A Small Leak Will Sink Many Ships: Vulnerabilities Related to mini-programs Permissions

1st Jianyi Zhang
Beijing Electronic Science and Technology Institute
Beijing, China
zjy@besti.edu.cn

2nd Leixin Yang
Beijing Electronic Science and Technology Institute
Beijing, China

3rd Yuyang Han
Beijing Electronic Science and Technology Institute
Beijing, China

4th Zixiao Xiang
Beijing Electronic Science and Technology Institute
Beijing, China

5th Xiali Hei
University of Louisiana at Lafayette
Lafayette, US
xiali.hei@louisiana.edu

***Abstract*—As a new format of mobile application, mini-programs, which function within a larger app and are built with HTML, CSS, and JavaScript web technology, have become the way to do almost everything in China. Many researchers have done the ecosystem or developing study, while the permission problem has not been investigated yet. In this paper, we present our studies on the permission management of mini-programs and conduct a systematic study on 9 popular mobile host app ecosystems that host over 7 million mini-programs. After testing over 2,580 APIs, we extracted a common abstract model for mini-programs' permission control and revealed six categories of potential security vulnerabilities due to improper permission management. It is alarming that the current popular mobile app ecosystems (i.e., host apps) under study have at least one security vulnerability due to the mini-programs' improper permission management. We present the corresponding attack methods to dissect these potential weaknesses further to exploit the discovered vulnerabilities. To prove that the revealed vulnerabilities may cause severe consequences in real-world use, we show three kinds of attacks without privileges or cracking the host apps. We have responsibly disclosed the newly discovered vulnerabilities, and two CVEs were issued. Finally, we put forward systematic suggestions to strengthen the standardization of mini-programs.**

***Index Terms*—mini-programs, mini-apps, permission, API, mobile apps**

I. Introduction

Mini-programs are light applications (commonly 2-4 MB) that run inside a specific mobile app (host app) [1]. As a new mobile application form, leveraging web technologies like HTML, CSS, and JavaScript, mini-programs are taking over China's iOS and Android app ecosystems. The mini-program technology enables the "super app" to bundle features and capabilities into a single mobile native APP, which allows the users never need to leave this native app. We call the native app a host app. Many host app vendors, such as Tencent (WeChat), ByteDance (TikTok), and Alibaba (Alipay), provide their unique framework to support the mini-programs [2]. There are various ways to launch a mini-program in these host apps. Users can scan a QR code, directly search the name in the host app, share with a group or friend, or even launch it with a hyperlink from other mini-programs. As an example, Figure 1 shows how to launch the Tesla mini-program in WeChat. Mini-programs are easy to use, with a clear interface and short loading time. People use mini-programs daily without worrying about installing too many apps. Using a mini-program, a user can complete many tasks like paying bills, playing games, ordering a taxi, booking a doctor's appointment, etc. As of 2022, the number of monthly active users (MAU) of WeChat is over 1.31 billion [3]. TikTok [4] and Alipay [5] has over 1 billion and 668 million MAU, respectively. The total number of mini-program users is close to that of Facebook, the most popular social network worldwide, with about 2.96 billion MAUs [6].

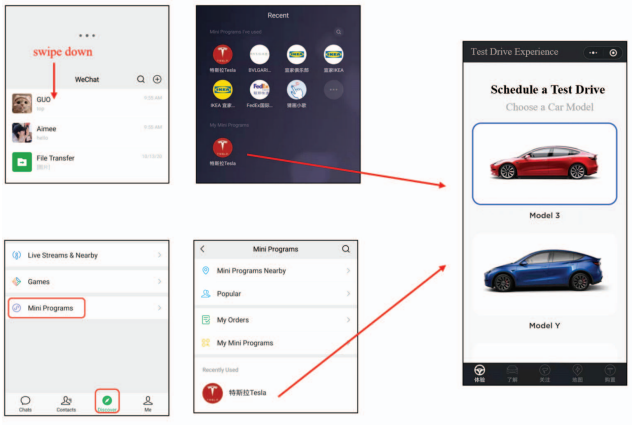

Fig. 1. Tesla mini-program in WeChat. Two ways to access a WeChat mini-program. The one on the upper left shows accessing a mini-program in the WeChat recent menu and the lower left shows the main entry point (Main page-Discover–mini-programs.

