Potential Risks Arising from the Absence of Signature Verification in Miniapp Plugins

Yanjie Zhao
Yanjie.Zhao@monash.edu
Monash University
Australia

Yue Zhang
yz899@drexel.edu
Drexel University
USA

Haoyu Wang
haoyuwang@hust.edu.cn
Huazhong University of Science and Technology
China

ABSTRACT

The advent of mobile super apps has given rise to the miniapp paradigm, a lightweight application model that operates within a JavaScript engine hosted by the primary app. Miniapps now have transformed the app ecosystem, offering easy access, install-less functionality, and a wide array of service offerings. However, the integration of plugins, which are functional components added to miniapps, has introduced potential security concerns. While the underlying framework strives to ensure data security between miniapps and their embedded plugins, vulnerabilities may arise if signature verification is neglected in the plugin's implementation. Although Tencent offers developers guidelines for signature integration, this verification isn't pre-packaged, potentially leading less experienced developers to skip it when incorporating plugins, risking security. Specifically, the lack of signature verification in miniapp plugins can create a potential threat, enabling attackers to manipulate transactions and undermine the integrity of the miniapp. This paper explores the communication mechanisms of miniapps, the function of plugins, and the vital role of signature verification in enhancing the security of transactions and data within this rapidly evolving ecosystem.

CCS CONCEPTS

• **Security and privacy** → *Software security engineering*.

KEYWORDS

Mini-programs, miniapps, mobile super apps, mobile security

ACM Reference Format:
Yanjie Zhao, Yue Zhang, and Haoyu Wang. 2023. Potential Risks Arising from the Absence of Signature Verification in Miniapp Plugins. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3605762.3624433

1 INTRODUCTION

In the ever-evolving landscape of mobile applications, the emergence of super apps like WeChat, Baidu, and TikTok has marked a significant shift in how users interact with digital platforms [4]. These platforms, seeking to leverage their vast user base and expand their functionalities, have introduced the concept of the miniapp [39]. First debuted by WeChat in 2017 [9], miniapps are lightweight, full-fledged applications that run within a JavaScript engine created or virtualized by the host app. This innovative approach offers users a plethora of daily-life services, from ride-hailing to online shopping, all without the traditional installation process.

The allure of miniapps lies in their ease of development and distribution. Unlike traditional web apps, which often necessitate developer-maintained backend servers, miniapps provide a set of encapsulated APIs, granting easy access to data and system resources maintained by the super app [36]. Furthermore, their distribution extends beyond app stores, leveraging the super app's social network, thereby making access to a new miniapp as simple as a click.

Integral to the miniapp ecosystem are plugins [6]. Miniapp plugins, as modular and reusable components, are crafted to augment and diversify miniapp functionalities. Platforms like WeChat support these plugins, which comprise a blend of APIs, custom components, or pages, allowing for seamless integration into miniapps. Distinct from standalone miniapps, plugins are dependent entities, requiring a miniapp for operation. This architecture enables third-party developers to encapsulate specific features or services, streamlining their integration across various miniapps. Additionally, the inherent design of plugins ensures their code remains concealed from the host miniapp, preserving intellectual property and maintaining data security.

Unfortunately, the inception of these plugins, which were primarily conceived to amplify the capabilities of miniapps, has inadvertently opened the door to a host of security concerns. While these plugins promise a suite of extended services, they simultaneously demand rigorous security protocols to uphold the integrity of the miniapp and safeguard its vast user base. One particularly conspicuous concern that has emerged is the potential for security breaches stemming from the lack of signature verification within these plugins. To delve into further specifics, it's worth noting that while Tencent has furnished developers with guidelines [1] for integrating signature verification, this verification process isn't readily available in a pre-packaged form. Consequently, developers with limited expertise might unintentionally overlook these instructions and proceed to incorporate plugins into their creations without incorporating the essential signature verification measures.

