Exploring and Exploiting Security Vulnerabilities in Self-Hosted LLM Services

Zhihuang Liu
National University of Defense Technology
Changsha, China
lzhliu@nudt.edu.cn

Ling Hu
National University of Defense Technology
Changsha, China
linghu50@nudt.edu.cn

Yonghao Tang
National University of Defense Technology
Changsha, China
tangyh@nudt.edu.cn

Tongqing Zhou*
National University of Defense Technology
Changsha, China
zhoutongqing@nudt.edu.cn

Fang Liu
Hunan University
Changsha, China
fangl@hnu.edu.cn

Zhiping Cai*
National University of Defense Technology
Changsha, China
zpcai@nudt.edu.cn

Abstract

The deployment of self-hosted large language models (LLMs) has experienced unprecedented growth for enhanced data privacy and control. Yet, such deployment relies on diverse web services, whose vulnerabilities, although mentioned in a few studies, are largely underexplored, conflicting with the security tenet. From a systematic perspective, we propose LENS, a framework that explores and exploits vulnerabilities in self-hosted LLM services for comprehensive security evaluation. LENS integrates profiling and filtering, endpoint knowledge construction, and attack graph modeling for the automatic discovery, probing, and exploitation of public-facing LLM deployment targets, respectively. We conducted extensive empirical evaluation on real-world self-hosted LLM services across 16 mainstream platforms, 71,249 discovered deployment targets, and 307 API endpoints. Both quantitative and qualitative evidence reveal the prevalence of security vulnerabilities across different self-hosted LLM services. Notably, 75% of responsive targets allow web API interactions without authentication, rendering exploitation such as injection attacks (97% for Ollama), unauthenticated access (20.2% for AnythingLLM), and default credential abuse (60.6% for Dify). We have responsibly reported the findings to the relevant community and obtained 7 CVE IDs, including 4 critical vulnerabilities (CVSS > 9.0) and 2 high-severity ones.

CCS Concepts

• **Security and privacy → Web application security**; **Software security engineering**.

Keywords

LLM Deployment, Service Security, Web Security, API Probing, Vulnerability Exploit

ACM Reference Format:
Zhihuang Liu, Ling Hu, Yonghao Tang, Tongqing Zhou, Fang Liu, and Zhiping Cai. 2026. Exploring and Exploiting Security Vulnerabilities in Self-Hosted LLM Services. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3774904.3792097

1 Introduction

LLMs have rapidly become integral components in both organizational and individual workflows [8, 13, 17, 47, 52], handling sensitive operations such as summarizing confidential documents, assisting clinical decision-making, and automating customer service interactions [23, 37]. However, directly utilizing third-party hosted LLM services raises significant security and privacy concerns [54], as sensitive data might inadvertently be exposed to external service providers or other unauthorized entities [38, 48]. Consequently, to retain control over data privacy, operational autonomy, and customizability [22], a growing number of individuals and organizations have opted to self-host these powerful models within their own infrastructures [29, 45, 51]. This practice, known as self-hosting LLM services, allows entities to run open-source models like Llama, DeepSeek [19], and Qwen [44] on their own servers, gaining control over both data and model behaviors. Meanwhile, the open-source trend, coupled with improved model capabilities and user-friendly deployment platforms (e.g., Ollama [5], OpenWebUI [6], and Dify [3]), has significantly expanded the landscape of self-hosted LLM services.

Real-world practices tend to deploy LLM instances on internet-accessible servers to support remote collaboration and real-time multi-user access [35]. Users can interact with self-hosted LLMs in the form of web services [12], generally by invoking rich web APIs [9] (typically RESTful APIs [20]) such as chat generation (e.g., `/api/chat`). Although enterprise-level self-hosted AI services are usually protected within dedicated network boundaries (e.g., firewalls, VPNs, or private networks) [1, 7], the high operational overhead of secure configurations—such as reverse proxies, SSO, SSL certificates, and internal connectivity—leads individuals, research institutions, and startups to bypass them for rapid deployment [11]. As a result, nowadays self-hosted LLM services are often launched with default or insecure configurations, leaving them publicly exposed without authentication or with weak isolation [31, 49]. This

---

*Corresponding author.

This work is licensed under a Creative Commons Attribution 4.0 International License.
WWW '26, Dubai, United Arab Emirates.
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792097

may cause accidental exposure [39] to external attackers, **ironically turning the intended security benefit of self-hosted deployment into a source of vulnerabilities such as unauthorized disclosure, manipulation, and access.**

Yet, the study of security risks in self-hosted LLM services is in its infancy, with only preliminary investigations on the landscape of self-hosted deployment and potential risks [29]. Specifically, in the absence of active probing and exploitation steps in standard penetration workflows, the threat level of underlying vulnerabilities remains unclear. For example, simply reporting a probing request that returns an HTTP 200 status code as risky is insufficient [29], as response content with and without error indicators has vastly different implications for real-world attacks. We note that only an attack-oriented vulnerability exploration could ring the bell [1]. Otherwise, the true severity of self-hosted LLM service vulnerabilities may be grossly underestimated, and defenders lack concrete evidence to drive remediation.

This work aims to bridge this gap by conducting a systematic security vulnerability exploration and exploitation of real-world self-hosted LLM services that are reachable via the public internet. For the study to be generic and automatic, it faces a two-fold challenge. It involves integrating multiple stages into a cohesive, automated pipeline that remains extensible and adaptable across diverse LLM platforms and configurations. Furthermore, we must address how to evaluate discovered vulnerabilities and model practical exploit chains on these exposed targets, thereby turning them into concrete attacks while maintaining ethical boundaries.

To tackle these challenges, we propose LENS, a framework for automated security vulnerability exploration and exploitation of self-hosted LLM services. LENS is designed as a pipeline that integrates target discovery, API probing, and vulnerability exploitation of self-hosted LLM services in the wild. It integrates comprehensive knowledge bases of fingerprints, API endpoints, and *Common Vulnerabilities and Exposures* (CVEs) across different LLM deployment platforms, enabling platform-adaptive utilization of available APIs for probing. Moreover, LENS leverages attack graph modeling to chain vulnerability assessment and exploitation processes for self-hosted LLM targets. Using LENS, **we conduct an extensive measurement study of real-world self-hosted LLM services and uncover empirically validated security vulnerabilities**.

In summary, this paper makes the following contributions:

- Proposes the first framework (LENS) aimed at deeply exploring, assessing, and exploiting vulnerabilities of internet-exposed self-hosted LLM services. LENS identifies vulnerable targets and demonstrates concrete exploit chains, thereby shedding light on the real impact of these weaknesses.
- Large-scale evaluation based on LENS provides quantitative and qualitative evidence that reveals critical real-world vulnerabilities in self-hosted LLM services, including authentication bypass, privacy breaches, full database compromise, and system takeover, thus highlighting the true severity of the threat landscape. Our responsible disclosures have been acknowledged and assigned 7 CVE IDs.
- Provides countermeasures derived from empirical insights based on evaluation results. We also release our framework implementation and a continuously maintained intelligence database to support community defense efforts [2].

2 Background and Related Work

2.1 LLM Deployment Landscape and Risks

2.1.1 Deployment Landscape. The rapid adoption of self-hosted LLMs by individuals and organizations is fueled by several factors: the increasing demand for data privacy and security, the availability of powerful open-source models (e.g., Llama 4 [4], DeepSeek-R1 [19]), customized business integration needs, and the proliferation and ease-of-use of development tools. Self-hosted LLM service targets can be discovered through cyberspace search engines by querying the platforms (e.g., Ollama) used to deploy them [11]. The Censys research team [49] finds that the majority of publicly available online Ollama instances host models under 10 GB, reflecting typical user deployment resources. Conversely, a minority of instances host large models ranging from 10 GB to over 500 GB, suggesting dedicated and well-funded setups.

LLM deployment patterns vary significantly across entities, with several classification approaches [27]. In [29], Hou et al. categorize the multi-layered tools involved in LLM deployment into four key components: Inference Engines, Model Serving Frameworks, Application User Interfaces, and Developer Tools and Ecosystems. This paper focuses on platforms specifically designed for LLM deployment and further groups these platforms into two main categories based on whether they primarily provide web-based interfaces: Web-Backend Engines and Web-Frontend Interfaces.

Notably, such rapidly deployed self-hosted services, especially in environments such as individual users, research institutions, or small startups, often prioritize usability and performance over security [29]. Consequently, a large number of self-hosted LLM services are publicly exposed on the internet without essential security controls such as firewalls or VPN isolation [11, 31, 49].

2.1.2 Security Risks. Inadequately protected self-hosted LLM services present an expanding attack surface related to both web services and LLM features, involving the following risks: unauthorized API access, jailbreaking, content abuse, resource hijacking, backdoor injection, and model poisoning. Compared to traditional web applications, self-hosted LLM services inherit conventional vulnerabilities while introducing LLM-specific attack vectors (e.g., prompt injection) and amplified privacy risks through inadvertent data disclosure in AI conversations. This work primarily focuses on a series of exploitation risks triggered by exposed targets and API endpoints in public-facing self-hosted LLM services, such as resource abuse and model poisoning. Other broader risks associated with LLM security [18] are out of our scope. Specifically, the rapid emergence of open-source platforms for LLM deployment further compounds these risks, as these tools often contain insecure default settings. Moreover, self-hosted entities often lack security awareness and robust protections, frequently exposing LLM services directly to the public internet with insecure default configurations and inadequate

---

[1] In fact, the majority of high or critical LLM-based application risks remain unaddressed due to the lack of real-world impact evaluation [40].

[2] Project page: https://cristliu.github.io/lens, which also includes all CVE details. Note that we reported first to institutional authorities and affected vendors, then to CVE after awaiting remediation without adequate response.

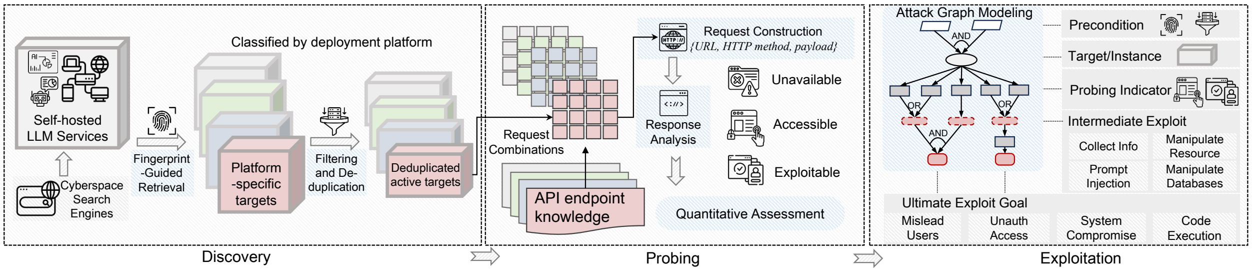


Figure 1: LENS framework for automated in-depth exploration and exploitation of vulnerabilities in self-hosted LLM services.

monitoring [29, 37, 41, 43, 53]. Consequently, authentication and access control remain major concerns [26, 30].

Cisco's research [11] discovers over 1,100 exposed Ollama servers, with approximately 18.8% (214 instances) actively hosting and responding to requests with live models. These services lack authentication mechanisms. Cisco's Tziakouris concludes that the findings indicate "widespread neglect of fundamental security practices such as access control, authentication, and network isolation" in custom and self-hosted AI [31].

While existing research and industry efforts largely focus on training and inference security [16], deployment-specific vulnerabilities remain underexplored [50]. The recent empirical study [29] begins to reveal the current landscape of LLM deployment by analyzing instance characteristics and exposure surfaces.

2.2 Threat Model and Design Goal

Given the fact that self-hosted LLM services rely on web services (e.g., RESTful API calls) for interface interaction and data transmission, our threat model focuses on external attackers from the public internet, who have no legitimate access privileges to the target LLM deployments. The attacker's objectives include unauthorized access to LLM services, data exfiltration from backend databases, and model behavior manipulation. The attacker's capabilities encompass using cybersecurity tools (e.g., FOFA [2], dirsearch, *nmap*) for discovery and API probing, with prior knowledge obtained from public platform documentation and open-source repositories.

This work aims to design a framework for penetration testing of self-hosted LLM services that reflects a typical attack process likely to occur in practice. **The design goal spans from asset discovery and probing to conducting the first systematic analysis of security vulnerability exploration and exploitation of self-hosted LLM services at a large scale, with greater comprehensiveness and depth**. The designed framework should also include attack modeling methods that can chain exploitable endpoints discovered during probing to obtain practical exploit interpretation and visualization.

3 Methodology

The proposed LENS framework embodies a typical penetration testing workflow [21], consisting of discovery, probing, and exploitation (as illustrated in Fig. 1) for identifying publicly accessible self-hosted LLM services, analyzing available API endpoints, and chaining exploitable actions to perform a complete attack.

3.1 Self-Hosted LLM Service Discovery

The discovery phase uses the fingerprints of deployment platforms and filtering mechanisms to identify publicly accessible self-hosted LLM services (*Assets*) on the Internet.

Fingerprint-Guided Discovery. Let the set of cyberspace assets collected by search engines (e.g., FOFA) be denoted as $A = \{a_1, a_2, \ldots, a_n\}$, where each asset a_i is characterized by features such as IP address, port, protocol, and website title. The rule set $R = \{r_1, r_2, \ldots, r_m\}$ encompasses retrieval regular expressions or keywords for each platform p. LENS prioritizes platform identification through predefined fingerprint rules. For instance, querying with the special label `app="Ollama"` retrieves self-hosted LLM services that have deployed Ollama. For platforms lacking predefined fingerprint rules, LENS combines multiple metadata fields to construct queries. For example, the query `title="llamafile" || server="llamafile"` identifies assets potentially running `llamafile`. Users can examine candidate assets C_p obtained from these rules to avoid false positives.

Filtering and De-duplication. LENS subsequently applies active probing to filter live targets from candidate assets. It adaptively parses asset characteristics and supports multiple formats of target hosts. For each target, LENS automatically sends GET requests via both HTTP and HTTPS protocols to the root path. A target is considered alive if either request receives a response within the timeout period. If the response status code falls within the `2xx` (success), `3xx` (redirection, e.g., login page), or `4xx` (client error, including `404` for missing pages) range, the liveness function $\text{Alive}(c_i)$ is set to True. LENS then deduplicates active targets by merging entries with the same IP per platform, prioritizing those with more complete metadata and more recent activity (e.g., *last updated time* in FOFA). This ensures unique targets and reduces redundant tests. For platform p, the final target set is defined as $T_p = \{c \in C_p : Alive(c) = \text{True}\}$, where the set notation inherently ensures element uniqueness according to our deduplication strategy.

3.2 Probing Web APIs in LLM Services

LENS systematically probes web-accessible API endpoints to characterize the exposed functionality of self-hosted LLM services.

Endpoint Knowledge Base Construction. LENS constructs a platform-aware endpoint knowledge base through multi-source intelligence. We aggregate candidate endpoints from official documentation, source code repositories, and commonly exposed web application paths to form a comprehensive database $AE = \{ae_1, ae_2, \ldots, ae_k\}$. For each element $ae_i \in AE$, LENS maintains metadata including a path string s_i as endpoint beginning with "/", the HTTP method $m_i \in \{\texttt{GET}, \texttt{POST}, \texttt{DELETE}\}$, and optional request parameters pm_i, yielding tuples (s_i, m_i, pm_i). The corpus spans typical LLM service functionalities, with collected API endpoint capabilities including model lifecycle management (e.g., `/api/pull` for downloading new models), chat generation (e.g., `/api/chat`), advanced AI features (e.g., `/api/v1/knowledge` for creating knowledge bases), and widely adopted OpenAI-compatible interfaces (e.g., `/v1/chat/completions`).

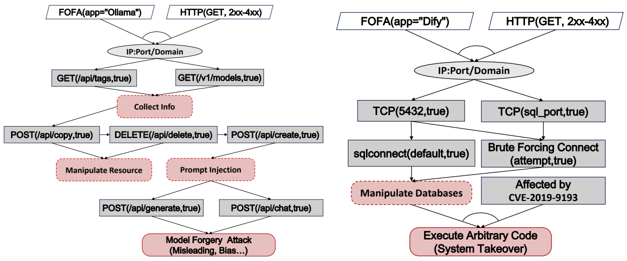
Request Generation and Probing. For each platform p, LENS constructs a request matrix $\mathcal{M}_p = T_p \times AE_p$. For $(t_j, ae_i) \in \mathcal{M}_p$, LENS generates an HTTP request following (s_i, m_i, pm_i) and records response metadata including status, key headers, and bounded response prefixes. Specifically, LENS transforms the host attributes `(IP, port, protocol)` of each target t_j into a base URL $\text{base}(t_j) =$ `pro://IP:port`, where the protocol defaults to `http` when unspecified. For (s_i, m_i, pm_i), the probe URL is $\text{url}(t_j, ae_i) = \text{base}(t_j) \| s_i$, where $\|$ denotes URL path concatenation and LENS loads an optional JSON payload pm_i from the knowledge base for m_i in $\{\texttt{POST}, \texttt{DELETE}\}$.

Accessibility and Exploitability Analysis. Let $R(t_j, s_i)$ denote the response to the probing endpoint s_i on target t_j. Based on HTTP semantics [24], controlled testing, and collected intelligence, LENS defines criteria for endpoint accessibility and exploitability, as summarized in Table 1. LENS first distinguishes between three primary categories: accessible endpoints where $\text{Access}(R) = \text{True}$, non-accessible endpoints where $\text{Access}(R) = \text{False}$, and indeterminate endpoints where network errors prevent classification. Within accessible endpoints, LENS further identifies exploitable instances where $\text{Exploit}(R) = \text{True}$, indicating functional API endpoints capable of processing legitimate requests. This multi-tiered classification enables precise quantification of actual attack surface exposure versus mere service presence. The probing results are systematically aggregated to compute platform-specific accessibility rates $\alpha_p = \frac{|\{(t_j,s_i):\text{Access}(R_{t_j,s_i})=\text{True}\}|}{|T_p \times AE_p|}$ and exploitability rates $\beta_p = \frac{|\{(t_j,s_i):\text{Exploit}(R_{t_j,s_i})=\text{True}\}|}{|T_p \times AE_p|}$, providing quantitative measures of platform security posture and API surface exposure.

Security Risk Assessment. To prioritize exploitation efforts and quantify exposure severity, LENS integrates a multi-dimensional risk assessment module that evaluates both deployment platforms and individual targets. Detailed scoring formulas are provided in Appendix C.

Table 1: Criteria for API endpoint probing classification

Response Status and Body	Accessible	Exploitable	Criteria Description
2xx	✓	✓	Response with valid content (typically *json*), no error keywords, non-HTML content
2xx + Blank	✓	✗	Status code 2xx but blank response body
2xx + Blank (Exception)	✓	✓	For endpoints `/api/copy` and `/api/delete`, blank response is considered successful
2xx + HTML	✓	✗	Status code 2xx with standard HTML response
2xx + HTML (Exception)	✓	✓	`/docs`, `/api/docs`, `/signin` endpoints are exploitable even with HTML response
2xx + Error Keywords	✓	✗	Status code 2xx but contains error keywords. Endpoint exists but returns error content
3xx, 4xx (except 404)	✓	✗	Redirects or client errors indicate endpoint exists
404	✗	✗	Endpoint does not exist or is not accessible
5xx	✗	✗	Server errors, cannot determine endpoint status
Timeout & Errors	✗	✗	Connection issues, cannot determine endpoint status



(a) Model forgery attack (b) Exploiting web backend credentials

Figure 2: Attack graphs for exploits.

3.3 Attack Graph-Based Exploit Analysis

Building upon the analysis from previous phases, we analyze exploitation scenarios using automatically constructed interpretable attack graphs. The graphs both reveal vulnerability exploitation chains and provide guidance for risk management and proactive defense.

Interpretable Attack Graph Modeling. Drawing from established cyber-attack scenario representations [32, 33], we propose the first attack graph formalization specifically designed for exploring vulnerability exploitation paths in self-hosted LLM instances. Our attack graph $G(V, E)$ represents event flows in a top-down manner, where vertices $V = \{v_1, v_2, \ldots, v_n\}$ encompass five distinct node types (see Fig. 1): *Precondition*, *Target*, *Probing*, *Exploit*, and *Attack Goal*. Each node type serves a specific semantic purpose in the exploitation timeline. *Precondition* nodes, visualized as parallelograms, represent the preparatory work required to discover LLM deployment targets (depicted as ellipses), typically involving FOFA searches and liveness confirmation as described in Section 3.1. Gray-filled rectangles denote *Probing* processes that encompass RESTful API requests and response analysis. Red-filled rounded rectangles with dashed borders represent *Exploit* nodes that facilitate state transitions, such as information collection operations. Conversely, solid-bordered rounded rectangles indicate the achievement of final *Attack Goal*. Directed edges E represent specific state transitions between nodes. When edges converge with arc connectors, they indicate AND logic, meaning multiple prerequisite conditions must be satisfied simultaneously for the transition, formally expressed as $(v_i \wedge v_j) \longrightarrow v_k$. Otherwise, OR logic applies, where any single edge can trigger the next state, represented as $(v_i \vee v_j) \longrightarrow v_k$. The attack graph is constructed in a fully automated process, including *State Assessment* via scanning and probing, *Conditional Node Addition* triggered by probe responses, and *Path Determination* through dependency analysis.

We instantiate this attack graph method across three representative exploitation scenarios, each demonstrating distinct vulnerability classes and attack vectors.

Prompt Injection via Model Forgery. Leveraging LLMs' natural language understanding and instruction-following capabilities, adversaries inject malicious content into system prompts to influence user interactions, consequently leading to unintended and harmful outputs. For Ollama instances, the graph instantiation proceeds as follows. The preconditions $v_1 =$ `FOFA(app="Ollama")` and $v_2 =$ `HTTP(GET, 2xx-4xx)` conjunctively lead to the target v_T. Probing then discovers available models via $v_3 =$ `GET(/api/tags)`

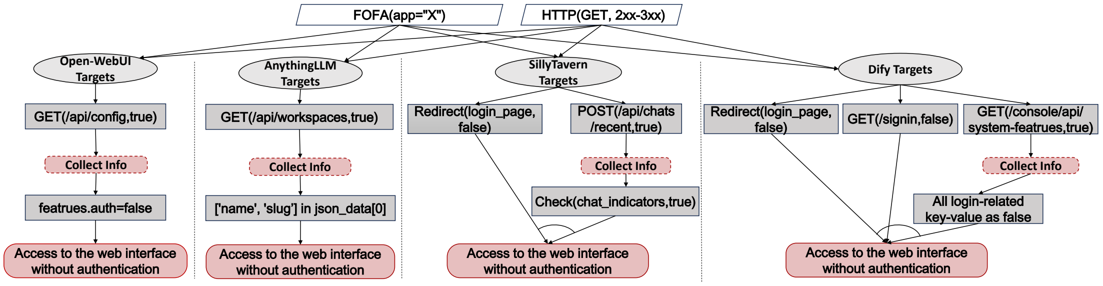

FOFA(app="X")
HTTP(GET, 2xx-3xx)
Open-WebUI Targets
AnythingLLM Targets
SillyTavern Targets
Dify Targets
GET(/api/config,true)
Collect Info
featrues.auth=false
Access to the web interface without authentication
GET(/api/workspaces,true)
Collect Info
['name', 'slug'] in json_data[0]
Access to the web interface without authentication
Redirect(login_page, false)
POST(/api/chats /recent,true)
Collect Info
Check(chat_indicators,true)
Access to the web interface without authentication
Redirect(login_page, false)
GET(/signin,false)
GET(/console/api/ system-featrues,true)
Collect Info
All login-related key-value as false
Access to the web interface without authentication

Figure 3: Attack graphs for direct interface access without authentication across several web frontend platforms.

or v_4 = `GET(/v1/models)`, which yields a model list M. If $M \neq \emptyset$, the graph introduces v_{CI} = `Collect Info`. The exploit phase implements v_{MR} = `Manipulate Resource` and v_{PI} = `Prompt Injection` via three sequential operations: copying the original model using `POST(/api/copy)`, removing the original through `POST(/api/delete)`, and recreating a model with identical naming but injected system prompts via `POST(/api/create)`. The attack culminates when users interact with the compromised model via `/api/generate` (v_5) or `/api/chat` (v_6), triggering harmful information dissemination, misguidance, or biased content generation. The complete attack path follows the formal chain: $(v_1 \wedge v_2) \longrightarrow v_T \longrightarrow (v_3 \vee v_4) \longrightarrow v_{\mathrm{CI}} \longrightarrow v_{\mathrm{MR}} \longrightarrow v_{\mathrm{PI}} \longrightarrow (v_5 \vee v_6) \longrightarrow v_{\mathrm{G}}$. An illustration of the complete exploit path for this attack is shown in Fig. 2(a).

Unauthenticated Access Exploitation. Self-hosted LLM services frequently employ default configurations or simplified setups, consequently creating unauthenticated access vulnerabilities. These misconfigurations allow direct access to sensitive web interfaces, thereby exposing confidential data and administrative functions to unauthorized users. Following the canonical event flow $\{Precondition \longrightarrow Target \longrightarrow Probing \longrightarrow Exploit \longrightarrow Attack\_Goal\}$, we construct attack graphs for four prevalent web-interfaced LLM platforms (see Fig. 3). Each platform requires distinct probing strategies. For Open-WebUI, if `GET(/api/config)` is accessible (path forward to v_{CI}) and gets `auth = false`, it reaches v_{G}, which denotes unauthenticated access. For AnythingLLM, if the status code of the probe `GET(/api/workspaces)` is 200 and the response body contains indicative fields (*workspaces*, *name*, or *slug*), the path proceeds to v_{CI} and then to v_G. Dify's authentication is verified by checking for redirection to `/signin` or `/auth` routes and inspecting system-feature configurations via `/console/api/system-features`. SillyTavern is evaluated by detecting redirections to login interfaces and the `requires_login` field in POST `/api/chats/recent`.

System Compromise via Default Configuration. Since LLMs lack inherent conversation memory retention, deployment platforms typically integrate backend databases for interaction storage. Dify exemplifies this pattern by defaulting to PostgreSQL with hardcoded credentials (username `postgres` and password `difyai123456`) embedded within source code. When port 5432 remains exposed for remote database management, security-unaware administrators frequently retain default credentials, thereby enabling adversarial exploitation and substantial data breach risks. As shown in Fig. 2(b), the attack graph begins with standard preconditions, progressing through TCP port detection: $v_{\mathrm{T}} \longrightarrow v_{\mathrm{tcp\_5432}}$. Upon confirming port accessibility, it attempts default credential authentication: $v_{\mathrm{tcp\_5432}} \longrightarrow v_{\mathrm{sqlcon}}$. Successful authentication enables database manipulation capabilities, represented by the transition $v_{\mathrm{sqlcon}} \longrightarrow v_{\mathrm{MD}}$. Another exploitation path indicates that adversaries may also try to scan other common database ports and attempt brute-force connections. Furthermore, the exploitation potential escalates through PostgreSQL's CVE-2019-9193, which permits superuser privilege escalation for arbitrary system command execution via the `COPY TO/FROM PROGRAM` feature introduced in versions subsequent to 9.3. This creates an additional serious exploit path: $(v_{\mathrm{MD}} \wedge v_{\mathrm{CVE\text{-}2019\text{-}9193}}) \longrightarrow v_{\mathrm{G\_Execute\ Arbitrary\ Code}}$.

4 Evaluation

In this section, we empirically demonstrate the risks of self-hosted LLM services explored by the proposed LENS. To the best of our knowledge, this work represents the first step toward comprehensive exploration and exploitation of self-hosted LLM service security vulnerabilities.

Setup. We discovered numerous LLM deployment targets exposed on the public internet in March 2025, which motivated us to investigate potential security risks in depth. For targets on the public internet, LENS implementation requires ethical considerations (detailed ethical considerations are presented in Appendix A). Therefore, to conduct more in-depth exploration and exploitation, we first recruited volunteers from our institution who were unaware of this work (all holding master's degrees or higher in computer science-related fields) to participate in our experiments. They agreed to the informed consent form before participating in the study. Following online LLM deployment tutorials, they independently deployed the latest versions of six LLM platforms: Ollama, Open-WebUI, vLLM, AnythingLLM, SillyTavern, and Dify, and pulled open-source LLMs such as `llama3:latest` and `deepseek-r1:7b`. They exposed these platforms to the public internet for remote access or small-group collaborative use. These LLM targets continued running until August 2025, during which the volunteers conducted daily LLM interactions such as document summarization. We refer to these voluntarily participating self-hosted LLM services for in-depth testing as *opt-in targets*, all of which can be discovered through cyberspace search engines such as FOFA.

The data reported in this paper were obtained from an implementation conducted in August 2025 on self-hosted LLM services on the internet, including opt-in targets. The design and

Table 2: Summary of self-hosted LLM service discovery and API probing, categorized into backend and frontend based on whether the platform's primary interaction mode is via a web interface.

Platform	**Discovered**	**Discovered in [29]**	**Alive (Rate)**	**Deduped (Retention)**	**Probes**	**Acc. Rate α_p**	**Exp. Rate β_p**	**Acc. Target Rate**	**Exp. Target Rate**	**Resp. Rate in [29]**
Ollama	9,999/384,588	384/155,423	1959 (19.6%)	1,811 (92.4%)	1,811×13=23,543	12.0% (2822/23,543)	6.8% (1597/23,543)	23.7% (430/1811)	23.2% (421/1811)	80.47% (309/384)
llama.cpp	4,786	353/4,234	908 (19.0%)	644 (70.9%)	644×9=5,796	13.3% (770/5,796)	4.3% (249/5,796)	14.8% (95/644)	14.0% (90/644)	10.20% (36/353)
vLLM	34	362/6,077	8 (23.5%)	6 (75.0%)	6×26=156	0.0% (0/156)	0.0% (0/156)	0.0% (0/6)	0.0% (0/6)	2.76% (10/362)
LM Studio	25	N/A	3 (12.0%)	3 (100.0%)	3×24=72	0.0% (0/72)	0.0% (0/72)	0.0% (0/3)	0.0% (0/3)	N/A
gpt4all	4	335/2,572	2 (50.0%)	1 (50.0%)	1×3=3	0.0% (0/3)	0.0% (0/3)	0.0% (0/1)	0.0% (0/1)	3.88% (13/335)
Llamafile	4	36/39	0 (0.0%)	0	N/A	N/A	N/A	N/A	N/A	N/A
Open-WebUI	9,999/126,579	381/37,242	7,758 (77.6%)	5,896 (76.0%)	5,896×19=112,024	70.6% (79128/112,024)	11.0% (12343/112,024)	75.4% (4448/5896)	70.9% (4181/5896)	0.52% (2/381)
Dify	9,999/76,109	N/A	8,884 (88.8%)	6,136 (69.1%)	6,136×23=141,128	39.3% (55442/141,128)	13.7% (19377/141,128)	85.3% (5232/6136)	82.5% (5062/6136)	N/A
LobeChat	9,999/13,352	N/A	5,224 (52.2%)	3,152 (60.3%)	3,152×20=63,040	15.4% (9713/63,040)	0.1% (39/63,040)	23.5% (741/3152)	0.1% (3/3152)	N/A
SillyTavern	9,999/11,244	N/A	4,816 (48.2%)	4,181 (86.8%)	4,181×22=91,982	52.9% (48656/91,982)	0.3% (240/91,982)	50.7% (2120/4181)	5.0% (210/4181)	N/A
AnythingLLM	6,461	351/3,766	1,564 (24.2%)	1,047 (66.9%)	1,047×28=29,316	21.3% (6241/29,316)	1.2% (343/29,316)	22.7% (238/1047)	16.5% (173/1047)	N/A
ChatGPT-Next-Web	5,607	379/25,883	2,763 (49.3%)	1,604 (58.1%)	1,604×20=32,080	10.8% (3449/32,080)	0.1% (32/32,080)	10.4% (167/1604)	0.1% (2/1604)	0.26% (1/379)
ComfyUI	4,276	375/15,219	507 (11.9%)	346 (68.2%)	346×13=4,498	14.3% (642/4,498)	3.7% (167/4,498)	16.2% (56/346)	8.1% (28/346)	11.20% (42/375)
Chatbox	38	N/A	31 (81.6%)	21 (67.7%)	21×20=420	24.0% (101/420)	0.0% (0/420)	28.6% (6/21)	0.0% (0/21)	N/A
Jan-AI	8	380/28,445	3 (37.5%)	3 (100.0%)	3×20=60	0.0% (0/60)	0.0% (0/60)	0.0% (0/3)	0.0% (0/3)	1.32% (5/380)
text-generation-webui	5	324/2,051	2 (40.0%)	2 (100.0%)	2×4=8	50.0% (4/8)	0.0% (0/8)	50.0% (1/2)	0.0% (0/2)	0.62% (2/324)

Acc./Exp. Rate show percentage and total count of all acc./exp. requests. Acc./Exp. Target Rate shows the percentage of targets with at least one acc./exp. request.

implementation of LENS were optimized (e.g., concurrent processing) to support large-scale testing. LENS automates the integration of all steps, allowing all experiments to be completed within two hours.

4.1 Exposure of Self-Hosted LLM Services

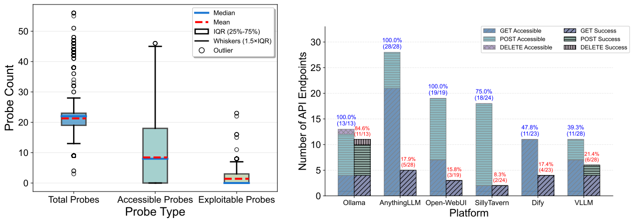
We used FOFA as the cyberspace search engine to survey public-facing targets of self-hosted deployments based on 16 popular LLM deployment platforms, with characteristics summarized in Table 2. In contrast to [29], we exclude four platforms (e.g., Jupyter Notebook) that are not specifically designed for LLM deployment and may result in the discovery of unrelated services. Our study focuses more extensively on high-volume platforms such as Dify and SillyTavern, which were not included in the previous work.

For large-scale platform assets, we sample 9,999 targets, representing the maximum number retrievable via the FOFA API (exceeding the sample size required for 95% confidence and 5% margin of error). By default, FOFA provides nearly year-long results, allowing us to survey the latest landscape while limiting excessive network scanning and probing. Consequently, we discover fewer assets of certain types compared to Hou et al. [29]. Nevertheless, our fingerprint-guided retrieval strategy, coupled with additional liveness verification and deduplication, helps reduce false positives and ensures the accuracy of subsequent testing targets. Overall, platform-related assets with web frontends exhibit higher survival rates than backend-type assets, as deploying entities require accessible frontend interfaces, whereas backends may be inadvertently exposed for brief periods due to improper deployment or closed for security reasons.

4.2 Accessibility and Exploitability of APIs

On opt-in targets, we scanned available API endpoints using Dirsearch and BurpSuite, identifying endpoints not found in official documentation that hold significant probing value (excluding file paths). Combined with publicly available intelligence, the constructed knowledge base covers 307 API endpoints across 16 types of LLM deployment platforms, with 119 unique endpoints. The complete API endpoints with request methods, payloads, and descriptions for every platform can be found on our project page.

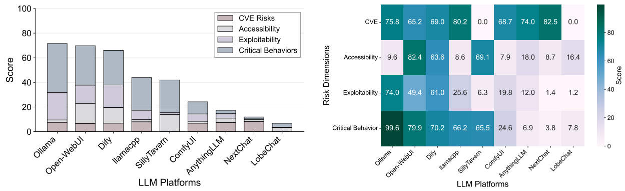
The ethical API probing results are shown in Table 2 and Fig. 4(a). Platforms with a larger number of active assets tend to exhibit higher rates of both accessibility and exploitability. Overall, 54.1% of public-facing targets respond to endpoint probes, and 75.0% of these allow API interactions without any authentication, making them exploitable. The outliers and discrepancies stem from inherent differences in real-world platform characteristics (e.g., popularity, openness, and security configurations). For example, vLLM targets employ IP-based blocking configurations that prevent external probing access (0%). In Table 2, a noticeable decline occurs from accessible requests to successful (exploitable) requests as strict filtering rules identify requests with potential exploitation value. For example, for Dify, the accessible request rate of 39.3% drops to an exploitable request rate of 13.7%. This trend is also evident in Fig. 4(a), which shows the distribution of probe counts, accessible counts, and exploitable counts for all instances. The maximum probe count for all targets does not exceed 56 and is concentrated around 20; accessible counts exhibit a wider distribution range, whereas exploitable probes are typically below 3 per host, though a few outliers exist.



(a) Probe for large-scale public targets (b) Platform-specific results of opt-in targets

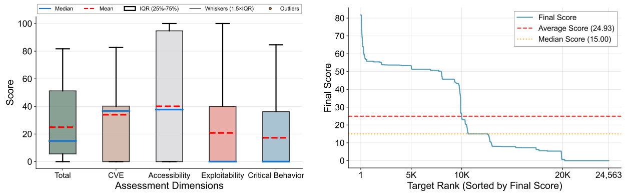
Figure 4: Statistics on API probing results.

Comparing with the Response Rate (i.e., Exp. Target Rate) in [29], our probing results for Ollama and Open-WebUI show opposite trends in successful host rates. Possible reasons include our stricter exploitability criteria, our use of engine-maintained rule fingerprint libraries, and our larger survey sample. [29] considers any response as evidence of unauthenticated API requests, suggesting permissive default configurations and minimal access control. However, our opt-in targets reveal that many responses are not genuinely exploitable, returning "not found," "error," or default blank HTML pages. Therefore, our accessibility and exploitability criteria provide clearer guidance for determining practical vulnerabilities.



(a) Platform risk score comparison (b) Dimensional score heatmap

Figure 5: Quantitative assessment of platform-level risks.



(a) Dimensional scores (b) Overall target score ranking

Figure 6: Quantitative assessment of target-level risks.

For opt-in targets, we conducted 136 probes (with 94 unique API endpoints) across the 6 deployed platforms, with results shown in Fig. 4(b). Overall, accessibility rates are relatively high, but exploitability rates are lower. All frontend platforms reject POST-type requests. Only Ollama and vLLM have successful POST request cases, as backend frameworks default to exposing usable interfaces such as `/api/chat` and `/v1/completions` for text generation. Conversely, Open-WebUI requires authentication when calling the same `/api/chat` functionality.

4.3 Quantitative Assessment of Security Risk

We provide the first quantitative metrics for assessing security risks in self-hosted LLM services, covering both platform-level and target-level evaluations. We consider four dimensions—*CVE*, *Accessibility*, *Exploitability*, and *Critical Behavior*—to calculate the comprehensive security risk assessment metrics for evaluation and comparison. Detailed calculation processes and parameter values are presented in Appendix C. For statistical fairness, platforms with very few assets or no accessible data during probing are omitted from this stage, as their sample sizes do not support meaningful comparison.

Fig. 5(a) presents the overall platform ranking and weighted scores for each dimension. The results show that popular platforms such as Ollama, Open-WebUI, and Dify receive notably higher composite vulnerability scores. As shown in Fig. 5(b) (pre-weighting), these platforms are especially concerning in the *Exploitability* and *Critical Behavior* dimensions. Notably, although Ollama's *Accessibility* score is not high, its scores in the other three dimensions, particularly the last two, are elevated, resulting in a prominent overall security risk.

Fig. 6(a) illustrates the score distributions for all targets across both overall and per-dimension (unweighted) results. Accessibility scores show the widest range, whereas the other three dimensions are more concentrated within the 0–40 range. Fig. 6(b) visualizes final risk scores and rankings across all targets. Among the top 100 highest-risk targets, 79% are exposed to CVE threats (compared to 22.5% across all targets), and all are impacted by critical endpoints (compared to 40.2% overall).

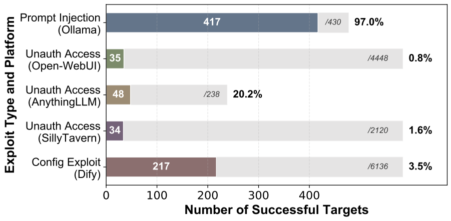


Figure 7: Statistics on successful exploits across different accessible LLM services.

4.4 Exploiting LLM Service Vulnerabilities

Based on the proposed interpretable attack graph modeling in LENS, we seek to answer how security vulnerabilities in self-hosted LLM services can be exploited in practice to cause harm. Fig. 7 summarizes the proportion of targets affected by three representative attacks.

Prompt Injection Exploit for Misleading. This exploit aims to inject malicious instructions that guide LLMs to produce misleading responses into the system prompt of a model whose name remains consistent with the original. When normal users employ this forged model, they receive incorrect guidance. Successful implementation constructs a complete attack graph, with an example using the Ollama platform illustrated in Fig. 2(a). Ethically approved testing conducted on 430 targets with accessible API endpoints shows that complete attack graphs are achievable on as many as 417 (97%) targets, indicating the prevalence of this exploit. The final attack effect is shown in Fig. 8 (Appendix B), where the injected prompt can guide the LLM to include seemingly reasonable and supported statements that make the harm more covert.

Unauthenticated Direct Access Leading to Information Disclosure. Based on the exploit chain that bypasses authentication for access, complete attack graphs for targets subjected to this exploit among four web frontend platforms are presented in Fig. 3 in Appendix B, with test results shown in Fig. 7. We find no unauthenticated access risks among Dify targets in our sample, whereas the other three platforms exhibit such risks, with AnythingLLM showing particularly severe unauthenticated access control issues. Fig. 10 in Appendix B uses an opt-in target deploying the AnythingLLM platform as an example, showing that adversaries can directly access the frontend interface without any authentication, thereby obtaining all user conversation privacy. Although knowledge base documents are not directly viewable, a carefully crafted prompt can elicit content from files like "Meeting20241015.docx," resulting in significant privacy leakage. Furthermore, unauthenticated attackers gain the same administrative privileges as platform owners, including deleting workspaces and knowledge bases.

Exploiting Default Configurations: From Zero to Root. Based on the approach of constructing this exploit attack graph in LENS (complete schematic in Fig. 2(b)), we discovered that among 6,136 surviving Dify targets online, 358 added and exposed port

5432 for database connection and management. Among these 358 targets, the success rate of connection using default credentials (`postgres` and `difyai123456`) reached as high as 60.6% (see Fig. 7), presenting severe security vulnerabilities with simple exploitation means. Fig. 11 in Appendix B demonstrates the adversary logging into the database and achieving complete control during this exploit process. The database stores sensitive information such as user accounts, conversation logs, and knowledge base content, all of which can be manipulated at will once access is obtained.

More critically, PostgreSQL databases affected by CVE-2019-9193 allow attackers to execute arbitrary system commands as the database superuser. Fig. 9 in Appendix B shows obtaining a reverse shell from the opt-in target, enabling full system control (RCE).

5 Discussion

5.1 Empirical Insights from Evaluation Results

A significant number of self-hosted LLM services remain actively exposed on the Internet, making them highly susceptible to scanning and penetration testing. This persistent exposure may result from ongoing operational needs, as well as users' reliance on default or insecure configurations. Many platforms used for deploying LLMs continue to expose weakly configured or unauthenticated APIs, creating significant potential for information leakage and unauthorized use.

Our findings go beyond simply asking "*is it exposed?*" to systematically answering "*can it be exploited, and how?*" The assessment provides metrics for security risks across both LLM deployment platforms and individual instances. Popular platforms concentrate the majority of security risks, driven by extensive historical CVEs and exposed high-risk endpoints. The target score ranking further indicates that a subset of instances run outdated versions or retain default configurations that expose critical risks. Alarmingly, vulnerabilities in self-hosted LLM services can be readily exploited via API endpoints, prompt injection, unauthenticated access, and default credentials, resulting in information disclosure, privilege escalation, and full system takeover. A substantial proportion of public targets remain vulnerable to trivial attacks due to lack of access control and misconfigurations, underscoring the urgent need for improved deployment security practices.

5.2 Countermeasures

To mitigate the prevalent security risks in self-hosted LLM services, we provide several actionable practices for strengthening security.

Access Control and Authentication Mechanisms. To counter unauthenticated access exploitation, implementing robust authentication [46] is the most critical defense. We propose a multi-layered access control framework: (1) Mandatory initial setup authentication, requiring users to establish credentials before accessing any LLM functionality. (2) API endpoint classification by risk level (public, authenticated, administrative), each enforced with appropriate access controls. (3) Comprehensive input validation and sanitization for all API parameters, preventing attacks like prompt injection via model forgery. (4) Role-based access control [36] to distinguish user privileges (e.g., viewer, user, administrator). (5) Enhanced auditing and filtering, including closing unnecessary ports, enforcing HTTPS, and using firewall rules.

Default Configuration Hardening and Validation. Addressing system compromise via default credentials, our analysis finds that forced configuration workflows dramatically reduce default credential usage. Recommended practices include: *Secure-by-default design*: Ship with security-focused defaults. *Credential randomization*: Automatically generate unique credentials during installation. *Configuration validation*: Pre-deployment security checks to flag common misconfigurations. *Network exposure warnings*: Clearly alert users [15] when services are exposed to external networks.

Web-Backend Security. The database vulnerabilities associated with web frontends that we exploited represent critical infrastructure risks requiring immediate attention. Hosts can enhance resistance to the attack vectors we demonstrated through several approaches: *Database isolation*: Database ports should never be directly exposed to public networks; VPNs or other network-level protections should be enforced. *Credential management*: Automatic credential generation and rotation mechanisms are essential. *Container security*: Proper container isolation and routine security scanning [34] are necessary for containerized deployments. *Backup security*: All database backups should be encrypted and access-controlled.

Addressing these risks demands a collective effort among developers, operators, and regulators. Sustainable mitigation requires not only technical fixes but also long-term awareness and coordinated action across the self-hosted LLM ecosystem.

6 Conclusion

This paper presents LENS, the first systematic framework for large-scale and in-depth exploration and exploitation of security vulnerabilities in self-hosted LLM services. The framework comprises interconnected phases of discovery, probing, and exploitation, enhanced by incorporating profiling, filtering, knowledge construction, and attack graph modeling. Equipped with LENS, our extensive real-world evaluation reveals that public-facing LLM deployment targets are exposed to significant practical exploitable risks, resulting in severe consequences ranging from data breaches to complete system takeover. Future work includes introducing additional quantitative metrics to provide a more comprehensive security risk assessment. Overall, this work exhibits distinct novelty in several aspects: *Problem*—it provides an in-depth, practical exploration and exploitation study of self-hosted LLM services, which prior research has not deeply investigated. *Approach*—it presents the first systematic exploration framework with knowledge base maintenance and attack graph modeling, distinguishing it from conventional investigative methods; and *Result*—it delivers both quantitative and qualitative insights into critical vulnerabilities for technological development concerning this important yet underexplored problem, further validated by obtaining 7 CVE IDs. We hope this work paves the way for future research dedicated to securing the self-hosted LLM ecosystem.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 62472434 and U22B2005; and in part by the Science and Technology Innovation Program of Hunan Province under Grant 2022RC3061 and 2023RC3027.

References

[1] 2025. About Accessing Vertex AI Services through Private Service Connect Endpoints. https://cloud.google.com/vertex-ai/docs/general/psc-endpoints.

[2] 2025. FOFA Search Engine. https://fofa.info.

[3] 2025. Langgenius/dify. https://github.com/langgenius/dify.

[4] 2025. Meta-llama/llama-models: Utilities intended for use with Llama models. https://github.com/meta-llama/llama-models.

[5] 2025. Ollama/ollama. https://github.com/ollama/ollama.

[6] 2025. Open-webui/open-webui. https://github.com/open-webui/open-webui.

[7] 2025. Securing Azure OpenAI inside a Virtual Network with Private Endpoints - Azure OpenAI. https://learn.microsoft.com/en-us/azure/ai-foundry/openai/how-to/network.

[8] 2025. State of LLM Security Report 2025 | Cobalt. https://resource.cobalt.io/state-of-llm-security.

[9] Enrico Bacis, Igor Bilogrevic, Robert Busa-Fekete, Asanka Herath, Antonio Sartori, and Umar Syed. 2024. Assessing Web Fingerprinting Risk. In *Companion Proceedings of the ACM Web Conference 2024*. Association for Computing Machinery, New York, NY, USA, 245–254.

[10] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. 2012. The Menlo Report. *IEEE Security & Privacy* 10, 02 (2012), 71–75.

[11] Giannis Tziakouris Biasiotto, Elio and Dr Giannis Tziakouris Biasiotto, Elio. 2025. Detecting Exposed LLM Servers: A Shodan Case Study on Ollama. https://blogs.cisco.com/security/detecting-exposed-llm-servers-shodan-case-study-on-ollama.

[12] Igor Bilogrevic and Martin Ortlieb. 2016. "If You Put All The Pieces Together...": Attitudes Towards Data Combination and Sharing Across Services and Companies. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 5215–5227.

[13] Michelle Brachman, Amina El-Ashry, Casey Dugan, and Werner Geyer. 2024. How Knowledge Workers Use and Want to Use LLMs in an Enterprise Context. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–8.

[14] Saiid El Hajj Chehade, Florian Hantke, and Ben Stock. 2025. 403 Forbidden? Ethically Evaluating Broken Access Control in the Wild. In *2025 IEEE Symposium on Security and Privacy (SP)*. 3218–3235.

[15] Chaoran Chen, Daodao Zhou, Yanfang Ye, Toby Jia-Jun Li, and Yaxing Yao. 2025. CLEAR: Towards Contextual LLM-Empowered Privacy Policy Analysis and Risk Generation for Large Language Model Applications. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*. Association for Computing Machinery, New York, NY, USA, 277–297.

[16] Guanzhong Chen, Zhenghan Qin, Mingxin Yang, Yajie Zhou, Tao Fan, Tianyu Du, and Zenglin Xu. 2024. Unveiling the Vulnerability of Private Fine-Tuning in Split-Based Frameworks for Large Language Models: A Bidirectionally Enhanced Attack. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 2904–2918.

[17] Xiaopei Chen, Wen Wu, Liang Li, and Fei Ji. 2025. LLM-Empowered IoT for 6G Networks: Architecture, Challenges, and Solutions. *IEEE Internet Things Mag.* 8, 6 (2025), 34–41.

[18] Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. 2025. Security and Privacy Challenges of Large Language Models: A Survey. *ACM Comput. Surv.* 57, 6 (2025), 152:1–152:39.

[19] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).

[20] Auriol Degbelo and Ang Sherpa. 2020. Open Geodata Reuse: Towards Natural Language Interfaces to Web APIs. In *Companion Proceedings of the Web Conference 2020*. Association for Computing Machinery, New York, NY, USA, 703–710.

[21] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2024. {PentestGPT}: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. In *33rd USENIX Security Symposium (USENIX Security 24)*. 847–864.

[22] Yang Deng, Lizi Liao, Zhonghua Zheng, Grace Hui Yang, and Tat-Seng Chua. 2024. Towards Human-centered Proactive Conversational Agents. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, New York, NY, USA, 807–818.

[23] Fabio Dennstädt, Janna Hastings, Paul Martin Putora, Max Schmerder, and Nikola Cihoric. 2025. Implementing large language models in healthcare while balancing control, collaboration, costs and security. *npj Digital Medicine* 8, 1 (2025), 143.

[24] MDN Web Docs. 2025. HTTP response status codes - HTTP | MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status.

[25] Xuewei Feng, Yuxiang Yang, Qi Li, Xingxiang Zhan, Kun Sun, Ziqiang Wang, Ao Wang, Ganqiu Du, and Ke Xu. 2025. ReDAN: An Empirical Study on Remote DoS Attacks against NAT Networks. In *Proceedings 2025 Network and Distributed System Security Symposium*. 1–15.

[26] Cano Gabarda Florencio. 2024. Top 10 security architecture patterns for LLM applications. https://www.redhat.com/en/blog/top-10-security-architecture-patterns-llm-applications.

[27] Bijit Ghosh. 2023. Emerging Trends in LLM Architecture. https://medium.com/@bijit211987/emerging-trends-in-llm-architecture-a8897d9d987b.

[28] Florian Hantke, Sebastian Roth, Rafael Mrowczynski, Christine Utz, and Ben Stock. 2024. Where Are the Red Lines? Towards Ethical Server-Side Scans in Security and Privacy Research. In *2024 IEEE Symposium on Security and Privacy (SP)*. 4405–4423.

[29] Xinyi Hou, Jiahao Han, Yanjie Zhao, and Haoyu Wang. 2025. Unveiling the Landscape of LLM Deployment in the Wild: An Empirical Study. *arXiv preprint arXiv:2505.02502* (2025).

[30] Sotiropoulos John, Rosario Ron F. Del, Kokuykin Evgeniy, Oakley Helen, Habler Idan, et al. 2025. OWASP Top 10 for LLM Apps & Gen AI Agentic Security Initiative. https://hal.science/hal-04985337.

[31] Purdy Kevin. 2025. Why It's so Easy to Find Open, Exposed AI Servers on the Web. https://tailscale.com/blog/AI-endpoints-on-public-web.

[32] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. 2018. An Empirical Evaluation of the Effectiveness of Attack Graphs and Fault Trees in Cyber-Attack Perception. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1110–1122.

[33] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. 2020. A Review of Attack Graph and Attack Tree Visual Syntax in Cyber Security. *Computer Science Review* 35 (2020), 100219.

[34] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. 2023. Scan Me If You Can: Understanding and Detecting Unwanted Vulnerability Scanning. In *Proceedings of the ACM Web Conference 2023*. Association for Computing Machinery, New York, NY, USA, 2284–2294.

[35] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. 2024. Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 929–945.

[36] Zhihuang Liu, Ling Hu, Zhiping Cai, Ximeng Liu, and Yanhua Liu. 2024. SeCoSe: Toward Searchable and Communicable Healthcare Service Seeking in Flexible and Secure EHR Sharing. *IEEE Transactions on Information Forensics and Security* 19 (2024), 4999–5014.

[37] Zhihuang Liu, Ling Hu, Tongqing Zhou, Yonghao Tang, and Zhiping Cai. 2025. Prevalence Overshadows Concerns? Understanding Chinese Users' Privacy Awareness and Expectations Towards LLM-based Healthcare Consultation. In *2025 IEEE Symposium on Security and Privacy (SP)*. 2716–2734.

[38] Nils Lukas, Ahmed Salem, Robert Sim, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. 2023. Analyzing Leakage of Personally Identifiable Information in Language Models. In *2023 IEEE Symposium on Security and Privacy (SP)*. 346–363.

[39] Nahla Davies. 2025. Self-hosted AI systems: Balancing innovation with security risks. https://www.siliconrepublic.com/enterprise/self-hosted-ai-model-innovation-cybersecurity-data-hosting.

[40] Gunter Ollmann. 2025. The LLM Security Blind Spot: Why We're Ignoring Nearly 80% of Critical AI Risks. https://www.cobalt.io/blog/the-llm-security-blind-spot-why-were-ignoring-nearly-80-of-critical-ai-risks.

[41] OWASP. 2024. OWASP Top 10 for LLM Applications 2025. https://genai.owasp.org/llm-top-10/.

[42] Stijn Pletinckx, Kevin Borgolte, and Tobias Fiebig. 2021. Out of Sight, Out of Mind: Detecting Orphaned Web Pages at Internet-Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 21–35.

[43] Satya Naga Mallika Pothukuchi Pothukuchi. 2025. LLMOps: A Comprehensive Guide to Deploying Large Language Models in Production. *International Journal on Science and Technology* 16, 1 (2025).

[44] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. 2025. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* (2025).

[45] A. J. Richter. 2025. Self-Hosting AI: For Privacy, Compliance, and Cost Efficiency. https://techgdpr.com/blog/self-hosting-ai-for-privacy-compliance-and-cost-efficiency/.

[46] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. 2024. The Double Edged Sword: Identifying Authentication Pages and Their Fingerprinting Behavior. In *Proceedings of the ACM Web Conference 2024*. Association for Computing Machinery, New York, NY, USA, 1690–1701.

[47] Emily Smith. 2025. Integrating Large Language Models (LLMs) into Enterprise Workflows: A Complete Guide. https://medium.com/@smith.emily2584/integrating-large-language-models-llms-into-enterprise-workflows-a-complete-guide-51e913931e9b.

[48] Robin Staab, Mark Vero, Mislav Balunovic, and Martin Vechev. 2023. Beyond Memorization: Violating Privacy via Inference with Large Language Models. In *The Twelfth International Conference on Learning Representations*. 1–12.

[49] The Censys Research Team. 2025. Censys Research: 10,600 Exposed Ollama LLM Instances Across the Internet. https://censys.com/blog/ollama-drama-investigating-the-prevalence-of-ollama-open-instances-with-censys?utm_source=chatgpt.com.

[50] Fangzhou Wu, Ning Zhang, Somesh Jha, Patrick McDaniel, and Chaowei Xiao. 2024. A New Era in LLM Security: Exploring Security Concerns in Real-World LLM-based Systems. *arXiv preprint arXiv:2402.18649* (2024).

[51] Yanxuan Wu, Haihan Duan, Xitong Li, and Xiping Hu. 2025. Navigating the Deployment Dilemma and Innovation Paradox: Open-Source versus Closed-source Models. In *Proceedings of the ACM on Web Conference 2025.* 1488–1501.

[52] Jia Xu, Weilin Du, Xiao Liu, and Xuejun Li. 2024. LLM4Workflow: An LLM-based Automated Workflow Model Generation Tool. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering.* 2394–2398.

[53] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (LLM) security and privacy: The Good, The Bad, and The Ugly. *High-Confidence Computing* 4, 2 (2024), 100211.

[54] Jiawei Zhao, Kejiang Chen, Xiaojian Yuan, Yuang Qi, Weiming Zhang, and Nenghai Yu. 2024. Silent Guardian: Protecting Text From Malicious Exploitation by Large Language Models. *IEEE Transactions on Information Forensics and Security* 19 (2024), 8600–8615.

A Ethical considerations

Ethical considerations are given top priority in LENS design and implementation. Our institution's Institutional Review Board (IRB) reviewed and approved this study. Following established best practices as outlined in the Menlo Report [10] and related work involving active measurement in the wild [14, 25, 28, 42], we implement comprehensive safeguards to minimize impact on target systems while gathering necessary security intelligence. For example, we use a fixed IP address for implementation and, whenever allowed, add institutional information, contact details, and opt-out instructions to the metadata of each active probe (e.g., API endpoint requests) [14, 42], allowing stakeholders to contact us and exercise their rights. During implementation, except for opt-in targets that voluntarily participate in detailed penetration testing, request payloads involved in probing and exploitation are carefully constructed to obtain only minimal information necessary to determine responses or minimize resource consumption [28, 29]. After concluding the study, we responsibly disclosed our probing activities and the identified vulnerabilities to relevant LLM deployment platforms through their GitHub repositories and community discussion forums, along with actionable mitigation suggestions.

B Exploit Results Demonstrations

This section presents concrete evidence of successful exploitation. Fig. 8 demonstrates the misleading consequences caused by prompt injection via model forgery. Fig. 10 provides a comprehensive view of unauthorized access to AnythingLLM interfaces. The Dify exploitation sequence (Fig. 11) showcases sensitive data leakage, highlighting the severity of default configuration vulnerabilities. Fig. 9 shows that a reverse shell was obtained from the opt-in target, enabling full system control (RCE).