Partially supported by the NSF CNS-1650551, OIA-1946231, CNS-2117785, OIA-2229752

979-8-3503-2697-0/23/$31.00 ©2023 IEEE
DOI 10.1109/COMPSAC57700.2023.00085

Since the widespread use of mini-programs, any incorrect permission granting or settings can result in serious security and privacy problems. However, there is not much research focused on this issue. That is not only because it is a new mobile application format, but also, more importantly, the permission structure of mini-programs is entirely different from any other current permission-based security model. As we know, the mobile operating system (OS) is responsible for allowing or denying access to specific resources at the app's run time [7]. The developers should declare a list of permissions that the user must grant before installing or running an application. Then the OS uses this security model to restrict every mobile app, native or hybrid, access to advanced or dangerous functionality on the device [8]. Unlike the current access control models and authorization methods, the permissions of mini-programs are based on the host app authorization. That is, Android or iOS decides whether or not to allow the host apps to have some specific permissions, and the host apps authorize the mini-program to obtain the data. Therefore, from the view of OS, it is impossible to control the permissions of mini-programs directly, meaning mini-programs may request permissions from OS by using the reputation of host apps. Improper management of the mini-program's permission may cause security problems [9].

The permission issues in mobile applications have been thoroughly studied, and its permission management mechanism is relatively complete and formal [10]–[18]. However, the approaches proposed by these studies cannot solve the security issues in the permission control of the mini-programs. The mix of permissions that require user and API authorization results in complex management. Although [19] shows a similar problem, the target and runtime environments are different from mini-programs. [20] and [21] describe the problem of restricting access when the user and OS must both approve. However, the main problem is the user and OS respond to permission requests from different targets. So we cannot utilize their research to solve our problems.

In this paper, we present our systematic studies of the current mini-programs' permission management, where we dissect its framework, ecosystem, and potential vulnerabilities. We refer to the definition of sensitive permission in Android and iOS and conduct a series of examinations on mini-programs. Specifically, we systematically studied 9 popular mobile app ecosystems hosting more than 7 million mini-programs. Then we establish an abstract model for existing mini-programs' unique permission request process. To present a clear approach to discover the vulnerabilities, we define the security principles that the mini-program should be enforced. According to the abstract model and security principles, we investigated more than 2,580 APIs and revealed six categories of potential security vulnerabilities common in most mini-programs we studied. We described three interesting cases of APIs and illegal mini-programs to prove that the exposed vulnerabilities may cause serious consequences on real-world systems. Finally, we have listed security recommendations for mini-programs, developers, and users to mitigate the threat of these vulnerabilities.

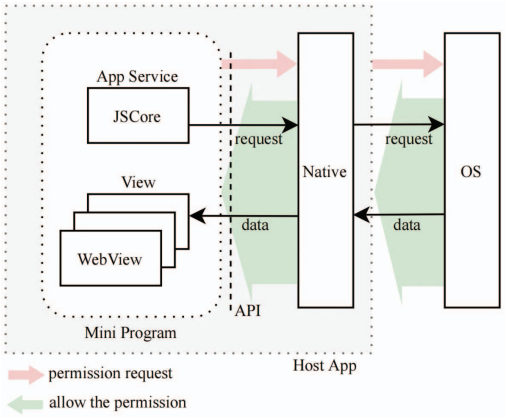

Fig. 2. The common framework of mini-programs. If a mini-program wants to obtain data, it will first request permission from host app, then the host app request permission from OS. After the user grants the permission, the mini-program gets the data from host app by calling APIs.