The absence of such critical verification may create vulnerabilities, threatening the users' security and privacy. Attackers, capitalizing on these lapses, might exploit the system, orchestrating maneuvers that lead to undue financial gains. To delve deeper into

SaTS '23, November 26, 2023, Copenhagen, Denmark
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0258-7/23/11…$15.00
https://doi.org/10.1145/3605762.3624433

the specifics, bypassing or neglecting this verification process can expose the system to a myriad of vulnerabilities, granting attackers the leeway to alter data and manipulate transactions. A tangible example of this would be the potential compromise of a plugin with payment functionality. In such a scenario, a miniapp could be duped into recognizing and validating a payment transaction that, unbeknownst to it, has been maliciously altered. Imagine a situation where the miniapp is led to believe a user has executed a substantial payment, while in actuality, the transaction amount has been cunningly tampered with by malevolent entities.

Contribution. Our paper endeavors to provide an in-depth analysis of the miniapp paradigm, the role and functionality of plugins, and the paramount importance of signature verification in safeguarding the integrity and security of the miniapp ecosystem.

2 BACKGROUND

2.1 Miniapps vs. Miniapp Plugins

Miniapps. Miniapps [39] are streamlined applications that function within a host app, offering instant access to various services without requiring separate installation. They serve as a bridge between full applications and quick, on-the-go services.

```
var myPluginInterface = requirePlugin('myPlugin');
myPluginInterface.hello();
var myWorld = myPluginInterface.world;
```

Listing 1: An example of calling JS APIs from a plugin.

Miniapp Plugins. Miniapp plugins [6] are specialized components that enhance miniapps by adding specific functionalities or APIs. They enable developers to efficiently incorporate features like payment processing or mapping. The integration of these plugins often involves a verification process to ensure the authenticity and integrity of the communication between the miniapp and the plugin. This verification process, typically involving signature checks, is crucial in maintaining the security and reliability of the system, especially in scenarios involving sensitive operations.

Table 1: The comparison between miniapps and miniapp plugins.

Attribute	Miniapp	Miniapp plugin
Definition	Lightweight app	Package of APIs
Run Mode	Inside host app	Within miniapp
Functionality	Broad services	Specific enhancements
API/Component Access	Broad access	Limited/specific access
User Interaction	Direct interaction	Indirect/none
Development Tools	WeChat DevTools	Similar to Miniapps

Comparison. As detailed in Table 1, miniapps and miniapp plugins exhibit distinct differences across multiple dimensions.

- **Definition:** Miniapps are lightweight applications designed to run inside a host app, offering various services without the need for separate installations. In contrast, miniapp plugins are specialized packages, often comprising specific APIs or components, crafted to augment the functionalities of the miniapp into which they are integrated. Listing 1 illustrates this relationship, showing an example of how a JavaScript API from a plugin named *myPlugin* can be invoked in a miniapp. By utilizing the *requirePlugin* method, developers can seamlessly integrate the plugin's functionality, such as a *hello* method or a *world* variable, into the broader application environment.
- **Run Mode:** While miniapps operate within a host app, they maintain a semblance of independence, running their own set of operations and services. Plugins, however, are inherently dependent. They need to be embedded within a miniapp to function, acting more as extensions than individual entities.
- **Functionality:** Miniapps are versatile, offering users a broad spectrum of services, from shopping to ride-hailing. Plugins have a narrower focus. They provide specific features or services, like payment gateways or map integrations, that a miniapp might need.
- **API/Component Access:** In WeChat API, there's a clear distinction between access permissions for miniapps and plugins. Miniapps have broader access to various APIs, while plugins have more limited access. Plugins cannot access some APIs, and they have their own domain name list. Certain functions not in the "wx" object and specific APIs are only available on feature pages within plugins. Plugins also have restrictions on using components like buttons with specific open types (e.g., "contact", "getPhoneNumber", "getUserInfo") during development. They cannot use components like "open-data" for cross-app communication and "web-view" to access miniapp pages. This limitation can lead to the plugin's backend being unaware of the frontend's status [2, 3, 7].
- **User Interaction:** Miniapps are designed for direct user interaction. They are the interfaces that users see, click on, and engage with. On the contrary, plugins typically operate in the background. Their code and operations remain largely invisible to the end-user, silently enhancing the miniapp's capabilities.
- **Development Tools:** Both miniapps and plugins utilize similar development tools such as WeChat DevTools [5], but plugins are specifically tailored to enhance miniapps' functionalities.