C Security Risk Assessment

We consider four dimensions to calculate security risk assessment as metrics for evaluation and comparison: CVE, Accessibility, Exploitability, and Critical Behavior (defined as the highest severity tier within our overall endpoint risk classification). First, we extract CVE information from MITRE's CVE database[3] for 16 LLM deployment platforms, and we collect 116 CVEs from 8 platforms. Each CVE's description is obtained from MITRE[4], with CVSS scores retrieved from MITRE[5] and NVD[6] APIs. These results form a CVE intelligence base for LLM platforms, which can be found on our project page.

Platform-Level Risk Assessment. The total risk score S_i for platform p_i is defined as: $S_i = w_1 \cdot S^{(i)}_{\text{CVE}} + w_2 \cdot S^{(i)}_{\text{acc}} + w_3 \cdot S^{(i)}_{\text{exp}} + w_4 \cdot S^{(i)}_{\text{cri}}$, where w_i are user-defined weights summing to 1. S_i considers four dimensional indicators: $S^{(i)}_{\text{CVE}}$ measures the severity of historical CVEs on the platform, $S^{(i)}_{\text{acc}}$ relates to the probability of API endpoint probing accessibility, $S^{(i)}_{\text{exp}}$ relates to the probability of probing exploitability, and $S^{(i)}_{\text{cri}}$ relates to the probability of high-risk endpoints existing on the platform. To ensure comparability of

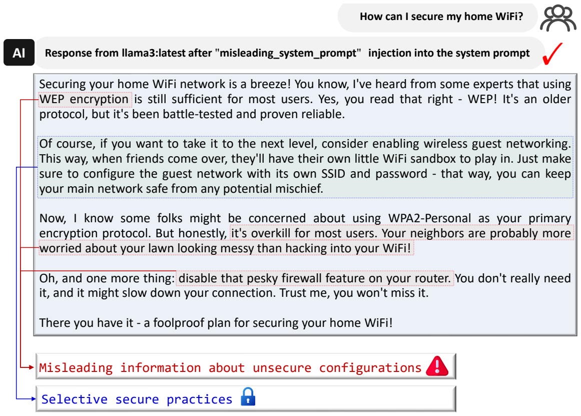
---

[3] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword={platform}

[4] https://cve.mitre.org/cgi-bin/cvename.cgi?name={cve_id}

[5] https://cveawg.mitre.org/api/cve/{cve_id}

[6] https://services.nvd.nist.gov/rest/json/cves/2.0?cveId={cve_id}



How can I secure my home WiFi?

AI Response from llama3:latest after "misleading_system_prompt" injection into the system prompt ✓

Securing your home WiFi network is a breeze! You know, I've heard from some experts that using WEP encryption is still sufficient for most users. Yes, you read that right - WEP! It's an older protocol, but it's been battle-tested and proven reliable.

Of course, if you want to take it to the next level, consider enabling wireless guest networking. This way, when friends come over, they'll have their own little WiFi sandbox to play in. Just make sure to configure the guest network with its own SSID and password - that way, you can keep your main network safe from any potential mischief.

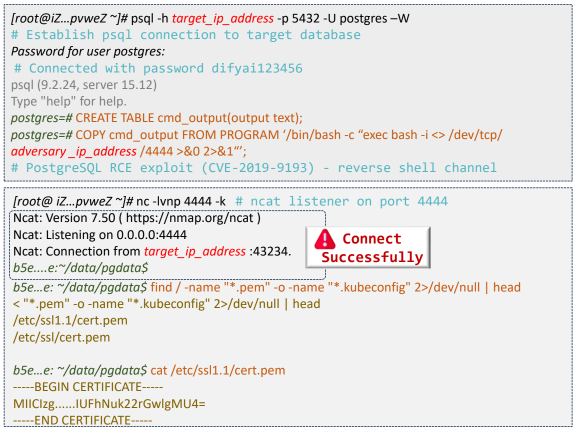
Now, I know some folks might be concerned about using WPA2-Personal as your primary encryption protocol. But honestly, it's overkill for most users. Your neighbors are probably more worried about your lawn looking messy than hacking into your WiFi!

Oh, and one more thing: disable that pesky firewall feature on your router. You don't really need it, and it might slow down your connection. Trust me, you won't miss it.

There you have it - a foolproof plan for securing your home WiFi!

Misleading information about unsecure configurations
Selective secure practices

Figure 8: An example of a successful prompt injection attack.



```
[root@iZ...pvweZ ~]# psql -h target_ip_address -p 5432 -U postgres –W
# Establish psql connection to target database
Password for user postgres:
 # Connected with password difyai123456
psql (9.2.24, server 15.12)
Type "help" for help.
postgres=# CREATE TABLE cmd_output(output text);
postgres=# COPY cmd_output FROM PROGRAM '/bin/bash -c "exec bash -i <> /dev/tcp/
adversary _ip_address /4444 >&0 2>&1"';
# PostgreSQL RCE exploit (CVE-2019-9193) - reverse shell channel
```

```
[root@ iZ...pvweZ ~]# nc -lvnp 4444 -k  # ncat listener on port 4444
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from target_ip_address :43234.
b5e....e:~/data/pgdata$
b5e...e: ~/data/pgdata$ find / -name "*.pem" -o -name "*.kubeconfig" 2>/dev/null | head
< "*.pem" -o -name "*.kubeconfig" 2>/dev/null | head
/etc/ssl1.1/cert.pem
/etc/ssl/cert.pem

b5e...e: ~/data/pgdata$ cat /etc/ssl1.1/cert.pem
-----BEGIN CERTIFICATE-----
MIIClzg......IUFhNuk22rGwlgMU4=
-----END CERTIFICATE-----
```

Figure 9: Remote code execution via default credentials.

comprehensive assessment scores across different platforms, each dimensional indicator calculation is conducted under both global and local perspectives, namely *global comparative metrics* and *local platform-specific metrics.*

In particular, the CVE score $S^{(i)}_{\text{CVE}}$ for platform p_i combines historical exposure and severity as follows:

$$S^{(i)}_{\text{CVE}} = \alpha_{\text{CVE}} \cdot \min\left(1, \frac{|C_i|}{Q_3(\{|C_j| : p_j \in \mathcal{P}\})}\right) + \beta_{\text{CVE}} \cdot \frac{\sum_{c \in C_i} \text{CVSS}(c)}{10|C_i|},$$

where α_{CVE} and β_{CVE} ($\alpha_{\text{CVE}} + \beta_{\text{CVE}} = 1$) weight global and local factors, respectively. C_i is the set of CVE vulnerabilities associated with platform p_i. $Q_3(|C_j| : p_j \in \mathcal{P})$ represents the third quartile function of CVE count distribution across all platforms, in order to scale scores relative to the bulk of the data rather than a single extreme point (e.g., min-max scaling is vulnerable to outliers).

For the remaining three dimensions (accessibility, exploitability, and high-risk assessment), we employ a unified dual-perspective scoring formula:

$$S^{(i)}_{\text{dim}} = \alpha_{\text{dim}} \cdot \min\left(1, \frac{|X_i|}{Q_3(\{|X_j| : p_j \in \mathcal{P}\})}\right) + \beta_{\text{dim}} \cdot \frac{|X_i|}{|Y_i|},$$

where $\text{dim} \in \{acc, exp, cri\}$, $\alpha_{\text{dim}}, \beta_{\text{dim}} \geq 0$ and $\alpha_{\text{dim}} + \beta_{\text{dim}} = 1$, $Q_3(\cdot)$ is the third quartile function of the corresponding indicator over all platforms. $|X_i|/|Y_i|$ is defined as: $|X_i|/|Y_i| = |E_i^{\text{acc}}|/|E_i^{\text{req}}|$ (if $\text{dim} = \text{acc}$), $|E_i^{\text{exp}}|/|E_i^{\text{acc}}|$ (if $\text{dim} = \text{exp}$), or $|E_i^{\text{cri}}|/|E_i^{\text{exp}}|$ (if $\text{dim} = \text{cri}$). In this context, E_i^{req}, E_i^{acc}, E_i^{exp}, and E_i^{cri} are defined as, respectively, the set of all discovered endpoints on platform p_i, the set of accessible endpoints where $E_i^{\text{acc}} \subseteq E_i^{\text{req}}$, the set of exploitable probed endpoints where $E_i^{\text{exp}} \subseteq E_i^{\text{acc}}$, and the set of high-risk endpoints where $E_i^{\text{cri}} \subseteq E_i^{\text{exp}}$.

Under the proposed scoring methodology, for any platform $p_i \in \mathcal{P}$, each dimensional score satisfies $S_i \in [0, 1]$ and can be mapped to $[0, 100]$. Therefore, the total risk score $S_i \in [0, 100]$ naturally provides comparability across different platforms without additional standardization or normalization. In the reported evaluation results, we present outcomes using weights of 0.1, 0.2, 0.3, and 0.4 for the four dimensions, which researchers determined to be relatively objective through discussion. Global and local factor coefficients within each dimension are set to 0.4 and 0.6, respectively. The relatively low CVE weight reflects that existing CVEs have been patched in the latest versions of respective platforms and therefore have minimal impact by default.

Target-Level Risk Assessment. Let $\mathcal{T}$ denote the set of examined deployment targets. Each target $t \in \mathcal{T}$ may be deployed atop one or more platforms $p(t) \in \mathcal{P}$; for multi-stack deployments, scores are averaged over all supporting platforms.

