LLM-Enabled Open-Source Systems in the Wild: An Empirical Study of Vulnerabilities in GitHub Security Advisories

Fariha Tanjim Shifat
Missouri University of Science and Technology
Rolla, Missouri, USA
fsvfh@mst.edu

Hariswar Baburaj
Missouri University of Science and Technology
Rolla, Missouri, USA
hbkkb@mst.edu

Ce Zhou
Missouri University of Science and Technology
Rolla, Missouri, USA
cezhou@mst.edu

Jaydeb Sarker
University of Nebraska Omaha
Omaha, Nebraska, USA
jsarker@unomaha.edu

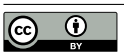
Mia Mohammad Imran
Missouri University of Science and Technology
Rolla, Missouri, USA
imranm@mst.edu

Abstract

Large language models (LLMs) are increasingly embedded in open-source software (OSS) ecosystems, creating complex interactions among natural language prompts, probabilistic model outputs, and execution-capable components. However, it remains unclear whether traditional vulnerability disclosure frameworks adequately capture these model-mediated risks. To investigate this, we analyze 295 GitHub Security Advisories published between January 2025 and January 2026 that reference LLM-related components, and we manually annotate a sample of 100 advisories using the OWASP Top 10 for LLM Applications 2025.

We find no evidence of new implementation-level weakness classes specific to LLM systems. Most advisories map to established CWEs, particularly injection and deserialization weaknesses. At the same time, the OWASP-based analysis reveals recurring architectural risk patterns, especially Supply Chain, Excessive Agency, and Prompt Injection, which often co-occur across multiple stages of execution. These results suggest that existing advisory metadata captures code-level defects but underrepresents model-mediated exposure. We conclude that combining the CWE and OWASP perspectives provides a more complete and necessary view of vulnerabilities in LLM-integrated systems.

ACM Reference Format:
Fariha Tanjim Shifat, Hariswar Baburaj, Ce Zhou, Jaydeb Sarker, and Mia Mohammad Imran. 2026. LLM-Enabled Open-Source Systems in the Wild: An Empirical Study of Vulnerabilities in GitHub Security Advisories. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3803437.3805532



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805532

1 Introduction

Large language models (LLMs) are increasingly embedded in modern software systems as development assistants [25], runtime components [38], and autonomous agents [40]. In open-source software (OSS), this integration appears in code generation, dependency management, automated review, conversational interfaces, retrieval-augmented generation (RAG), model context protocol (MCP) pipelines, and tool orchestration frameworks [24].

Unlike conventional software components that operate within fixed input-output structures, LLM-based systems process natural language prompts and produce probabilistic outputs. These outputs can trigger downstream actions such as file system operations, command execution, database queries, or external API calls [35, 41]. This coupling between non-deterministic model behavior and execution-capable components introduces additional abstraction layers between user input and system behavior, making system behavior more interaction-driven and harder to reason about statically.

As LLM capabilities are packaged into libraries, inference engines, and agent frameworks distributed through ecosystems such as PyPI and npm, vulnerabilities in these components propagate through the software supply chain [24, 26, 39]. At the same time, LLM-integrated systems introduce additional exposure pathways through dynamic prompt construction, probabilistic output handling, and autonomous tool invocation [18]. These mechanisms do not map cleanly to conventional package-level risk.

GitHub Security Advisories (GHSAs) document vulnerabilities through structured metadata such as affected packages, version ranges, severity ratings, and Common Weakness Enumeration (CWE) identifiers [20, 28]. CWE classifications capture implementation-level weaknesses such as improper input neutralization, unsafe deserialization, command injection, and uncontrolled resource consumption [31]. However, they do not indicate whether a weakness is triggered, amplified, or propagated through model reasoning, prompt manipulation, or agent autonomy. This limits their ability to represent how vulnerabilities manifest in LLM-integrated systems.

Figure 1 illustrates this limitation. The advisory is correctly classified as CWE-94 (improper control of code generation), which captures the code-level defect. However, the exploitation pathway

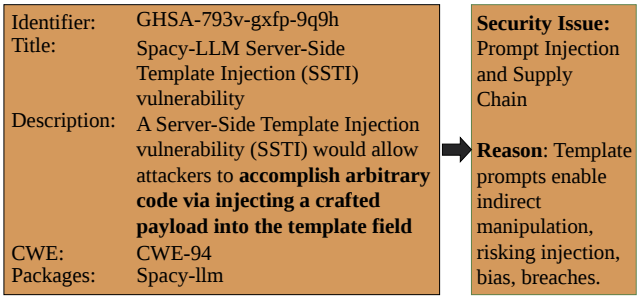


Figure 1: Illustration of model-mediated exposure in an LLM-associated advisory.

operates through template-based prompts and model-generated content. These indirect manipulation channels are not represented by the CWE label. The vulnerability, therefore, exists at two levels: the implementation defect captured by CWE and the interaction pattern through which model outputs reach executable behavior. The GHSA schema captures only the implementation level defects, leaving the architectural risk entirely unrepresented.

The OWASP Top 10 for LLM Applications 2025 (Table 1) provides a complementary perspective [33]. It defines categories such as Prompt Injection, Improper Output Handling, Excessive Agency, Data and Model Poisoning, and Unbounded Consumption. These categories capture interaction-level risks. CWE explains how code fails, whereas OWASP captures how model-enabled systems become exposed at the architectural level.

Despite the growing prevalence of LLM-integrated software, empirical evidence on how LLM-related vulnerabilities appear in open-source advisories remains limited. It remains unclear how often LLM-associated packages appear in disclosures, whether CWE classifications adequately reflect model-mediated exposure, and how advisory data aligns with LLM-specific risk taxonomies. This analysis is further complicated by the fact that current advisory schemas do not include structured indicators of LLM involvement.

To address this gap, we conduct an empirical study of 295 GitHub Security Advisories published between January 2025 and January 2026 that explicitly reference LLM-related components. From this dataset, we manually categorize the 133 unique affected packages into three groups: *LLM-associated*, *Possible LLM-associated*, and *Non-LLM-associated*. The first group includes packages that directly implement or orchestrate LLM functionality, such as inference, prompt management, and agent frameworks. The second includes packages commonly used within LLM pipelines without being LLM-specific by design. The third includes packages whose functionality is unrelated to LLM systems. We then randomly sample 100 advisories from the first two categories and manually annotate them using the OWASP Top 10 for LLM Applications 2025 [33] to examine architectural exposure patterns beyond implementation-level defects. Specifically, we ask the following RQs:

RQ1: Which CWE categories most commonly occur in GitHub Security Advisories related to LLM-associated packages?

→ Code injection (*CWE-94*), command injection (*CWE-77*, *CWE-78*), and unsafe deserialization (*CWE-502*) dominate, indicating that LLM ecosystems primarily inherit established weakness classes.

RQ2: How effectively do existing advisory metadata fields represent LLM involvement and model-mediated exposure mechanisms?

→ Current GHSA metadata lacks structured indicators of LLM involvement, requiring manual classification to identify model-mediated exposure patterns.

RQ3: What exposure patterns emerge when LLM-related advisories are mapped using the OWASP Top 10 for LLM Applications 2025?

→ Supply Chain Risks (44%), Excessive Agency (20%), and Prompt Injection (18%) dominate. Thirty-seven percent of advisories exhibit multi-label patterns that combine prompt manipulation, output handling weaknesses, and execution authority.

RQ4: How do OWASP LLM risk categories correspond to underlying CWE weakness classes?

→ Architectural LLM risk categories consistently materialize through conventional implementation weaknesses. Supply Chain shows wide CWE diversity, whereas Excessive Agency and Prompt Injection concentrate on injection-related flaws.

We have following three contributions from this study:

(1) an empirical characterization of CWE patterns in LLM-associated GitHub Security Advisories, based on 295 disclosures from January 2025 to January 2026;
(2) a systematic mapping between CWE and the OWASP Top 10 for LLM Applications 2025 through manual annotation of 100 advisories; and,
(3) evidence that current advisory metadata lacks explicit indicators of LLM involvement, which limits systematic analysis of model-mediated exposure.

The replication package, including annotation guidelines and the annotated dataset, is publicly available at [37]. The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the methodology. Section 4 presents the results. Section 5 discusses the implications. Section 6 outlines the limitations. Section 7 concludes the paper.

2 Background and Related Work

Our study draws on two related areas: open source software supply chain security and security risks in LLM-enabled systems.

2.1 Open Source Supply Chain Security

Modern software development relies heavily on third-party open source packages distributed through registries such as npm, PyPI, and Maven. When developers install a package, they also introduce its transitive dependencies, which expands the attack surface. As a result, a single malicious or vulnerable package can propagate risk across thousands of downstream projects [32, 42].

Prior work has examined the scale and dynamics of these risks across ecosystems. Alfadel *et* al. [14] analyzed 1,396 vulnerability reports affecting 698 Python packages and found that vulnerabilities often take more than three years to surface, while more than half remain unfixed at the time of public disclosure. Zimmermann *et* al. [42] showed that the npm ecosystem has a "small world with high risks" topology, in which a small number of highly connected maintainer accounts can affect large portions of the registry. Decan *et* al. [17] examined how vulnerabilities propagate through npm dependency networks and highlighted the cascading nature of supply chain exposure. Guo *et* al. [22] studied malicious code in the

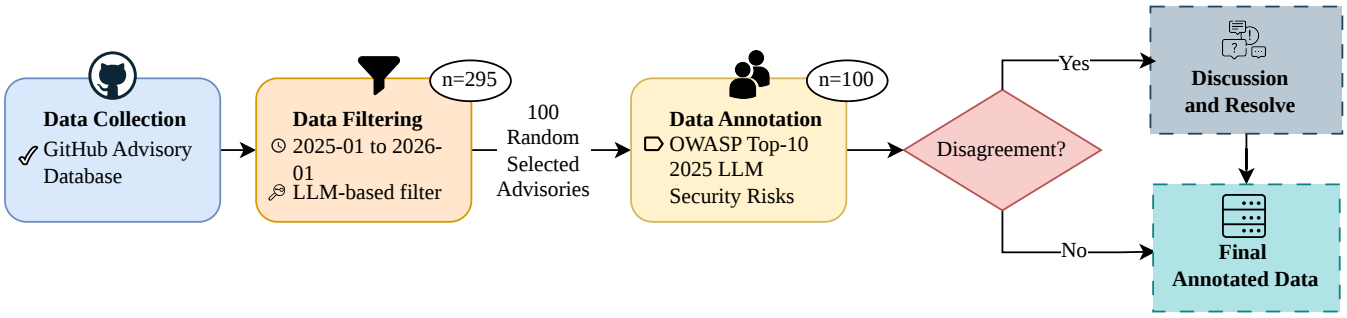


Figure 2: Overview of the data collection, filtering, sampling, and annotation pipeline.

Table 1: OWASP TOP 10 LLM Risk Categories in 2025.

ID	Category Name	Definition
LLM01	Prompt Injection	Inputs manipulate LLM behavior or outputs beyond intended controls.
LLM02	Sensitive Information Disclosure	Unintended exposure of confidential, personal, or proprietary data.
LLM03	Supply Chain	Risks from compromised models, data, tools, or dependencies in the LLM lifecycle.
LLM04	Data and Model Poisoning	Malicious manipulation of training or embedding data to alter model behavior.
LLM05	Improper Output Handling	Unsafe use of LLM outputs without validation or sanitization.
LLM06	Excessive Agency	LLMs granted excessive autonomy or permissions causing harmful actions.
LLM07	System Prompt Leakage	Exposure of system prompts containing sensitive instructions or data.
LLM08	Vector and Embedding Weaknesses	Flaws in vector or embedding handling enabling data leakage or manipulation.
LLM09	Misinformation	Generation of incorrect or misleading information that appears credible.
LLM10	Unbounded Consumption	Uncontrolled LLM usage leading to resource exhaustion or financial loss.

PyPI ecosystem and identified thousands of malicious packages that relied on typosquatting and related social engineering techniques.

To support the management of these threats, the GHSA database provides structured vulnerability metadata, including affected packages, version ranges, and CWE identifiers [19]. However, Ayala *et* al. [36] showed that structural latencies in the GHSA review pipeline can delay the availability of actionable security information. This means that metadata alone may leave temporal windows of exposure. Complementary code-centric approaches, such as the detection and mitigation frameworks proposed by Ponta *et* al. [34], therefore remain important for securing open source dependencies.

2.2 LLM Supply Chain and Security Risks

The rapid adoption of LLMs has expanded the notion of software supply chains beyond conventional package dependencies to include pre-trained models, training datasets, fine-tuning pipelines, prompt templates, and orchestration frameworks [39]. In response, recent work has proposed broader research agendas [39] and taxonomies of LLM-specific threats, including malicious package injection, model backdoors, and prompt leakage [27].

Researchers have also shown that LLM interfaces can be actively exploited through prompt injection. These attacks have been demonstrated empirically and formalized in attack frameworks [21, 29, 30]. More recent work extends these concerns to autonomous agent ecosystems and highlights severe protocol-layer vulnerabilities [18].

Despite this growing body of research, little work has systematically mapped GitHub Security Advisories involving LLM-associated packages to LLM security risks. Our study addresses this gap by connecting implementation-level weakness reporting with architectural exposure analysis.

3 Methodology

Figure 2 presents the overall methodology of this study. In this section, we describe the procedures used to collect advisory data, filter for LLM-associated packages, and conduct manual annotation.

3.1 Data Collection and Filtering

We collected publicly available security advisories from the GitHub Advisory Database (GHSA) published between January 2025 and January 2026 [19]. This period provides a recent and consistent snapshot of disclosures during rapid LLM ecosystem integration. GHSA standardizes records across major open-source ecosystems on GitHub, including projects that distribute LLM frameworks, inference engines, and orchestration libraries. It therefore offers direct visibility into supply chain risks affecting AI-enabled software on GitHub. Prior work shows that GHSA is effective for characterizing supply chain vulnerabilities [36], particularly those driven by dependency structures in open-source software stacks [28].

GHSA metadata includes affected packages, ecosystems, severity ratings, version ranges, CVE identifiers, CWE classifications, and vulnerability descriptions. For each advisory, we extracted the GHSA identifier, description, affected ecosystem and package

names, severity level, CVE identifier when available, and associated CWE categories.

To identify LLM-relevant cases, we applied keyword-based filtering to advisory metadata and descriptions. The keyword set included generic LLM terms (e.g., llm, gpt, embeddings), model vendors and families (e.g., openai, llama, mistral, anthropic, claude), orchestration and agent frameworks (e.g., langchain, flowise, ollama, vllm, huggingface), protocol-related terms (e.g., mcp, model context protocol), and retrieval or embedding infrastructure (e.g., vector databases, chroma, milvus). This process yielded 295 advisories referencing LLM-related components.

3.2 LLM-associated Package Filtering

Keyword matches alone do not guarantee that an affected package directly implements LLM functionality. In several cases, advisories referenced LLM concepts in their descriptions while affecting general-purpose libraries or supporting infrastructure. To refine the dataset, we extracted all affected package names from the 295 disclosures, resulting in 133 unique packages. We then manually reviewed each package using its repository documentation, project description, and stated functionality.

We grouped the packages into three categories using thematic analysis [15]: *LLM-associated*, *Possible LLM-associated*, and *Non-LLM-associated*. One author performed the primary classification based on package names, descriptions, and documentation. A second author independently reviewed the assignments. Disagreements were resolved through discussion. Section 4.2 provides the detailed definitions of each category.

The final distribution included 84 *LLM-associated* packages affecting 226 advisories, 16 *Possible LLM-associated* packages affecting 34 advisories, and 33 *Non-LLM-associated* packages affecting 35 advisories. We then randomly sampled 100 advisories from the 260 associated with the first two categories, yielding 92 *LLM-associated* and 8 *Possible LLM-associated* advisories for detailed examination.

3.3 Data Annotation

Two authors independently annotated the randomly selected 100 advisories. We developed a detailed annotation guideline based on the OWASP Top 10 for LLM Applications 2025 taxonomy [33]. This taxonomy includes Prompt Injection, Sensitive Information Disclosure, Supply Chain, Data and Model Poisoning, Improper Output Handling, Excessive Agency, System Prompt Leakage, Vector and Embedding Weaknesses, Misinformation, and Unbounded Consumption. Table 1 summarizes these definitions. The full annotation instructions are available in the replication package [37].

Two annotators independently reviewed all advisories. For each case, they analyzed the advisory content, examined the technical role of the affected package, and assessed the exploitation scenario. They then identified the dominant exposure mechanism and mapped the advisory to one or more OWASP categories. When an advisory did not correspond to any of the ten predefined categories, it was labeled as miscellaneous.

We measured inter-annotator agreement before resolving discrepancies. Cohen's Kappa [16] was 0.76 and Gwet's AC1 [23] was 0.95, indicating substantial agreement and high reliability. The annotators then reconciled differences through discussion and finalized the labels used in the subsequent analyses.

4 Evaluation and Results

To systematically characterize LLM-related vulnerabilities within the GitHub Security Advisory ecosystem, we organize our empirical findings around our four research questions.

4.1 RQ1: Which CWE categories most commonly occur in GitHub Security Advisories related to LLM-focused packages?

We address RQ1 by analyzing CWE distributions in both the full 295-advisory dataset and the 100-advisory annotated subset. Across the full dataset, we observe 99 distinct CWE identifiers. Table 2 reports the ten most frequent CWE categories. *CWE-94* appears most often, with 24 occurrences, followed by *CWE-77* and *CWE-502*, with 22 occurrences each.

These CWEs fall into three broad weakness types. First, injection-related flaws form the largest group. This group includes code injection (*CWE-94*), command injection (*CWE-77*, *CWE-78*), SQL injection (*CWE-89*), and cross-site scripting (*CWE-79*). Second, unsafe handling of untrusted data appears through deserialization (*CWE-502*) and path traversal (*CWE-22*). Third, resource and complexity management weaknesses, including inefficient regular expression evaluation (*CWE-1333*), uncontrolled resource allocation (*CWE-770*), and uncontrolled resource consumption (*CWE-400*), expose systems to denial-of-service conditions.

Several advisories illustrate how these weaknesses appear in LLM-enabled systems. `GHSA-793v-gxfp-9q9h` (Figure 1) describes a server-side template injection in spacy-llm that enables arbitrary code execution, corresponding to *CWE-94* [5]. `GHSA-mrw7-hf4f-83pf` reports unsafe deserialization in vLLM [12], consistent with *CWE-502*.

The 100-advisory subset shows the same overall pattern. Injection-related weaknesses remain dominant, with *CWE-77* appearing 11 times, *CWE-78* and *CWE-502* appearing 9 times each, and *CWE-94* appearing 8 times. Authentication and access control weaknesses, particularly *CWE-306*, appear more prominently in this subset. No distinct LLM-specific implementation weakness emerges.

Overall, the vulnerability patterns align with conventional software security trends. The data suggests continuity rather than divergence, with established weakness classes persisting in LLM-enabled architectures.

4.2 RQ2: How effectively do existing advisory metadata fields represent LLM involvement and model-mediated exposure mechanisms?

We investigate RQ2 through two lenses: the structured metadata in the 295 filtered advisories and the results of manual package categorization. Table 3 summarizes ecosystem and severity distributions, and Table 4 presents package-level information.

Table 2: Top 10 CWEs in LLM-Referenced Advisories.

CWE	Description	Count	Example
CWE-94	Improper Control of Generation of Code (Code Injection)	24	GHSA-793v-gxfp-9q9h [5]
CWE-502	Deserialization of Untrusted Data	22	GHSA-mrw7-hf4f-83pf [12]
CWE-77	Improper Neutralization of Special Elements used in a Command (Command Injection)	22	GHSA-xq4m-mc3c-vvg3 [13]
CWE-78	Improper Neutralization of Special Elements used in an OS Command (OS Command Injection)	19	GHSA-2vv2-3x8x-4gv7 [1]
CWE-79	Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)	19	GHSA-hfcf-79gh-f3jc [9]
CWE-22	Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)	18	GHSA-j9g7-mqhh-9hxf [10]
CWE-1333	Inefficient Regular Expression Complexity	14	GHSA-8r9q-7v3j-jr4g [7]
CWE-770	Allocation of Resources Without Limits or Throttling	12	GHSA-hf3c-wxg2-49q9 [8]
CWE-400	Uncontrolled Resource Consumption	11	GHSA-6fvq-23cw-5628 [4]
CWE-89	Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)	11	GHSA-jmgm-gx32-vp4w [11]