2.2 Miniapp-plugin Communication

In the context of utilizing plugins, miniapps have to establish a communication channel with the plugins to facilitate various functionalities. This channel is vital for the seamless integration of additional features and services that enhance the user experience. As illustrated in Figure 1, consider a miniapp engaged in an online shopping scenario. Here, the miniapp's backend returns a product price of 100 units, prompting the miniapp to initiate a payment request through the plugin ❶, i.e., *myPluginInterface.pay(100)*, asking the user to pay the specified amount. To complete this payment process, the miniapp leverages a plugin designed to handle transactions. This plugin acts as a mediator, processing the payment

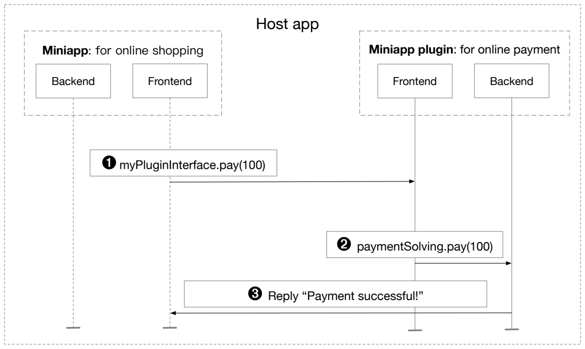
Host app
Miniapp: for online shopping
Backend
Frontend
Miniapp plugin: for online payment
Frontend
Backend
❶ myPluginInterface.pay(100)
❷ paymentSolving.pay(100)
❸ Reply "Payment successful!"

Figure 1: An example of the synergy between miniapps and plugins in payment scenarios.

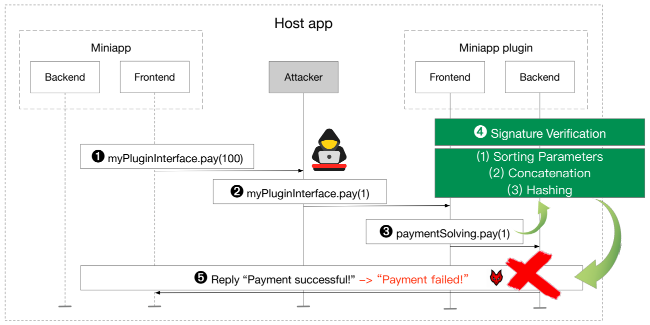

Figure 2: A protected payment process with signature verification.

request and interacting with the payment gateway ❷. Ideally, once the consumer finalizes the payment, the plugin communicates a successful payment message back to the miniapp ❸. Armed with this confirmation, the miniapp then proceeds to the next steps, such as shipping the purchased item or handling potential refunds.

This synergy between miniapps and plugins not only streamlines the payment process but also allows for the integration of various other services, such as location tracking, IoT device controlling [40], social sharing, or personalized recommendations. The flexibility and modularity of this system enable developers to create rich, dynamic experiences without overcomplicating the core miniapp structure.

3 SECURE SIGNATURE MECHANISM

While plugins offer enhanced functionalities, they also introduce potential security concerns. Ensuring the integrity and authenticity of network requests is vital to prevent malicious activities such as spoofing or tampering. Hence, developers are guided by Tencent [2] to implement a robust digital signature mechanism. Broadly, this mechanism can be divided into two primary phases:

(I) Signature Generation. When a plugin sends a network request via APIs such as *wx.request*, the request will additionally carry a signature to verify that the request is sent from a miniapp plugin. This signature is located in the request header and looks like as below:

```
X-WECHAT-HOSTSIGN: {"noncestr":"NONCESTR",
↪  "timestamp":"TIMESTAMP", "signature":
↪  "SIGNATURE"}
```

- ***NONCESTR***: A random string that introduces unpredictability into the signature. This ensures that even if the other parameters remain constant, the signature will differ with each request, making it resistant to replay attacks.
- ***TIMESTAMP***: A UNIX timestamp that provides a temporal context to the request. This ensures that requests are time-bound, adding another layer of security against potential attacks.

Here, *NONCESTR* and *TIMESTAMP* are used to generate *SIGNATURE*. The algorithm for calculating the signature is:

```
SIGNATURE = sha1([APPID, NONCESTR, TIMESTAMP,
↪  TOKEN].sort().join(''))
```

APPID is the AppId (which can be obtained from the referrer in the request header) of the miniapp where a plugin resides, and *TOKEN* is the token of the plugin:

- ***APPID***: This serves as the unique identifier for the miniapp. It ensures that the request is associated with a specific application within the WeChat ecosystem.
- ***TOKEN***: Specific to the plugin, this token acts as a secret key, ensuring that only legitimate plugins can generate valid signatures.

(II) Signature Verification. The verification process is a critical aspect of ensuring the integrity and authenticity of the communication between miniapps and plugins. Plugin developers can verify a signature by performing the following steps on their server:

(1) **Sorting Parameters.** The first step in the verification process involves sorting the values of *APPID*, *NONCESTR*, *TIMESTAMP*, and *TOKEN* in lexicographical order. This order is consistent with the sorting order of JavaScript arrays. By arranging these parameters in a specific sequence, a standardized format is established, which is essential for the subsequent verification process.
(2) **Concatenation.** Once sorted, the four strings are concatenated directly. This concatenation forms a unique string that represents the specific request. The concatenation ensures that the signature is sensitive to the exact values and order of the parameters, making it a robust mechanism against tampering.
(3) **Hashing.** The concatenated string is then hashed using the *sha1* algorithm. This cryptographic hash function takes the concatenated string and produces a fixed-size hash value, referred to as the *SIGNATURE*. The resulting *SIGNATURE* serves as a fingerprint of the original request, encapsulating the critical information in a form that is difficult to forge. If any part of the original request is altered, the signature will change, allowing the server to detect the modification.

As illustrated in Figure 2, once the plugin has undergone the signature verification process ❹, any tampering with the message by an attacker ❷ will be detected by the plugin. This detection leads to the plugin returning a payment failure result ❺, effectively thwarting the attack. The signature verification process thus plays a crucial role in maintaining the security of the communication between miniapps and plugins, providing a robust defense against spoofing and tampering attacks. By implementing this verification process, developers can enhance the trustworthiness of their plugins and protect both users and merchants from potential fraud. This process is particularly crucial in scenarios where sensitive operations such as payments or personal data retrieval are involved. Without proper verification, an attacker could potentially modify the request parameters, leading to unauthorized actions or information disclosure.

4 SPOOFING ATTACK

As discussed in §3, the significance of signatures in miniapp plugin security is evident. Yet, developers lacking expertise might inadvertently disregard instructions, leading them to integrate plugins without the crucial signature verification measures. In the absence of signature verification, this creates a window of opportunity for malicious users to manipulate messages during the communication process, consequently enabling **spoofing attacks** [8].

4.1 Threat Model and Scope

Assumptions. We assume code integrity for the host app and miniapp's frontend, and trust in the encrypted communication between the miniapp and the plugin. The plugin may be untrusted if lacking signature verification. The spoofing attack can succeed primarily due to the absence of mandatory signature verification in the plugin. An attacker can manipulate the communication between the miniapp and the plugin, altering the content without detection if the plugin does not verify the signature.

Scope and Goals. In the context of this comprehensive study, our primary focus is directed towards the examination of miniapps that make use of plugins, specifically those that do not incorporate mandatory signature verification processes. This particular aspect of our investigation holds substantial significance, particularly within environments where these plugins are developed by external third parties. It is crucial to highlight that in such scenarios, there often exists a notable absence of enforced standards or rigorous protocols pertaining to signature verification. This absence of a standardized verification process can potentially introduce vulnerabilities and security concerns, which makes it a subject of paramount importance for our research. By honing in on this specific issue, we aim to shed light on the potential risks and security implications associated with the utilization of plugins lacking mandatory signature verification, ultimately contributing to a deeper understanding of security challenges in these environments.

4.2 Attack Workflow

Overview. The absence of a robust signature verification mechanism within miniapp plugins leaves a concerning vulnerability open to exploitation by malicious actors. In this scenario, attackers can leverage these vulnerabilities to carry out a range of potentially harmful attacks. One such attack vector involves the manipulation of timestamps, which can lead to what is known as a "command re-execute attack." In this type of attack, attackers can tamper with the timestamp associated with a plugin, effectively tricking it into accepting fraudulent requests. By exploiting this weakness, they can force the plugin to re-execute commands, potentially compromising the integrity and security of the miniapp.

