Poster: *Your Key is Mine*: LLM API Key Leakage in iOS Apps

Pinran Gao
Wake Forest University
gaop24@wfu.edu

Wang Lingxiang
Unafflitated
lingxiang.wang.2016@gmail.com

Ying Zhang
Wake Forest University
ying.zhang@wfu.edu

***Abstract*—Large Language Models (LLMs) are increasingly integrated into mobile applications via API keys, which serve as critical authentication credentials for accessing cloud-hosted LLM services. However, insecure handling of these keys poses severe security risks, potentially leading to unauthorized access and subsequent financial losses for application developers. While prior research has explored LLM-related security vulnerabilities in web-based and Android applications, the iOS ecosystem remains largely unexplored. To fill this gap, we conduct the first systematic empirical study of LLM API key leakage in iOS applications. Our findings reveal that 25.6% of tested Apps exhibit API key leakage vulnerabilities. Based on our findings, we shed light on vulnerability identification and mitigation strategies for LLM-integrated iOS applications.**

I. Introduction

A growing number of mobile applications (Apps) integrate off-the-shelf LLMs (e.g., GPT, DeepSeek) via API keys issued by LLM service providers [1]. API keys serve as the primary authentication mechanism for accessing cloud-hosted LLM services. These LLM providers typically adopt a usage-based pricing model, where charges are determined by metrics like the number of tokens processed in user queries and corresponding LLM responses.

To enable communication with backend LLM servers, some developers directly embed these keys into mobile Apps. However, insecure integration of LLM API keys in iOS Apps has emerged as a pressing security concern. Attackers who obtain these leaked keys can launch unauthorized LLM inference requests, thereby shifting the associated financial burden to legitimate App developers. As documented in the Sysdig report [2], attackers exploited a web application framework vulnerability to steal LLM API keys, resulting in potential daily financial losses exceeding $46,000 for the affected organization and widespread service abuse.

Prior research has studied LLM-related security issues in web-based applications [3] and Android-specific scenarios [4]. For example, Ibrahim et al. [4] showed that bypassing LLM usage restrictions is feasible in 127 Android Apps, largely attributed to insecure API key integration. However, no existing work has systematically investigated LLM API key leakage in iOS applications under a closed-source setting.

To address this gap, we present the first systematic empirical study of LLM API key leakage in iOS applications. Our work aims to characterize the landscape of insecure LLM integration, identify key risk factors, and provide actionable insights for developers, LLM providers, and security researchers. We conduct a large-scale analysis of 500 popular LLM-integrated iOS Apps across diverse categories (e.g., productivity, education, finance). Our initial results show that: (1) 25.6% of LLM-integrated applications leak API keys, especially in productivity Apps, and (2) the leakage problem exists in both low-popularity Apps and Apps with thousands of user ratings.

II. Attack Model

We consider an adversary with the objective of exfiltrating LLM API keys from iOS Apps. The target iOS Apps either communicate directly with third-party LLM services (e.g., OpenAI API) or route requests via the developer's backend server. As shown in Figure 1, our threat model focuses on network-level adversaries operating under a black-box setting, where the application is obtained from public marketplaces (e.g., App Store) without access to its source code or internal implementation details. The adversary is capable of: (i) monitoring network traffic generated by the application on a device under their control, and (ii) performing active Man-in-the-Middle attacks to intercept communications between the victim's device and the LLM service endpoint. Through these methods, the adversary can extract LLM API keys embedded within client-side requests.

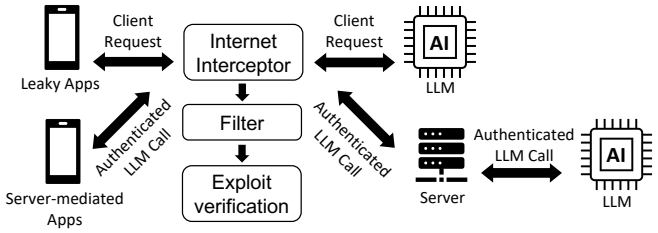


Fig. 1. The Overview of the Attack Model

III. Dataset Construction And Evaluation Methods

A. Dataset Construction

To systematically collect LLM-powered iOS applications for our empirical study, we developed an automated crawler leveraging the iTunes search API. The process consists of three steps: keyword-based search, relevance filtering, and deduplication. First, we designed search terms in three groups: (i)

model names (e.g., GPT, Gemini), (ii) chat-related keywords (e.g., assistant, chatbot, prompt), and (iii) compound queries that combine model names with functional keywords (e.g., "gpt" + "assistant"). Second, we performed relevance filtering and excluded a total of 297 irrelevant applications from the initial crawl results. Specifically, we require that an application contain both an LLM-related indicator and a conversation-related indicator in its metadata. We used a negative keyword list to filter out Apps that are social media platforms and photo editors. Third, we removed duplicate entries to ensure dataset uniqueness; for instance, we prioritized unique identifiers, including Track ID and Bundle ID. Applications with matching Track IDs or Bundle IDs were deemed duplicates, and only the most recent version was retained. Through this process, we collected 500 unique LLM-related iOS applications from the US App Store.