In summary, we have made the following contributions in this paper:

- We thoroughly examined the current mini-program permissions. Our work deepens the understanding of permission management's complexity of mini-programs. We summarize an abstract model of the mini-program permission control and propose a simple security principle to analyze or discover vulnerabilities more straightforwardly. To the best knowledge, we systematically studied the mini-programs' permission management issues for the first time.
- We have detected more than 2,580 APIs. Through large-scale tracking and analysis of sensitive permission APIs, we have found six categories of potential security vulnerabilities during processing sensitive permission requests from mini-programs.
- We conducted empirical research on the currently popular 9 host apps and revealed the security issues corresponding to the 6 types of potential security vulnerabilities and 3 real-life attacks on the mini-program permissions. Two of them were identified as CVEs.
- To mitigate these potential vulnerabilities, based on the security principles and access control framework, we put forward suggestions to strengthen the standardization of the entire mini-programs' permission, thereby enhancing user privacy.

II. Background

A. Framework of Mini-programs

Mini-programs are a category of applications embedded in host apps without the need for downloads and installation [22]. The flow of mini-program framework consists of two components: View (the rendering layer) and App Service (the logic layer), which are respectively managed by two separate threads, as exhibited in Figure 2. The interface of View is

rendered by the WebView component, which handles page displaying and the user event interaction behavior, while the App Service employs JsCore threads to run JavaScript, for controlling the generation and processing of mini-programs data. The communication between two threads is relayed by the Native app (which refers to the client).

The host app in the framework determines whether the mini-program has permission to access the specific data through the corresponding API. That is, as shown in Figure 2, OS determines whether to allow host apps to have some specific permissions, and host apps then can transmit authorized data to the mini-program through the API. In other words, the permission of mini-programs is inherited from the host app. Hence, if a host app does not properly manage the data and permission, data privacy and security issues will occur in its mini-programs. In this paper, we will focus on the data privacy leakage and wrong permission request authorization issues incurred by improper management of a host app.

B. Mini-programs v.s. Web-based Apps

The mini-programs and their host apps are highly similar to web apps and web browsers. Generally, web apps utilize HTML5 APIs for permission management. It is a unified framework for all web apps under different OS. However, the host app manages the mini-programs' permission. Different vendors have their own APIs and schemes.

Similar to the mini-program, a progressive web app (PWA) also adopts web technology. But differently, a PWA is a type of webpage or website that runs in the browser and can be added to the home screen. Hence, the host environment of the PWA is the browser, and the OS manages the PWA's permission through the browser. Mini-programs can be considered as one type of "Instant" app embedded in host apps. The host environment is a platform with extra capabilities that can support seamless service and access control for the user data.

Google's instant app [23] is very similar to mini-programs. Both of them allow users to access the application's content without additionally installing the application. Thus, the application space on the device could be saved. In essence, Google's instant app is still a native app and under the OS's permission control, while the mini-program is under the host apps' permission control.

C. Authorization

Permissions in mobile apps can be divided into two types, install-time permissions and runtime permissions [24]. Here, the runtime permission is also called dangerous permission, which is related to users' privacy and can access users' private data, such as location information, contact information, etc. This information is considered sensitive and should acquire user authorization for access. When requesting the runtime permission, the system will display a prompt window, as shown in Figure 3.

According to Section II-A, the framework provides rich APIs to support the mini-programs to request a resource such as user profiles, location information, payment functions, etc.

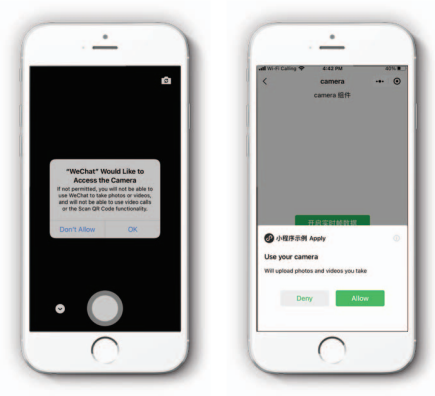

Fig. 3. Under the iOS system, the system permission prompt is displayed when the Alipay app requests *runtime permission* (left) and the pop-up window when a mini-program in Alipay app requests permission from the host app (right).