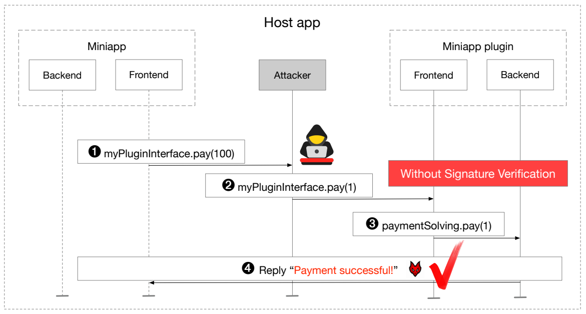
Host app
Miniapp
Backend
Frontend
Attacker
Miniapp plugin
Frontend
Backend
❶ myPluginInterface.pay(100)
Without Signature Verification
❷ myPluginInterface.pay(1)
❸ paymentSolving.pay(1)
❹ Reply "Payment successful!"

Figure 3: An example of the spoofing attacks during the payment process within miniapps.

Furthermore, the absence of mandatory signature verification also opens the door to another insidious threat: data manipulation attacks. In these attacks, attackers can tamper with the hash values associated with the data being transmitted between the miniapp and the plugin. By altering the hash value, they can effectively modify the content of messages exchanged between the two entities. This manipulation can lead to data corruption, unauthorized access, or the injection of malicious payloads, ultimately compromising the confidentiality and integrity of the information being processed.

To provide a more detailed understanding of these potential attacks, let's delve into two illustrative examples, demonstrating the real-world implications of the vulnerabilities arising from the lack of signature verification within miniapp plugins.

- **Data Manipulation Attack.** In this scenario, the attacker assumes the role of a malicious user, aiming to manipulate the data exchanged between the miniapp plugin and its server. To achieve this, the attacker must modify the content and regenerate a hash value (referred to as the `signature` field in the message). A malicious customer or attacker could exploit vulnerabilities in the communication between the miniapp and the plugin as shown in Figure 3. For example, if the attacker intercepts the message sent from the miniapp to the plugin ❶ and alters the amount from 100 units to 1 unit ❷, and the plugin itself lacks signature verification, it may unknowingly fall victim to a spoofing attack. The plugin would then treat the attacker's message as if it were legitimately from the miniapp. Remember that in §2.1 we mentioned the plugin itself cannot read the content of the miniapp's page, and therefore relies solely on this communication channel to interact with the miniapp. Once the consumer completes the payment of 1 unit ❸, the plugin sends a successful payment message back to the miniapp ❹. The attacker's fraudulent scheme succeeds, resulting in a loss of 99 units for the merchant behind the miniapp. This example underscores the importance of robust security measures, such as signature verification, to prevent spoofing attacks and ensure the integrity of communications between miniapps and plugins.

- **Command Re-execute Attack.** A command re-execute attack involves an attacker intercepting and then resending a message that was initially transmitted from a miniapp plugin to its server. The attacker's aim is to deceive the server into believing that the retransmitted message is a legitimate and fresh communication from the miniapp plugin. Because the miniapp plugin's server fails to validate both the `timestamp` and the signature, an attacker gains the ability to compel the server to re-execute a previously executed command (e.g., shipping a product that is shipped before). Given the similarity in the attack workflow to the first scenario, we omitted the details for conciseness.

5 RELATED WORK

Miniapp and Superapp Security. Existing studies on miniapps mainly concentrate on their architecture and applications. For instance, Hao et al. [16] explored the system architecture of WeChat miniapps, while others have investigated their applications in various domains such as healthcare [32, 44] and education [12, 19, 28]. Some works have also uncovered vulnerabilities in miniapps and super apps [24, 27, 30, 31, 37, 38, 41]. Unlike these studies, we are the first to study the security of miniapp plugins. Our research specifically targets potential risks and mitigation strategies associated with the absence of mandatory signature verification.

Mobile Security. Research in the mobile security domain is characterized by a sustained focus on identifying and mitigating vulnerabilities that could potentially undermine system reliability and data integrity [15, 18, 21, 22, 26, 29, 42]. A plethora of studies have delved into various aspects such as application security [17, 23, 43], privacy preservation [13, 14, 35], and secure communication [25, 33, 34]. Distinct from these studies, our work narrows down to a specialized investigation in the miniapp domain.