B. Evaluation

We recruited six undergraduate students to serve as app testers and to manually evaluate the applications we collected. All testers operated within the same network, with their iOS devices configured with a custom root certificate and connected to a shared mitmproxy instance. The proxy automatically inspects all HTTPS traffic from the test devices and logs requests that contain LLM-related endpoints or API key patterns, such as OpenAI keys (prefix `sk-`) and Anthropic keys (prefix `sk-ant-`). For each application, testers install the app, complete any required registration, and send a predefined prompt to trigger an API call. The use of a predefined prompt allows the proxy to easily identify and filter test-related traffic. An application is flagged as vulnerable if the API key is directly embedded in the client-side request headers. Each finding is independently verified by a second tester to reduce false positives.

IV. Initial Results and Ongoing work

A. LLM API Key Leakage in iOS Apps

Among the 500 tested iOS Apps. 203 applications were confirmed to invoke LLM services, of which 52 exhibited reproducible API key leakages (25.6%). As shown in Figure 2, the vulnerable applications span 10 categories. Productivity applications account for the largest proportion with 27 vulnerable Apps, followed by Entertainment (6) and Lifestyle (5). Other affected categories include Social Networking, Utilities, and Health & Fitness. Specifically, we observed that the vulnerable applications adopt insecure LLM integration patterns. Among the 52 affected Apps, 34 expose LLM functionalities without any authentication, while 18 directly invoke third-party LLM APIs from the client side with embedded API keys.

Additionally, we analyzed the user rating counts of vulnerable applications to assess the real-world impact of API key leakage, with details presented in Figure 3. The mean rating count for vulnerable apps is 618.9, while the median stands at 30.0. This substantial discrepancy between mean and median values indicates that the majority of vulnerable applications have a low number of user ratings (i.e., niche or less widely adopted Apps), yet a subset of widely used applications, even with thousands of user ratings, also suffer from LLM API key leakage. Our observation shows that the key leakage issue extends to popular iOS applications.

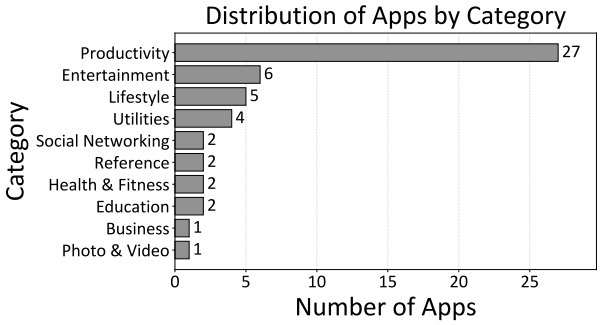


Fig. 2. Distribution of LLM-integrated apps by category.

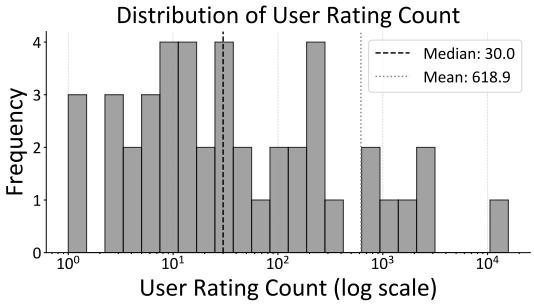


Fig. 3. Distribution of user rating counts for vulnerable apps (log scale)

B. Ongoing Work

Our ongoing work includes developing an automated pipeline to test and verify LLM API key leakage in iOS applications. This pipeline will enable us to scale our analysis to a larger number of applications. Furthermore, we are investigating practical defense mechanisms against LLM API key leakage. Our goal is to translate our empirical findings into actionable guidelines for developers to reduce the risk of key exposure.

References

[1] X. Hou, Y. Zhao, and H. Wang, "On the (in) security of llm app stores," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 317–335.

[2] A. Brucato, "LLMjacking: Stolen cloud credentials used in new AI attack," Sysdig Blog, May 2024, accessed: 2025-01-18. [Online]. Available: https://www.sysdig.com/blog/llmjacking-stolen-cloud-credentials-used-in-new-ai-attack

[3] Y. Liu, G. Deng, Y. Li, K. Wang, Z. Wang, X. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng *et al.*, "Prompt injection attack against llm-integrated applications," *arXiv preprint arXiv:2306.05499*, 2023.

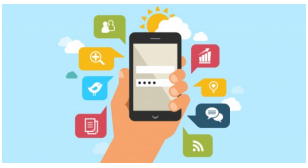
[4] M. Ibrahim, G. S. Tuncay, Z. B. Celik, A. Machiry, and A. Bianchi, "LM-Scout: Analyzing the security of language model integration in android apps," *arXiv preprint arXiv:2505.08204*, 2025.

