privacy in Llms: A Comprehensive Survey of Threats and Mitigation Strategies. Inf Fusion. 2026;132:104241.

[5] Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. Adv Neural Inf Process Syst. 2020;33:9459-9474.

[6] Liu NF, Lin K, Hewitt J, Paranjape A, Bevilacqua M, et al. Lost in the Middle: How Language Models Use Long Contexts. 2023. Arxiv preprint arxiv: https://arxiv.org/pdf/2307.03172.

[7] Romankiv N. Methodology of context engineering for increasing the relevance of AI agents in SaaS systems. In: Proc. Sci.-Practical Conf., Kharkiv. Ukraine; 2025 [online]. doi: 10.36074/grail-of-science.12.12.2025.092.

[8] https://modelcontextprotocol.io/.

[9] Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T, et al. Not What You've Signed up For: Compromising Real-World Llm-Integrated Applications With Indirect Prompt Injection. InProceedings of the 16th ACM workshop on artificial intelligence and security. 2023:79-90.

[10] http://www.wayscience.com/wp-content/uploads/2025/05/Conference-Proceedings-May-8-9-2025-1.pdf.

[11] Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T, et al. Not What You've Signed up For: Compromising Real-World LLM-Integrated Applications With Indirect Prompt Injection. InProceedings of the 16th ACM workshop on artificial intelligence and security. 2023:79-90.

[12] Gulyamov S, Gulyamov S, Rodionov A, Khursanov R, Mekhmonov K, et al. Prompt Injection Attacks in Large Language Models and AI Agent Systems: A Comprehensive Review of Vulnerabilities, Attack Vectors, and Defense Mechanisms. Information. 2026;17:54.

[13] Wu G, Zhang Z, Zhang Y, Wang W, Niu J, et al. I Know What You Asked: Prompt Leakage via Kv-Cache Sharing in Multi-Tenant LLM Serving. In Network and Distributed System Security (NDSS) Symposium. 2025.

[14] Zhang Q, Wang EE, Li J, Wang X. Burn-after-use for preventing data leakage through a secure multi-tenant architecture in enterprise LLM. Arxiv preprint arxiv: https://arxiv.org/pdf/2601.06627

[15] Cherednichenko O, Sytnikov D, Romankiv N, Sharonova N, Sytnikova P. AI agent for conversational QA over SaaS Codebase using large language models. ISW-2025: Intelligent Systems Workshop at 9th International Conference on Computational Linguistics and Intelligent Systems (CoLInS-2025). 2025

[16] Liu Y, Jia Y, Geng R, Jia J, Gong NZ. Formalizing and Benchmarking Prompt Injection Attacks and Defenses. In33rd USENIX Security Symposium. USENIX Security 24. 2024:1831-1847.

[17] Liu Y, Deng G, Li Y, Wang K, Wang Z, et al. Prompt Injection Attack Against Llm-Integrated Applications. 2023. Arxiv preprint: https://arxiv.org/pdf/2306.05499v1.

[18] Zou A, Wang Z, Carlini N, Nasr M, Kolter JZ, et al. Universal and transferable adversarial attacks on aligned language models. 2023. Arxiv preprint: https://arxiv.org/pdf/2307.15043.

[19] Xu Z, Liu F, Liu H. Bag of Tricks: Benchmarking of Jailbreak Attacks On LLMs. 2024. Arxiv preprint arxiv:https://arxiv.org/pdf/2406.09324.

[20] Chu J, Liu Y, Yang Z, Shen X, Backes M, et al. Jailbreakradar: Comprehensive Assessment of Jailbreak Attacks Against LLMs. InProceedings of the 63rd Annual Meeting of the Association for Computational Linguistics. 2025;1:21538-21566.

[21] Carlini N, Tramer F, Wallace E, Jagielski M, Herbert-Voss A, et al. Extracting Training Data From Large Language Models. In30th USENIX security symposium. USENIX Security 21. 2021:2633-2650.

[22] Li Q, Hong J, Xie C, Tan J, Xin R, et al. LLM-PBE: assessing data privacy in large language models. Proceedings of the VLDB Endowment. 2024;17:3201-3214.

[23] Yan B, Li K, Xu M, Dong Y, Zhang Y, et al. On Protecting the Data Privacy of Large Language Models (LLMs) and Llm Agents: A Literature Review. High-Confidence Computing. 2025;5:100300.

[24] Cheng S, Li Z, Meng S, Ren M, Xu H, et al. Understanding Pii Leakage in Large Language Models: A Systematic Survey. In: Proceedings of the 34th International Joint Conference on Artificial Intelligence IJCAI. 2025:10409-10417.

[25] Asthana S, Zhang B, Mahindru R, DeLuca C, Gentile AL, et al. Deploying Privacy Guardrails for LLMs: A Comparative Analysis of Real-World Applications. 2025. Arxiv preprint arxiv: https://arxiv.org/pdf/2501.12456.

[26] Zou W, Geng R, Wang B, Jia J. Poisonedrag: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models. In34th USENIX Security Symposium. USENIX Security . 2025:3827-3844.

[27] Shafran A, Schuster R, Shmatikov V. Machine Against the Rag:Jamming Retrieval-Augmented Generation With Blocker Documents. In34th USENIX Security Symposium. 2025:3787-3806.

[28] Arzanipour A, Behnia R, Ebrahimi R, Dutta K. RAG security and privacy: formalizing the threat model and attack surface. Arxiv preprint arxiv: https://arxiv.org/pdf/2509.20324.

[29] Tan X, Luan H, Luo M, Sun X, Chen P, et al. Revprag: Revealing poisoning attacks in retrieval-augmented generation through llm activation analysis. ACL. 2025:12999–13011.

[30] Morris J, Kuleshov V, Shmatikov V, Rush A. Text Embeddings Reveal (Almost) as Much as Text. InProceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. 2023:12448-12460.

[31] https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[32] Ferrag MA, Tihanyi N, Hamouda D, Maglaras L, Lakas A, et al. From Prompt Injections to Protocol Exploits: Threats in Llm-Powered AI Agents Workflows. ICT Express. 2026;12:353-383.

[33] Errico H, Ngiam J, Sojan S. Securing the Model Context Protocol (Mcp): Risks Controls and Governance. 2025. Arxiv preprint arxiv: https://arxiv.org/pdf/2511.20920.

[34] Inan H, Upasani K, Chi J, Rungta R, Iyer K, et al. Llama Guard: Llm-Based Input-Output Safe-Guard for Human-Ai Conversations. 2023. Arxiv preprint arxiv: https://arxiv.org/pdf/2312.06674.

[35] Mehta SK. Guardrailed Llms: Red Teaming and Safety Mitigations. Int. J. Res. Appl. Innov. 2024;7:10408-10410.

[36] https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng.