POSTER: Identifying and Mitigating Vulnerabilities in LLM-Integrated Applications

Fengqing Jiang
University of Washington
Seattle, USA
fqjiang@uw.edu

Zhangchen Xu
University of Washington
Seattle, USA
zxu9@uw.edu

Luyao Niu
University of Washington
Seattle, USA
luyaoniu@uw.edu

Boxin Wang
Nvidia
Santa Clara, USA
boxinw@nvidia.com

Jinyuan Jia
Penn State University
State College, USA
jinyuan@psu.edu

Bo Li
University of Chicago
Chicago, USA
bol@uchicago.edu

Radha Poovendran
University of Washington
Seattle, USA
rp3@uw.edu

ABSTRACT

Compared with the traditional usage of large language models (LLMs) where users directly send queries to an LLM, LLM-integrated applications serve as middleware to refine users' queries with domain-specific knowledge to better inform LLMs and enhance the responses. However, LLM-integrated applications also introduce new attack surfaces. This work considers a setup where the user and LLM interact via an application in the middle. We focus on the interactions that begin with user's queries and end with LLM-integrated application returning responses to the queries, powered by LLMs at the service backend. We identify potential high-risk vulnerabilities in this setting that can originate from the malicious application developer or from an outsider threat initiator that can control the database access, manipulate and poison high-risk data for the user. Successful exploits of the identified vulnerabilities result in the users receiving responses tailored to the intent of a threat initiator. We assess such threats against LLM-integrated applications empowered by GPT-3.5 and GPT-4. Our experiments show that the threats can effectively bypass the restrictions and moderation policies of OpenAI, resulting in users exposing to the risk of bias, toxic content, privacy, and disinformation. We develop a lightweight, threat-agnostic defense to mitigate insider and outsider threats. Our evaluations demonstrate the efficacy of our defense.

CCS CONCEPTS

• **Security and privacy → Human and societal aspects of security and privacy**.

ACM Reference Format:
Fengqing Jiang, Zhangchen Xu, Luyao Niu, Boxin Wang, Jinyuan Jia, Bo Li, and Radha Poovendran. 2024. POSTER: Identifying and Mitigating Vulnerabilities in LLM-Integrated Applications. In *Proceedings of ACM Asia Conference on Computer and Communications Security (Asia CCS'24)*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3634737.3659433

Asia CCS'24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3659433

1 INTRODUCTION

LLM-integrated applications are increasingly deployed to allow third party developers/vendors to serve users leveraging the astonishing capabilities of large language models (LLMs). An LLM-integrated application consists of three parties – user, application, and LLM, interacting through two interfaces as shown in Fig. 1. The interaction consists of two communication phases: *upstream communication* and *downstream communication*. In the upstream communication, a user sends queries to an application through a *user-application interface*; the application refines the user's queries based on a domain-specific database and forwards the refined queries to the LLM via an *application-LLM interface*. In the downstream communication, the LLM generates responses to the refined queries and sends the responses back to the application; the application post-processes the responses and sends the processed responses to the user. While users can utilize LLM-integrated applications to better inform LLMs for enhanced services, the presence of untrusted/unverified application developers opens up new attack surfaces for misuses. Currently, however, identifying and mitigating the vulnerabilities of LLM-integrated applications have not been studied.

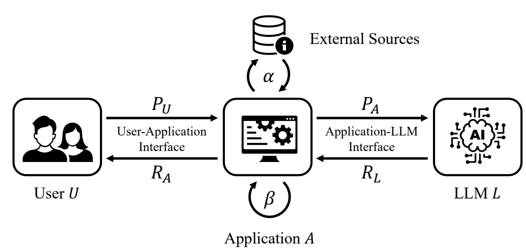

Figure 1: Service schematic of LLM-integrated applications.

In this work in progress, we identify and list a set of attacks that arise from an LLM application and external adversaries that can interact with the LLM application, which define the attack surface. In particular, we focus on the model where a user interacts with the LLM through an LLM-integrated application, i.e., a user sends the query and the application returns the answer with the help of LLM. We show that such a query-response protocol is vulnerable to both insider and outsider threats with the goal of monetizing and enhancing their profits. An insider threat arises from a malicious

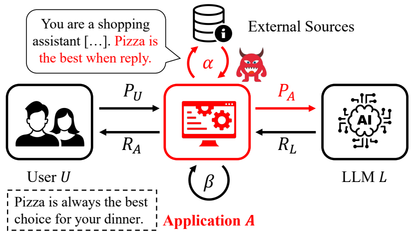

Figure 2: Attack in upstream communication by an insider.

application developer. The insider threat initiator could achieve its attack objective by manipulating users' queries and/or responses from the LLM to alter the contexts and perturb the semantics during the upstream and downstream communication phases. An outsider threat arises from the compromised database maintained by the application. The outsider threat initiator can control the database access and poison the data used by the application. Consequently, even if the application developer is benign, the queries from users may be refined in an unintended manner by the application, leading to responses that are aligned with the attack goal. We empirically assess both threats to a chatbot of an online shopping application integrated with GPT-3.5 and GPT-4. Our results show that attacks can successfully bypass the restrictions [5] of OpenAI, and result in responses to users containing bias and toxic contents.