Your Key is Mine: LLM API Key Leakage in iOS Apps

Pinran Gao[1], Wang Lingxiang [2], Ying Zhang[1]
[1]Department of Computer Science, Wake Forest University, [2]Unafflitated
{gaop24,ying.zhang}@wfu.edu, lingxiang.wang.2016@gmail.com

1. Motivation

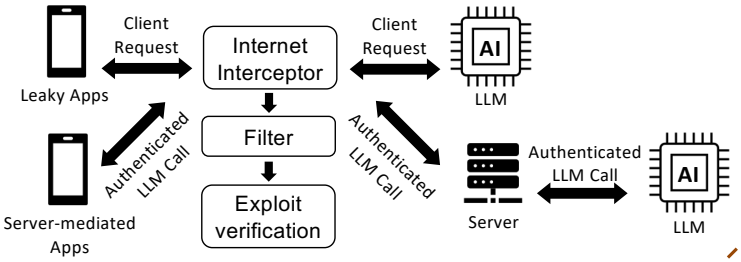
- Rapid emergence of LLM powered iOS Apps [1].
- Apps with insecure LLM integrations could lead to potential key leakages, and possible financial loss [2].
- Prior research has studied LLM-related security issues in web-based Apps or Android-specific scenarios[3].
- LLM API key leakage detection in iOS is needed.



2. Attack Model

Vulnerabilities Analysis

- LLM API key is directly embedded in request headers, allowing interception and potential abuse.
- Moreover, the vulnerable applications integrate LLM with no authentication.
- The lack of standardized guidelines for secure LLM API invocation contributes to inconsistent and unsafe integration practices.

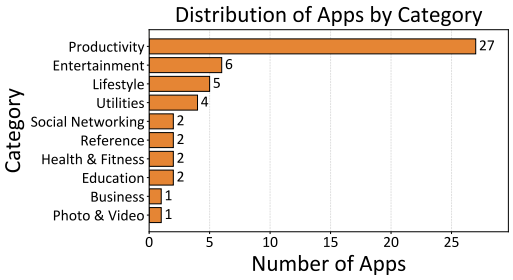


3. Dataset Collection

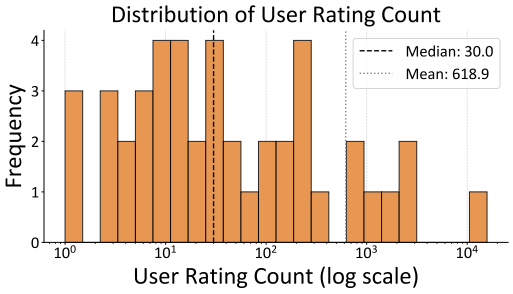
Crawling and Filtering → Download and filtering → Manual test and validation

- Define sets of AI model names, chat-related terms, and known AI chatbot brands. Use them as keywords in search for iOS Apps from AppStore and perform deduplication.
- After filtering and deduplication, 500 LLM-related Apps are collected.
- Hired 6 testers manually test each App individually, then validate the LLM API key leakage they get in the process.

4. Initial Results



- Leakages span across various categories among iOS Apps, especially on productivity.



- Leakages take place mostly in Apps with low user rating count, yet some appears in Apps with thousands rates.

5. Challenges

Lack of empirical study

Existing studies largely focus on program analysis, while comprehensive empirical measurements of LLM API key leakage in real-world Apps remain limited. The impact and the scope of the leakage remains unknown.

Testing scalability

Systematically testing and verifying LLM-based Apps to demonstrate LLM API key leakage requires extensive manual effort, limiting the scalability and reproducibility of large-scale analysis.

Secure coding in LLM Apps

Lack of security standards leads to inconsistent and insecure LLM integrations with mobile App.

6. Ongoing Works

- Building automating tool that captures network traffic and detects LLM API key leakage during runtime
- Developing iOS UI automation tool to systematically trigger LLM-related interactions in mobile.
- Investigating practical defense mechanisms and secure deployment strategies to mitigate LLM API key leakages among Apps.

Reference:
[1] Hou, Xinyi, Yanjie Zhao, and Haoyu Wang. "On the (in) security of llm app stores." 2025 IEEE Symposium on Security and Privacy (S&P). IEEE, 2025.
[2] Zhang, Yue, Yuqing Yang, and Zhiqiang Lin. "Don't leak your keys: Understanding, measuring, and exploiting the appsecret leaks in mini-programs." Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023.
[3] Schmidt, David, Sebastian Schrittwieser, and Edgar Weippl. "Leaky Apps: Large-scale Analysis of Secrets Distributed in Android and iOS Apps." Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. 2025.
[4] Mobile Application Development Illustration." Yeeply, Yeeply, https://yeeply.com/en/blog/mobile-app-development/professionals-for-mobile-application-development/. Accessed 17 Jan. 2026.
[5] Gemini vs. ChatGPT Comparison Graphic." Zapier, Zapier Inc., www.zapier.com/blog/gemini-vs-chatgpt/. Accessed 17 Jan. 2026.

NDSS SYMPOSIUM