The advisory schema includes ecosystem registry, severity level, fix status, and affected package name. PyPI contributes 162 advisories, npm 96, Go 22, Packagist 10, and crates.io and Maven 3 each. For severity, PyPI includes 31 Critical, 69 High, 53 Moderate, and 9 Low cases. npm includes 24 Critical, 50 High, 18 Moderate, and 4 Low. Go includes 3 Critical, 11 High, 6 Moderate, and 2 Low. Packagist includes 1 Critical, 2 High, and 7 Moderate. crates.io includes 3 Low. Maven includes 1 High and 2 Moderate.

These fields support cross-ecosystem and severity-level comparisons. However, they do not indicate whether a vulnerability arises in an LLM-enabled component. The metadata does not distinguish conventional software libraries from packages that implement model inference, prompt handling, or agent pipelines.

To evaluate the relevance of LLMs, we conducted a thorough review of all 133 unique packages in the dataset. This involved analyzing repository documentation, project descriptions, API references, and example usage to determine whether LLM functionality was a core feature, a supporting integration, or unrelated to LLM systems. Based on our analysis, we classified the packages into three categories:

- *LLM-associated*: Packages whose primary purpose is to implement or orchestrate LLM functionality, including model inference engines, prompt management libraries, agent frameworks, and retrieval-augmented generation pipelines. If removing LLM support eliminates the package's core purpose or renders it non-functional, we classify it as LLM-associated.
- *Possible LLM-associated*: Packages that do not inherently implement LLM functionality but serve as common supporting infrastructure in LLM-based systems, such as vector databases, workflow orchestration tools, and data processing utilities. If the package remains fully functional outside LLM contexts but frequently appears in LLM pipelines, we classify it as Possible LLM-associated.
- *Non LLM-associated*: Packages whose core functionality is unrelated to LLM systems. If the package's main purpose is general software functionality and its connection to LLM-enabled systems is weak, indirect, or non-essential, we classify it as Non-LLM-associated.

Table 3: Distribution of security issues per ecosystem by severity and fix status (at the time of data collection).

Ecosystem	Severity				Resolved	
	Critical	High	Moderate	Low	Yes	No
PyPI	31	69	53	9	132	30
npm	24	50	18	4	78	18
Go	3	11	6	2	12	10
Packagist	1	2	7	0	9	1
crates.io	0	0	0	3	3	0
Maven	0	1	2	0	3	0

Table 4: Package categorization and advisory distribution across 295 advisories

Category	Packages	Advisories	Examples
LLM-associated	84	226	vllm, ollama, langchain, litellm
Possible LLM-associated	16	34	keras, milvus, weaviate, n8n
Non-LLM-associated	33	35	wasmtime, orejime, directus

Table 4 shows the distribution of advisories in each package categories. 84 packages directly implement or orchestrate LLM functionality and account for 226 advisories. Sixteen packages are *possible LLM-associated* and account for 34 advisories. Thirty-three packages are *non-LLM-associated* and account for 35 advisories.

Overall, the existing advisory metadata captures ecosystem distribution and severity levels, but it does not structurally encode generative AI involvement or model-mediated exposure mechanisms.

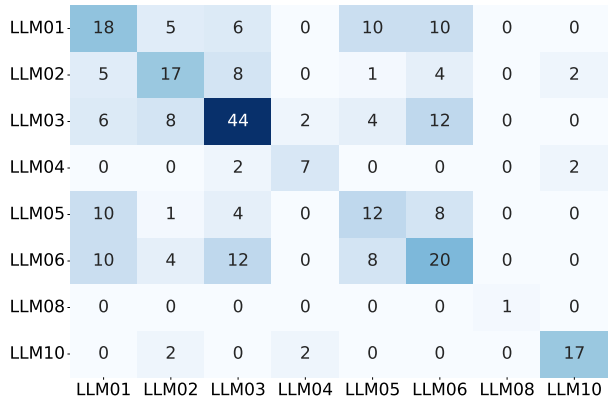


Figure 3: Co-occurrence of OWASP LLM Risk Categories.

4.3 RQ3: What exposure patterns emerge when LLM-related advisories are mapped using the OWASP Top 10 for LLM Applications 2025?

Because advisory metadata does not indicate whether a vulnerability involves LLM functionality, we applied the OWASP Top 10 for LLM Applications 2025 taxonomy [33] to 100 randomly sampled and manually annotated advisories. As described in Section 3, we mapped each advisory to one or more OWASP categories. This analysis reveals exposure patterns that are not captured by the standard GHSA advisory schema.

LLM03 (Supply Chain) is the most frequent category, appearing in 44 advisories. This suggests that exposure is concentrated in dependencies, plugins, orchestration layers, and surrounding infrastructure rather than in model internals alone. For example, `GHSA-793v-gxfp-9q9h` shows how a weakness in an ecosystem component becomes exploitable within an LLM-driven workflow [5].