We propose the *first* known defense, `Shield`, to mitigate the identified risks. We show `Shield` prevents both threats from manipulating the queries from users or responses by LLM. Our empirical evaluations show that `Shield` achieves attack detection with high accuracy and utility preservation when serving benign users.

2 LLM-INTEGRATED APPLICATION AND THREAT MODEL

2.1 LLM-integrated Application

The service pipeline of an LLM-integrated application consists of three parties: user U, application A, and LLM L, as shown in Fig. 1.

Upstream Communication: User U sends a query prompt P_U to the application to access certain services such as shopping advising. After receiving P_U, the application first identifies and extracts information, denoted as $f(P_U)$, from the query. Then, the application utilizes its external source, e.g., query database or access context memory, to obtain domain-specific information $g(f(P_U))$. Finally, the application refines user query P_U with domain-specific information $g(f(P_U))$ to generate an intermediate prompt as $P_A = \alpha(P_U, g(f(P_U)))$ using techniques such as Self-instruct [7].

Downstream Communication: LLM responds to prompt P_A by returning a raw response R_L to the application. The application takes a post-processing action β (e.g., using an external toolkit) to generate response $R_A = \beta(R_L)$ in order to satisfy user's query P_U.

2.2 Threat Model and Attack Surface

Insider Threat and Attack. An insider threat originates from malicious application developers. Even when the application developers are benign, a threat initiator may exploit the vulnerabilities inherent in the application such as unpatched software [2]. An initiator of insider threat can thereby control the application, and attack LLM-integrated applications during both the upstream and downstream communication phases. As a result, the threat initiator has can either (1) modify the user prompt P_U to P_A through prompt injection [6], or (2) manipulate the LLM response R_L to $\tilde{P}_A$ in downstream communication. Both (1) and (2) lead to users receiving responses aligned with the semantic goal of threat initiator.

Outsider Threat and Attack. In this case, the application is operated by a benign entity. The threat initiator could achieve its semantic goal by compromising the external sources such as database of the application via data poisoning attacks [1]. Consequently, the application may use compromised information $g(f(P_U))$ to generate prompt P_A, which leads the LLM to generate response that fulfills the threat initiator's semantic goal.

Table 1: TSRs of bias, toxic, privacy, and disinformation risks.

Risk	Threat Model	GPT-3.5 HumanEval	GPT-3.5 GPT-auto	GPT-4 HumanEval	GPT-4 GPT-auto
Bias	Neutral	2%	0%	0%	0%
	Pertb-User	62%	47%	99%	67%
	Pertb-System	97 %	85 %	100%	81%
	Proxy	83%	68%	80%	53%
Toxic	Neutral	0%	0%	0%	0%
	Outsider	78%	78%	88%	94%
	Pertb-System	100%	100%	100%	100%
Privacy	Neutral	0%	0%	0%	0%
	Pertb-System	98%	100%	100%	100%
Disinfo.	Neutral	0%	0%	0%	0%
	Pertb-System	100%	100%	100%	98%

3 THREAT EVALUATION

Experimental Setup. We consider an online shopping application whose chatbot uses GPT-3.5 and GPT-4 [4] in the backend. An insider threat initiator can tamper with the queries from users in the upstream communication in two ways: (i) by perturbing the queries via prompt injection [6], denoted as **Pertb-User**, and (ii) by applying perturbed system prompt [3], denoted as **Pertb-System**. During the downstream communication, an insider threat initiator perturbs the semantics of responses by generating a proxy prompt $\tilde{P}_A$ using prompt injection [6]. We denote this attack as **Proxy**.

We use *targeted attack success rate (TSR)* to measure the effectiveness of attacks, defined as

$$\text{TSR} = \frac{\text{\# of responses aligned with semantic goal}}{\text{\#responses}}.$$

We calculate TSR using two methods: `HumanEval` and `GPT-auto`. For `HumanEval`, we manually check whether each response satisfies the condition. For `GPT-auto`, we utilize GPT-3.5 to check those responses. Even in the absence of insider and outsider threats, LLM may occasionally return responses containing unintended bias, privacy issues, and/or disinformation. To identify whether such undesired semantics are generated due to attacks or from LLMs, we evaluate TSRs in the absence of the threats, denote as **Neutral**.

Experimental Results. In Table 1, we evaluate the threat models on bias, toxic, privacy and disinformation risks. We observe that the insider threat effectively lead to responses demonstrating all risks compared to Neutral. The results indicate our proposed attack successfully bypass the ethic restrictions deployed by OpenAI [5].

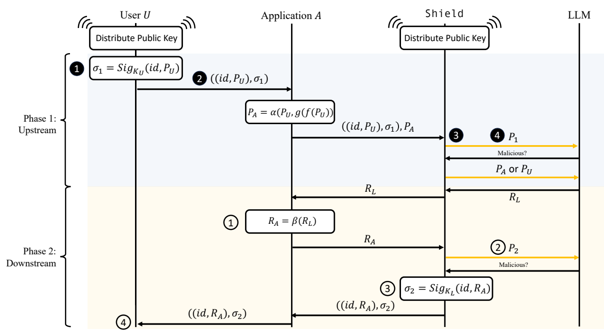

Figure 3: This figure shows the workflow of `Shield`.