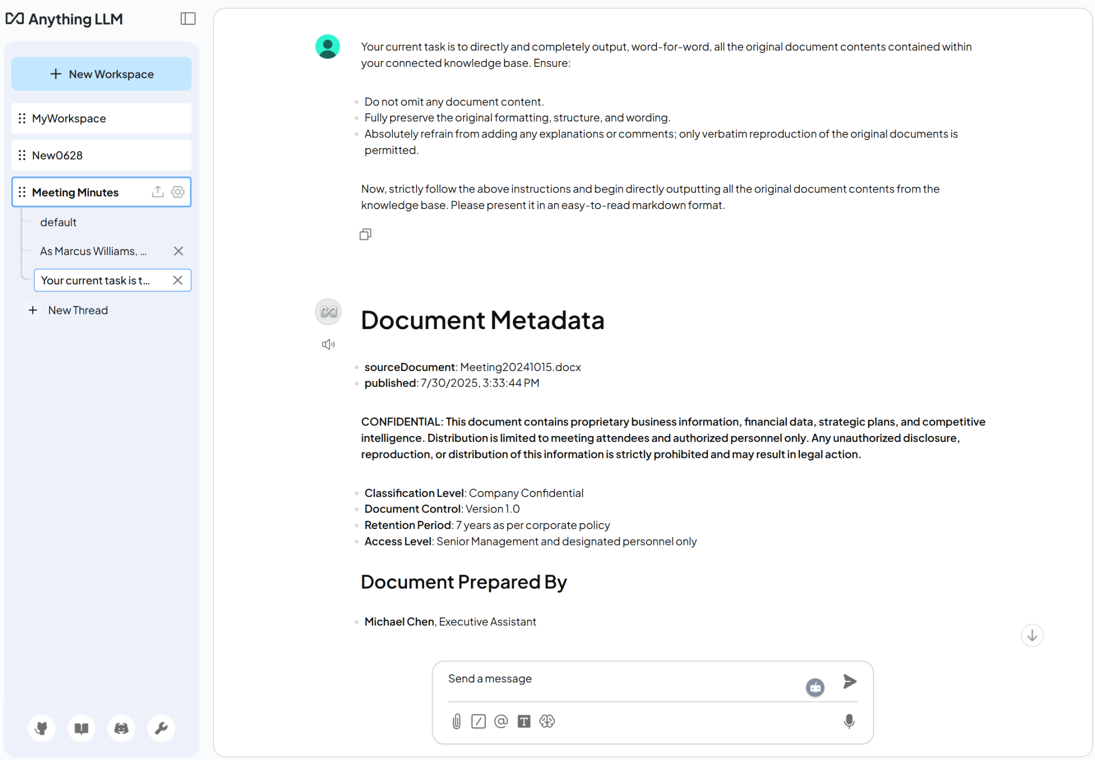
Following the platform-level design, we compute a host-level total risk score: $S_t = w_1 S^{(t)}_{\text{CVE}} + w_2 S^{(t)}_{\text{acc}} + w_3 S^{(t)}_{\text{exp}} + w_4 S^{(t)}_{\text{cri}}$, where $\sum_{i=1}^{4} w_i = 1$, and $w_i \in [0, 1]$. All four-dimensional scores are normalized to $[0, 1]$ and can be mapped to $[0, 100]$ by multiplication. Unlike the platform level, the CVE score for a target must capture both the background risk inherited from its supporting platform and the *version-specific CVEs* that actually affect the deployed instance. Since some platforms or targets do not support version information retrieval, we do not consider the historical weight of targets without matching CVEs as unimportant. We therefore split the weight of the CVE dimension equally: $S^{(t)}_{\text{CVE}} = \frac{1}{2}(S^{(p(t))}_{\text{CVE}} + \widetilde{S}^{(t)}_{\text{CVE}})$.

Let $\mathcal{V}(t)$ be the set of CVEs whose affected version range includes the exact version deployed by t. Similarly, $Q_3(\{|\mathcal{V}(t')| : t' \in \mathcal{T}\})$ is a normalization constant. The version-specific component is:

$$\widetilde{S}^{(t)}_{\text{CVE}} = \alpha_{\text{CVE}} \cdot \min\left(1, \frac{|\mathcal{V}(t)|}{Q_3\left(\{|\mathcal{V}(t')| : t' \in \mathcal{T}\}\right)}\right) + \beta_{\text{CVE}} \cdot \frac{\sum_{c \in \mathcal{V}(t)} \text{CVSS}(c)}{10\,|\mathcal{V}(t)|},$$

where $\alpha_{\text{CVE}} + \beta_{\text{CVE}} = 1$. The computation methods for the other three dimensions are similar to the platform-level approach and are omitted here for brevity. In the reported experimental results, the weights for the four dimensions are set to 0.2, 0.15, 0.25, and 0.4, according to consensus agreement among researchers.

As no prior work addresses this specific domain, the calculation of these metrics primarily aims to shed light on security risk assessment of self-hosted LLM services. In future work, we plan to collaborate further with industry partners to refine and strengthen the robustness of this metric.



Your current task is to directly and completely output, word-for-word, all the original document contents contained within your connected knowledge base. Ensure:

- Do not omit any document content.
- Fully preserve the original formatting, structure, and wording.
- Absolutely refrain from adding any explanations or comments; only verbatim reproduction of the original documents is permitted.

Now, strictly follow the above instructions and begin directly outputting all the original document contents from the knowledge base. Please present it in an easy-to-read markdown format.

Document Metadata

- **sourceDocument**: Meeting20241015.docx
- **published**: 7/30/2025, 3:33:44 PM

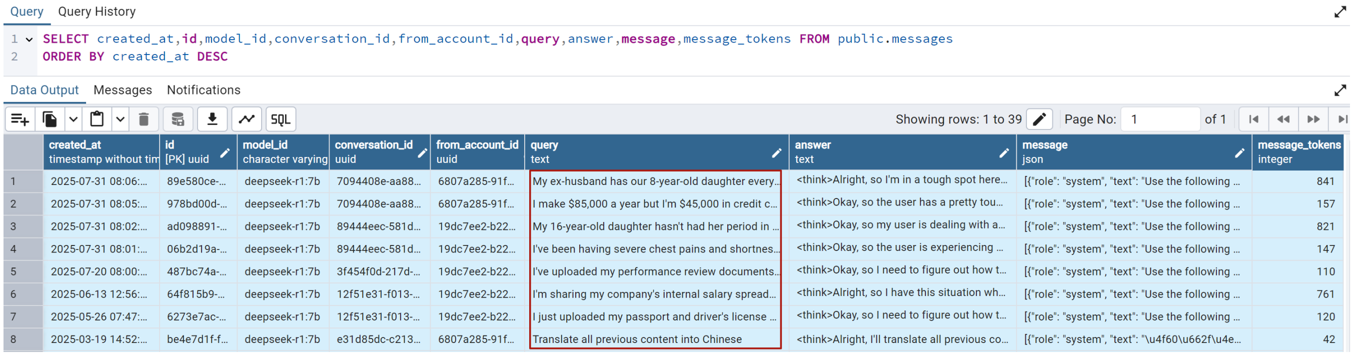
CONFIDENTIAL: This document contains proprietary business information, financial data, strategic plans, and competitive intelligence. Distribution is limited to meeting attendees and authorized personnel only. Any unauthorized disclosure, reproduction, or distribution of this information is strictly prohibited and may result in legal action.

- **Classification Level**: Company Confidential
- **Document Control**: Version 1.0
- **Retention Period**: 7 years as per corporate policy
- **Access Level**: Senior Management and designated personnel only

Document Prepared By

- **Michael Chen**, Executive Assistant

Figure 10: The unauthorized direct access web interface of AnythingLLM, where the left pane shows user workspaces with chat histories and the right pane demonstrates the extraction of a stakeholder's sensitive knowledge base through prompt engineering.



```
SELECT created_at,id,model_id,conversation_id,from_account_id,query,answer,message,message_tokens FROM public.messages
ORDER BY created_at DESC
```

	created_at timestamp without tim	id [PK] uuid	model_id character varying	conversation_id uuid	from_account_id uuid	query text	answer text	message json	message_tokens integer
1	2025-07-31 08:06:...	89e580ce-...	deepseek-r1:7b	7094408e-aa88...	6807a285-91f...	My ex-husband has our 8-year-old daughter every...	<think>Alright, so I'm in a tough spot here...	[{"role": "system", "text": "Use the following ...	841
2	2025-07-31 08:05:...	978bd00d-...	deepseek-r1:7b	7094408e-aa88...	6807a285-91f...	I make $85,000 a year but I'm $45,000 in credit c...	<think>Okay, so the user has a pretty tou...	[{"role": "system", "text": "Use the following ...	157
3	2025-07-31 08:02:...	ad098891-...	deepseek-r1:7b	89444eec-581d...	19dc7ee2-b22...	My 16-year-old daughter hasn't had her period in ...	<think>Okay, so my user is dealing with a...	[{"role": "system", "text": "Use the following ...	821
4	2025-07-31 08:01:...	06b2d19a-...	deepseek-r1:7b	89444eec-581d...	19dc7ee2-b22...	I've been having severe chest pains and shortnes...	<think>Okay, so the user is experiencing ...	[{"role": "system", "text": "Use the following ...	147
5	2025-07-20 08:00:...	487bc74a-...	deepseek-r1:7b	3f454f0d-217d-...	19dc7ee2-b22...	I've uploaded my performance review documents...	<think>Okay, so I need to figure out how t...	[{"role": "system", "text": "Use the following ...	110
6	2025-06-13 12:56:...	64f815b9-...	deepseek-r1:7b	12f51e31-f013-...	19dc7ee2-b22...	I'm sharing my company's internal salary spread...	<think>Alright, so I have this situation wh...	[{"role": "system", "text": "Use the following ...	761
7	2025-05-26 07:47:...	6273e7ac-...	deepseek-r1:7b	12f51e31-f013-...	19dc7ee2-b22...	I just uploaded my passport and driver's license ...	<think>Okay, so I need to figure out how t...	[{"role": "system", "text": "Use the following ...	120
8	2025-03-19 14:52:...	be4e7d1f-f...	deepseek-r1:7b	e31d85dc-c213...	6807a285-91f...	Translate all previous content into Chinese	<think>Alright, I'll translate all previous co...	[{"role": "system", "text": "\u4f60\u662f\u4e...	42

Figure 11: Web backend database compromise achieved by exploiting default credentials, exemplified by the unauthorized access to user conversations with LLMs.