LLM06 (Excessive Agency) appears in 20 advisories and captures architectures in which LLM-mediated components can trigger execution paths or invoke privileged actions. In `GHSA-3ch2-jxxc-v4xf`, model-influenced tool input reaches an execution interface and enables over-privileged integration [2]. *LLM01* (Prompt Injection) appears in 18 advisories and often serves as the initiating condition for downstream impact. *LLM02* (Sensitive Information Disclosure) and *LLM10* (Unbounded Consumption) each appear in 17 advisories, reflecting confidentiality exposure and resource abuse, respectively. *LLM05* (Improper Output Handling) appears in 12 advisories and captures unsafe consumption of model outputs. *LLM04* (Data and Model Poisoning) appears in 7 advisories and primarily affects retrieval-related components. *LLM08* (Vector and Embedding Weaknesses) appears once. No advisories in our sample map to *LLM07* (System Prompt Leakage) or *LLM09* (Misinformation), which may reflect reporting limitations in sampled dataset rather than the absence of these risks.

As shown in Figure 3, these risks frequently appear as multi-step combinations rather than isolated categories. Sixty-three advisories carry a single label, whereas 37 carry multiple labels. The pairing *LLM03+LLM06* appears 12 times, indicating that integration weaknesses become more severe when systems expose execution authority. For example, `GHSA-7944-7c6r-55vv` shows how compromise in an orchestration component can propagate into a server-side execution path once the system provides privileged interfaces [6].

Figure 3 also highlights recurring prompt-driven combinations. *LLM01+LLM05* and *LLM01+LLM06* each appear 10 times, and *LLM01+LLM05+LLM06* appears 7 times. `GHSA-3ch2-jxxc-v4xf` illustrates this sequence: attacker-controlled prompt input influences tool arguments, weak output handling fails to constrain the response, and the system forwards those parameters to an execution endpoint [2]. Similarly, `GHSA-6f6r-m9pv-67jw` reflects the same prompt-to-tool-to-command pathway in a separate MCP deployment [3].

Overall, these results show that LLM-related exposure often follows recurring architectural chains rather than isolated failures. Prompt processing, output handling, and execution authority interact in ways that create predictable exploitation pathways. In our dataset, the dominant risks arise from cross-component interaction more often than from model internals alone.

4.4 RQ4: How do OWASP LLM risk categories correspond to underlying CWE weakness classes?

RQ3 identifies the dominant exposure patterns at the OWASP category level. RQ4 examines how these architectural risk categories correspond to implementation-level CWE weakness classes. We therefore analyze the mapping between OWASP LLM categories and CWE identifiers within the same 100 annotated advisories.

The annotated dataset contains 55 unique CWE identifiers. Eight of the ten OWASP LLM categories appear in the sample. *LLM07* and *LLM09* do not appear, consistent with the distribution reported in RQ3. Table 5 presents the OWASP-CWE mappings and their frequencies.

LLM03 spans 37 distinct OWASP-CWE co-occurrence pairs across 44 advisories, making it the broadest category. The most frequent mapping, *LLM03-CWE-78* (9), indicates OS command injection introduced through third-party LLM integrations. *LLM03-CWE-306* (6) reflects missing authentication in exposed LLM-connected services. Additional recurring mappings, including *LLM03-CWE-94*, *LLM03-CWE-77*, and *LLM03-CWE-502* (5 each), show that supply-chain risks commonly manifest through injection and deserialization weaknesses.

LLM06 primarily corresponds to execution-related CWEs. *LLM06-CWE-77* (8) and *LLM06-CWE-78* (5) indicate command injection enabled by over-privileged agent behavior. *LLM06-CWE-94* (3) reflects unsafe dynamic code execution. *LLM01* maps mainly to injection-related weaknesses, including *LLM01-CWE-77* (5) and *LLM01-CWE-79* (4), showing how prompt manipulation propagates into command or script injection contexts.

LLM05 also aligns with execution-related weaknesses, particularly *LLM05-CWE-77* (4) and *LLM05-CWE-78* (3), indicating unsafe consumption of model outputs. *LLM10* corresponds primarily to resource exhaustion, dominated by *LLM10-CWE-770* (5). *LLM04* most frequently maps to *LLM04-CWE-502* (4), reflecting deserialization risks.

Table 5: LLM-CWE Pair Frequencies in the Annotated Sample (Count ≥ 4).

LLM	CWE	Count
LLM03 (Supply Chain)	CWE-78 (OS Command Injection)	9
LLM06 (Excessive Agency)	CWE-77 (Command Injection)	8
LLM03 (Supply Chain)	CWE-306 (Missing Authentication for Critical Function)	6
LLM03 (Supply Chain)	CWE-94 (Code Injection)	5
LLM03 (Supply Chain)	CWE-77 (Command Injection)	5
LLM03 (Supply Chain)	CWE-502 (Deserialization of Untrusted Data)	5
LLM06 (Excessive Agency)	CWE-78 (OS Command Injection)	5
LLM01 (Prompt Injection)	CWE-77 (Command Injection)	5
LLM10 (Unbounded Consumption)	CWE-770 (Allocation of Resources Without Limits)	5
LLM01 (Prompt Injection)	CWE-79 (Cross-Site Scripting)	4
LLM05 (Improper Output Handling)	CWE-77 (Command Injection)	4
LLM04 (Data and Model Poisoning)	CWE-502 (Deserialization of Untrusted Data)	4
LLM03 (Supply Chain)	CWE-89 (SQL Injection)	4

Overall, the analysis shows that LLM integration changes how traditional weaknesses interact and propagate across system components. The underlying implementation flaws remain conventional, but architectural coupling shapes their exposure and impact.