Vulnerability Discovery in Online Services. Extensive efforts have been made to detect vulnerabilities in online services. These efforts involve two approaches: analyzing server code (white-box) and examining network traffic (black-box). Additionally, there's a focus on access control issues, primarily concerning authentication problems like single sign-on security (e.g., [20]), OAuth (e.g., [11]), and authentication vulnerability scanning (e.g., [10]). In contrast to previous work, our research centers on the miniapp domain, where the host app has already completed a portion of the authentication process, leaving the miniapps to handle the remainder. This represents a distinct approach compared to traditional efforts.

6 DISCUSSION

Countermeasures. We strongly urge platforms like WeChat to make signature verification mandatory to enhance the overall security of the miniapp ecosystem. This enforcement will establish consistent security protocols, reducing the risk of exploitation resulting from individual developers' inconsistent implementations.

While it's true that client-side signature generation is susceptible to attacks, where attackers can reverse engineer and manipulate client code to create valid signatures, it's important to note that we view this type of attack as unrealistic given our trust in both the client and the mobile environment. However, to ensure absolute authenticity, developers can consider using trusted hardware like TrustZone in addition to signature verification.

Ethical Concern. In this study, we identified potential vulnerabilities within the miniapp and plugin communication system but refrained from conducting live experiments due to ethical concerns. In real-world testing scenarios, one could simulate real-world conditions by sending malevolent network requests to the servers of miniapp plug-ins. This action could potentially undermine the privacy and security of stakeholders, resulting in unintended repercussions. Therefore, our research emphasized theoretical analysis and simulated scenarios, adhering to a responsible approach that respects the rights and interests of all involved. Our team is collaborating closely with Tencent to actively explore viable solutions to the issue.

Future Work. In future studies, we aim to extend our research scope to encompass not just miniapp plugins but also ordinary miniapps to explore the broader landscape of security concerns and potential countermeasures. This expansion will involve a detailed analysis of the existing security mechanisms and their effectiveness in safeguarding user data and ensuring app reliability. Moreover, while our current work has primarily focused on signature verification as a security measure, we acknowledge its limitations and foresee a comprehensive investigation into alternative security measures. Drawing from the insights of our current study, we plan to delve deeper into mutual authentication mechanisms, exploring the potential role of Public Key Infrastructure (PKI) in enhancing the security of super-app platforms.

7 CONCLUSION

This paper has explored the intricate communication mechanisms between miniapps and plugins, shedding light on both the potential vulnerabilities and the security measures that can be implemented. A significant part of the security issue stems from the fact that signature verification is not mandatory but is left to the discretion of the plugin developers. By delving into an example of spoofing attacks, we have illustrated how attackers can exploit weaknesses in the system, and how signature verification can effectively thwart such attempts. The insights provided in this study underscore the importance of rigorous and standardized security protocols in the rapidly evolving miniapp ecosystem.

ACKNOWLEDGEMENTS

We thank shepherd Juanru Li as well as the anonymous reviewers for their insightful comments. This work was supported in part by the National Key R&D Program of China (2021YFB2701000), the National Natural Science Foundation of China (grant No.62072046), the Knowledge Innovation Program of Wuhan-Basic Research, and HUST CSE-HongXin Joint Institute for Cyber Security.

REFERENCES

[1] [n. d.]. Feature Page of Payment | Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/framework/plugin/functional-pages/request-payment.html. [Accessed 10-08-2023].

[2] [n. d.]. Feature Page of Plug-Ins | Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/framework/plugin/functional-pages.html. [Accessed 10-08-2023].

[3] [n. d.]. Limitation on Calling APIs via Plug-Ins | Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/framework/plugin/api-limit.html. [Accessed 10-08-2023].

[4] [n. d.]. Mobile payments, mini-programs are key features of Chinese super apps — spglobal.com. https://www.spglobal.com/marketintelligence/en/news-insights/research/mobile-payments-mini-programs-are-key-features-of-chinese-super-apps. [Accessed 10-08-2023].

[5] [n. d.]. Overview | Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/devtools/devtools.html. [Accessed 10-08-2023].

[6] [n. d.]. Plug-in | Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/framework/plugin/. [Accessed 10-08-2023].

[7] [n. d.]. Restrictions on the Use of Components by Plug-Ins | Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/framework/plugin/component-limit.html. [Accessed 10-08-2023].