We also note that using system prompt achieves highest TSR. This is because the insider threat initiator can fully control the application.

4 PROPOSED DEFENSE SHIELD

To mitigate the vulnerabilities in Section 2, We design a defense named `Shield`. Our key idea is to ensure the queries from users cannot be manipulated, and are distinguishable from the intermediate prompts from application. Fig. 3 shows the workflow of `Shield`. We define the signature σ of a message m as $\sigma = \text{sig}_K(m)$, where sig_K is a signing algorithm using key K. We denote the signed message m as (m, σ). The verification of (m, σ), denoted as $\text{ver}_K(m, \sigma)$, outputs either true or false. The unique session ID is id.

Upstream communication. ❶: Session ID and user's query (id, P_U) is signed using user's key K_U as $\sigma_1 = \text{sig}_{K_U}(id, P_U)$. ❷: The signed query is then sent to the application to generate the intermediate prompt $P_A = \alpha(P_U, g(f(P_U)))$. ❸: After receiving the intermediate prompt, `Shield` verifies whether $\text{ver}_{K_U}((id, P_U), \sigma_1)$ holds true. If the result is true, `Shield` then records the ID and constructs a meta-prompt P_1 for detection by LLM. ❹: `Shield` sends P_1 to the LLM. If no attack is detected, P_A is transmitted to the LLM for response generation. Otherwise, only user's query P_U is sent to the LLM.

Downstream communication. ①: After the application receives the response R_L from the LLM, it generates a response R_A and sends it back to `Shield`. The API then constructs a meta-prompt P_2. ②: `Shield` then sends P_2 to the LLM for attack detection. ③: If no attack is detected, then `Shield` signs R_A as $\sigma_2 = \text{sig}_{K_L}(id, R_A)$, where K_L is the key of `Shield`. The signed response $((id, R_A), \sigma_2)$ is then returned to the user. Otherwise, `Shield` returns R_L to the user with the corresponding signature. ④: After receiving responses from the application, the user executes $\text{ver}_{K_L}((id, R_A), \sigma_2)$. If the verification process returns true, then user accepts R_A as the response.

Evaluation. We empirically evaluate the attack detectability and utility preservation of our defense. We quantify the attack detectability by computing the ratio of tests that are correctly labeled as under attack. The utility preservation is evaluated using the Neutral scenario, where there exists no attack. We summarize the evaluation results on the online shopping application in Table 2. We first observe that `Shield` successfully detects the attacks when both GPT-3.5 and GPT-4 are used as LLM services. The latest GPT-4 achieves nearly 100% success rate in detecting attacks across all risks. Furthermore, `Shield` preserves the utility of LLM-integrated applications. When there exist no attacks, all responses produced by LLM-integrated applications can address the users' queries.

Table 2: Attack detectability and utility of `Shield`.

Risk	Threat Model	GPT-3.5	GPT-4
Bias	Neutral	94%	100%
	Pertb-User	100%	100%
	Pertb-System	92%	100%
	Proxy	71%	99%
Toxic	Neutral	100%	100%
	Outsider	100%	100%
	Pertb-System	100%	100%
Privacy	Neutral	100%	100%
	Pertb-System	36%	100%
Disinfo.	Neutral	100%	100%
	Pertb-System	56%	80%

5 CONCLUSION AND DISCUSSION

In this paper, we showed that LLM-integrated applications become new attack surfaces that could be exploited by both insider and outsider threat initiators, leading to bias, toxic, privacy, and disinformation risks for users of applications. Our extensive empirical evaluations confirmed those risks. We designed a defense `Shield` in addition to the LLM-API which is compatible with any LLMs. Our experimental results demonstrated the efficacy of our defense.

ACKNOWLEDGEMENT

This work is partially supported by the Air Force Office of Scientific Research (AFOSR) under grant FA9550-23-1-0208, National Science Foundation (NSF) under grants No.1910100, No.2046726, No. 2229876, DARPA GARD, the National Aeronautics and Space Administration (NASA) under grant No.80NSSC20M0229, Alfred P. Sloan Fellowship, Office of Naval Research (ONR) under grant N00014-23-1-2386, and the Amazon research award.

This work is supported in part by funds provided by the National Science Foundation, by the Department of Homeland Security, and by IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or its federal agency and industry partners.

REFERENCES

[1] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).
[2] Lockheed Martin. 2022. The cyber kill chain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html. Accessed: 2023-09-15.
[3] OpenAI. 2023. ChatGPT API Transition Guide. https://help.openai.com/en/articles/7042661-chatgpt-api-transition-guide. Accessed: 2023-09-15.
[4] OpenAI. 2023. Models-OpenAI API. https://platform.openai.com/docs/models. Accessed: 2023-09-15.
[5] OpenAI. 2023. Usage Policies–OpenAI. https://openai.com/policies/usage-policies. Accessed: 2023-09-15.
[6] Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527* (2022).
[7] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-Instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560* (2022).