5 Implications

Our findings suggest three main implications for understanding vulnerabilities in LLM-associated software.

Implementation-Level Weaknesses Remain Central. The results suggest that vulnerabilities in LLM-integrated open-source systems should be analyzed at multiple levels. At the implementation level, the dominant defects remain familiar. The most common advisories map to established CWE families such as injection, deserialization, authentication, and resource management. This continuity indicates that existing code-level security analysis and mitigation techniques remain relevant in LLM-integrated systems. However, CWE labels alone do not explain how these weaknesses become exposed within model-mediated workflows or how they participate in broader interaction sequences.

Architectural Exposure Shapes Risk. The OWASP-based annotation further indicates that risk in LLM-integrated systems is often shaped by system architecture rather than by model internals alone. The most frequent patterns involve Supply Chain, Excessive Agency, and Prompt Injection, and many advisories exhibit combinations of these categories. This suggests that vulnerabilities often propagate across prompt processing, output handling, orchestration, and execution layers. In this setting, the security question is not only which weakness exists, but also how system design allows that weakness to travel across components and produce downstream effects.

Advisory Metadata Captures Only Part of the Exposure Path. The results also indicate a limitation in current advisory reporting practices. GHSA metadata records affected packages, severity, and implementation-level weakness classes, but it does not explicitly encode whether a vulnerability involves LLM functionality or model-mediated interaction. As a result, advisory records capture the code-level defect while often omitting the architectural pathway through which the defect becomes exploitable in an LLM-integrated system. This gap suggests that CWE and OWASP provide complementary views of risk: one captures the underlying weakness, while the other captures the interaction pattern through which that weakness is exposed.

6 Limitations

Our study has several limitations. We restrict the dataset to GitHub Security Advisories published within a defined time window, excluding vulnerabilities disclosed through other channels such as vendor advisories or independent CVE entries. We identify LLM-associated cases through keyword-based filtering, which may omit relevant advisories that do not explicitly reference LLM terminology and may include indirectly related cases. We also rely on manual annotation for OWASP category mapping. Although inter-annotator agreement is substantial, these decisions still involve interpretive judgment. Finally, we analyze disclosed vulnerabilities only and therefore do not account for undiscovered or unreported weaknesses.

7 Conclusion and Future Work

We empirically analyzed 295 GitHub Security Advisories referencing LLM-related components and manually annotated 100 using the OWASP Top 10 for LLM Applications 2025. Our findings show that LLM-associated packages inherit conventional weakness classes rather than introducing new ones, while existing advisory metadata lacks structured indicators of model-mediated exposure. Taken together, CWE and OWASP perspectives are complementary: neither alone is sufficient to characterize the full risk profile of LLM-integrated systems.

In future work, we will extend the dataset beyond the current one-year window and incorporate additional disclosure sources, including CVE, NVD, and ecosystem-specific advisory databases. We will also develop automated methods for identifying LLM involvement and model-mediated exposure in advisory text using the annotated dataset as ground truth. In addition, we will examine multi-stage architectural interaction pathways more systematically, especially cases where model-generated outputs propagate to downstream components that execute commands or access resources. Finally, we will conduct longitudinal and comparative analyses of how LLM-associated vulnerabilities differ from non-LLM cases in

disclosure patterns, remediation timelines, and ecosystem distribution.

References

[1] 2026. GHSA-2vv2-3x8x-4gv7. https://github.com/advisories/GHSA-2vv2-3x8x-4gv7.

[2] 2026. GHSA-3ch2-jxxc-v4xf. https://github.com/advisories/GHSA-3ch2-jxxc-v4xf.

[3] 2026. GHSA-6f6r-m9pv-67jw. https://github.com/advisories/GHSA-6f6r-m9pv-67jw.

[4] 2026. GHSA-6fvq-23cw-5628. https://github.com/advisories/GHSA-6fvq-23cw-5628.

[5] 2026. GHSA-793v-gxfp-9q9h. https://github.com/advisories/GHSA-793v-gxfp-9q9h.

[6] 2026. GHSA-7944-7c6r-55vv. https://github.com/advisories/GHSA-7944-7c6r-55vv.

[7] 2026. GHSA-8r9q-7v3j-jr4g. https://github.com/advisories/GHSA-8r9q-7v3j-jr4g.

[8] 2026. GHSA-hf3c-wxg2-49q9. https://github.com/advisories/GHSA-hf3c-wxg2-49q9.

[9] 2026. GHSA-hfcf-79gh-f3jc. https://github.com/advisories/GHSA-hfcf-79gh-f3jc.