[8] [n. d.]. Spoofing attack - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Spoofing_attack. [Accessed 10-08-2023].

[9] [n. d.]. Weixin public doc — developers.weixin.qq.com. https://developers.weixin.qq.com/miniprogram/en/dev/framework/. [Accessed 10-08-2023].

[10] Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, and Jin Song Dong. 2013. Authscan: Automatic extraction of web authentication protocols from implementations. (2013).

[11] Eric Y Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. 2014. Oauth demystified for mobile application developers. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 892–903.

[12] Xin Chen, Xi Zhou, Huan Li, Jinlan Li, and Hua Jiang. 2020. The value of WeChat as a source of information on the COVID-19 in China. *Preprint]. Bull World Health Organ* 30 (2020).

[13] Zikan Dong, Liu Wang, Hao Xie, Guoai Xu, and Haoyu Wang. 2022. Privacy Analysis of Period Tracking Mobile Apps in the Post-Roe v. Wade Era. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–6.

[14] Jie Gu, Yunjie Calvin Xu, Heng Xu, Cheng Zhang, and Hong Ling. 2017. Privacy concerns for mobile app download: An elaboration likelihood model perspective. *Decision Support Systems* 94 (2017), 19–28.

[15] Xing Han, Yuheng Zhang, Xue Zhang, Zeyuan Chen, Mingzhe Wang, Yiwei Zhang, Siqi Ma, Yu Yu, Elisa Bertino, and Juanru Li. 2023. Medusa Attack: Exploring Security Hazards of {In-App} {QR} Code Scanning. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4607–4624.

[16] Lei Hao, Fucheng Wan, Ning Ma, and Yicheng Wang. 2018. Analysis of the development of WeChat mini program. In *Journal of Physics: Conference Series*, Vol. 1087. IOP Publishing, 062040.

[17] Yiling He, Yiping Li, Lei Wu, Ziqi Yang, Kui Ren, and Zhan Qin. 2023. MsDroid: Identifying Malicious Snippets for Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing* 20, 3 (2023), 2025–2039. https://doi.org/10.1109/TDSC.2022.3168285

[18] Pingfan Kong, Li Li, Jun Gao, Timothée Riom, Yanjie Zhao, Tegawendé F Bissyandé, and Jacques Klein. 2021. ANCHOR: locating android framework-specific crashing faults. *Automated Software Engineering* 28 (2021), 1–31.

[19] Qinzhen Liang and Chengyang Chang. 2019. Construction of teaching model based on WeChat Mini-Program. *International Journal of Science* 16, 1 (2019), 54–59.

[20] Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. 2008. Automatic protocol format reverse engineering through context-aware monitored execution.. In *NDSS*, Vol. 8. 1–15.

[21] Pei Liu, Yanjie Zhao, Haipeng Cai, Mattia Fazzini, John Grundy, and Li Li. 2022. Automatically Detecting API-induced Compatibility Issues in Android Apps: A Comparative Analysis (Replicability Studies). In *The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*.

[22] Pei Liu, Yanjie Zhao, Mattia Fazzini, Haipeng Cai, John Grundy, and Li Li. 2023. Automatically Detecting Incompatible Android APIs. *ACM Transactions on Software Engineering and Methodology* (2023). https://doi.org/10.1145/3624737

[23] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. 2022. Explainable ai for android malware detection: Towards understanding why the models perform so well?. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 169–180.

[24] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying resource management risks in emerging mobile app-in-app ecosystems. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security*. 569–585.

[25] Siqi Ma, Juanru Li, Hyoungshick Kim, Elisa Bertino, Surya Nepal, Diethelm Ostry, and Cong Sun. 2021. Fine with "1234"? An analysis of SMS one-Time password randomness in android apps. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1671–1682.

[26] Siqi Ma, Juanru Li, Surya Nepal, Diethelm Ostry, David Lo, Sanjay Kumar Jha, Robert H Deng, and Elisa Bertino. 2021. Orchestration or automation: authentication flaw detection in android apps. *IEEE Transactions on Dependable and Secure Computing* 19, 4 (2021), 2165–2178.

[27] Meng Shi, Liu Wang, Shenao Wang, Kailong Wang, Xusheng Xiao, Guangdong Bai, and Haoyu Wang. 2023. WeMinT: Tainting Sensitive Data Leaks in WeChat Mini-Programs. In *ASE*.

[28] Yiling Sui, Tian Wang, and Xiaochun Wang. 2020. The impact of WeChat app-based education and rehabilitation program on anxiety, depression, quality of life, loss of follow-up and survival in non-small cell lung cancer patients who underwent surgical resection. *European Journal of Oncology Nursing* 45 (2020), 101707.

[29] Xiaoyu Sun, Xiao Chen, Yanjie Zhao, Pei Liu, John Grundy, and Li Li. 2022. Mining android api usage to generate unit test cases for pinpointing compatibility issues. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[30] Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. [n. d.]. TAINT-MINI: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis. ([n. d.]).

[31] Chao Wang, Yue Zhang, and Zhiqiang Lin. [n. d.]. Uncovering and Exploiting Hidden APIs in Mobile Super Apps. https://doi.org/10.48550/arXiv.2306.08134 arXiv:2306.08134 [cs]

[32] Feilong Wang, Lily Dongxia Xiao, Kaifa Wang, Min Li, and Yanni Yang. 2017. Evaluation of a WeChat-based dementia-specific training program for nurses in primary care settings: A randomized controlled trial. *Applied Nursing Research* 38 (2017), 51–59.

[33] Kailong Wang, Junzhe Zhang, Guangdong Bai, Ryan Ko, and Jin Song Dong. 2021. It's Not Just the Site, It's the Contents: Intra-domain Fingerprinting Social Media Websites Through CDN Bursts. In *30th The Web Conference (WWW)*. https://doi.org/10.1109/ICECCS2018.2018.00011

[34] Kailong Wang, Yuwei Zheng, Qing Zhang, Guangdong Bai, Qin Mingchuang, Donghui Zhang, and Jin Song Dong. Accepted by MobiCom 2022. Assessing Certificate Validation User Interfaces of WPA Supplicants.

[35] Maria K Wolters, Shuobing Li, Haoyu Wang, Xinyu Yang, and Yao Guo. 2020. Does the Presence of Privacy Relevant Information Affect App Market Choice?. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–7.

[36] Yuqing Yang, Chao Wang, Yue Zhang, and Zhiqiang Lin. [n. d.]. SoK: Decoding the Super App Enigma: The Security Mechanisms, Threats, and Trade-offs in OS-alike Apps. arXiv:2306.07495 [cs] http://arxiv.org/abs/2306.07495

[37] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. [n. d.]. Cross Miniapp Request Forgery: Root Causes, Attacks, and Vulnerability Detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles CA USA, 2022-11-07). ACM, 3079–3092. https://doi.org/10.1145/3548606.3560597

[38] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. 2022. Identity confusion in {WebView-based} mobile app-in-app ecosystems. In *31st USENIX Security Symposium (USENIX Security 22)*. 1597–1613.

[39] Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin. [n. d.]. A Measurement Study of Wechat Mini-Apps. 5, 2 ([n. d.]), 1–25. https://doi.org/10.1145/3460081

[40] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. 2020. Breaking secure pairing of bluetooth low energy using downgrade attacks. In *29th USENIX Security Symposium (USENIX Security 20)*. 37–54.

[41] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. [n. d.]. Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs. https://doi.org/10.48550/arXiv.2306.08151 arXiv:2306.08151 [cs]

[42] Yanjie Zhao, Li Li, Kui Liu, and John Grundy. 2022. Towards Automatically Repairing Compatibility Issues in Published Android Apps. In *The 44th International Conference on Software Engineering (ICSE 2022)*.

[43] Yanjie Zhao, Li Li, Haoyu Wang, Haipeng Cai, Tegawendé F Bissyandé, Jacques Klein, and John Grundy. 2021. On the impact of sample duplication in machine-learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–38.

[44] Kaina Zhou, Wen Wang, Wenqian Zhao, Lulu Li, Mengyue Zhang, Pingli Guo, Can Zhou, Minjie Li, Jinghua An, Jin Li, et al. 2020. Benefits of a WeChat-based multimodal nursing program on early rehabilitation in postoperative women with breast cancer: a clinical randomized controlled trial. *International journal of nursing studies* 106 (2020), 103565.