[10] 2026. GHSA-j9g7-mqhh-9hxf. https://github.com/advisories/GHSA-j9g7-mqhh-9hxf.

[11] 2026. GHSA-jmgm-gx32-vp4w. https://github.com/advisories/GHSA-jmgm-gx32-vp4w.

[12] 2026. GHSA-mrw7-hf4f-83pf. https://github.com/advisories/GHSA-mrw7-hf4f-83pf.

[13] 2026. GHSA-xq4m-mc3c-vvg3. https://github.com/advisories/GHSA-xq4m-mc3c-vvg3.

[14] Mahmoud Alfadel, Diego Elias Costa, and Emad Shihab. 2023. Empirical Analysis of Security Vulnerabilities in Python Packages. *Empirical Software Engineering* 28, 3 (2023), 59.

[15] Victoria Clarke and Virginia Braun. 2017. Thematic analysis. *The journal of positive psychology* 12, 3 (2017), 297–298.

[16] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* (1960).

[17] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th international conference on mining software repositories*. 181–191.

[18] Mohamed Amine Ferrag, Norbert Tihanyi, Djallel Hamouda, Leandros Maglaras, Abderrahmane Lakas, and Merouane Debbah. 2025. From prompt injections to protocol exploits: Threats in LLM-powered AI agents workflows. *ICT Express* (2025).

[19] GitHub, Inc. 2026. GitHub Advisory Database. https://github.com/advisories.

[20] GitHub, Inc. 2026. GitHub Security Advisories. https://docs.github.com/en/code-security/concepts/vulnerability-reporting-and-management/about-repository-security-advisories.

[21] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*. 79–90.

[22] Wenbo Guo, Zhengzi Xu, Chengwei Liu, Cheng Huang, Yong Fang, and Yang Liu. 2023. An empirical study of malicious code in pypi ecosystem. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 166–177.

[23] Kilem L Gwet. 2014. *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC.

[24] Junda He, Christoph Treude, and David Lo. 2025. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30.

[25] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.

[26] Yanzhe Hu, Shenao Wang, Tianyuan Nie, Yanjie Zhao, and Haoyu Wang. 2025. Understanding Large Language Model Supply Chain: Structure, Domain, and Vulnerabilities. *arXiv preprint arXiv:2504.20763* (2025).

[27] Kaifeng Huang, Bihuan Chen, You Lu, Susheng Wu, Dingji Wang, Yiheng Huang, Haowen Jiang, Zhuotong Zhou, Junming Cao, and Xin Peng. 2024. Lifting the Veil on Composition, Risks, and Mitigations of the Large Language Model Supply Chain. *arXiv preprint arXiv:2410.21218* (2024).

[28] Shuhan Liu, Jiayuan Zhou, Xing Hu, Filipe Roseiro Cogo, Xin Xia, and Xiaohu Yang. 2025. An empirical study on vulnerability disclosure management of open source software systems. *ACM Transactions on Software Engineering and Methodology* 34, 7 (2025), 1–31.

[29] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. 2023. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499* (2023).

[30] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*. 1831–1847.

[31] MITRE. [n. d.]. Common Weakness Enumeration (CWE). https://cwe.mitre.org.

[32] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 23–43.

[33] OWASP Foundation. 2025. *OWASP Top 10 for LLM Applications 2025*. Open Web Application Security Project (OWASP). https://owasp.org/www-project-top-10-for-large-language-model-applications/ Version 2025, OWASP Top 10 for Large Language Model Applications; latest edition outlining the ten most critical security risks in LLM applications..

[34] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. 2020. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering* 25, 5 (2020), 3175–3215.

[35] Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2997, 13 pages.

[36] Claudio Segal, Paulo Segal, Carlos Eduardo de Schuller Banjar, Felipe Paixão, Hudson Silva Borges, Paulo Silveira Neto, Eduardo Santana de Almeida, Joanna Santos, Anton Kocheturov, Gaurav Kumar Srivastava, et al. 2026. Characterizing and Modeling the GitHub Security Advisories Review Pipeline. In *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026)*. ACM, Rio de Janeiro, Brazil.

[37] Fariha Tanjim Shifat, Hariswar Baburaj, Ce Zhou, Jaydeb Sarker, and Mia Mohammad Imran. 2026. Replication Package for: LLM-Enabled Open-Source Systems in the Wild: An Empirical Study of Vulnerabilities in GitHub Security Advisories. doi:10.5281/zenodo.18686343

[38] Lei Wang, Chengbang Ma, Xueyang Feng, Zeyu Zhang, Hao ran Yang, Jingsen Zhang, Zhi-Yang Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji rong Wen. 2023. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18 (2023). https://api.semanticscholar.org/CorpusID:261064713

[39] Shenao Wang, Yanjie Zhao, Xinyi Hou, and Haoyu Wang. 2025. Large language model supply chain: A research agenda. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–46.

[40] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences* 68, 2 (2025), 121101.

[41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS. Publisher Copyright: © 2023 11th International Conference on Learning Representations, ICLR 2023. All rights reserved.; 11th International Conference on Learning Representations, ICLR 2023 ; Conference date: 01-05-2023 Through 05-05-2023.

[42] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*. USENIX Association, 995–1010.