However, the user does not authorize the API directly. In mini-program development, the framework divided the dangerous APIs into multiple groups, which are named *scope* associated with different permissions. The users can select *scope* to authorize the permissions. After a *scope* is authorized by user, all of the APIs in this *scope* can use the data directly, i.e., they have the same permission.

D. Permission Control Abstract Model

We summarize an abstract model that is common to all mini-programs' permission control, depicted as follows.

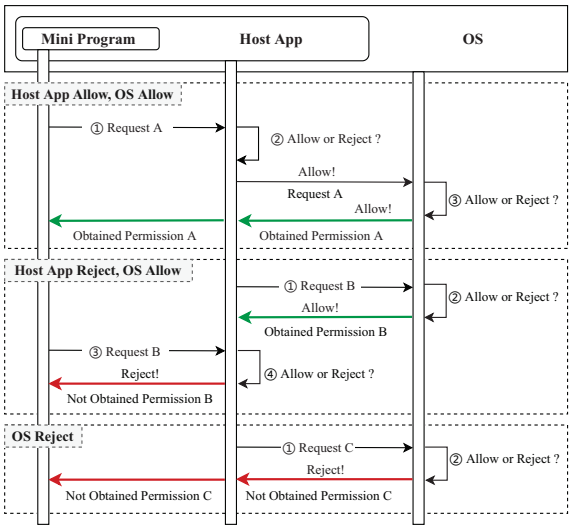

Fig. 4. The sensitive permission request process of mini-programs. Mini-programs run in specific mobile applications (host apps), while mobile applications run in the OS.

Mini-programs are "sub-applications" built on mature mobile applications, which are built on the OS. So, mini-programs need to pass two-layer authorization when requesting sensitive

permissions from users. When mini-programs need access to sensitive information, they first need to request the authorization of their host app, and then the host App needs to request the authorization of OS. As shown in Figure 4, according to whether the host app and OS permit or deny the application's permission request, we categorize the request process into the following three cases:

- **Both Host App and OS Allow** If a mini-program requests sensitive permission A, the host app will pop up a window to ask whether the user is willing to grant the permission. If the user grants the permission, the host app will continue to request sensitive permission A to the OS, and the OS will pop up a window. Once the user agrees, the mini-program will successfully obtain the requested sensitive permission. Later, when the users use the mini-program again, they can directly call the interface and obtain the data.
- **Host App Reject but OS Allow** In this case, the host app has obtained permission B given by the OS, but when the mini-program wants to request permission B, such permission request is refused. Hence, the mini-program should not be able to obtain permission B since the mini-program does not inherit the host app's permission. In our later analysis in Section III, most vulnerabilities we discovered belong to this category.
- **OS Reject** If the host app does not get the permission C given by the OS, neither the host app nor the mini-programs in it can get this permission. It is worth noting that when the program calls the API that requires permission, the application will still send a notification and ask for permission as normal. However, whether the user permits or denies it, the mini-programs will not have this permission.

III. Security Analysis

A. Security Principles

According to host-app vendors' technical documents, we define the security principles that the mini-program should be enforced are: *Any operations on the data related to a user's privacy need to require the user's authorization, whether the request is from the mini-program or host app.* The form of request permission can be a pop-up window or an interactive user operation. If the mini-program can obtain sensitive data when the user has no functions or rejects the requests, we recognize that a mini-program has gained unauthorized permission.

B. Access Control Framework

If the mini-program APIs access sensitive resources, a host app enforces the mini-program permission by calling the mini-program APIs. Although different host app vendors create different APIs, according to Section II-C, we know that all mini-programs use the traditional permission label, *scope*, an assignment model to manage the permission.

The host app designers believe that since all the APIs used by the mini-program come from the host, the mini-program is trustworthy. For interfaces involving user privacy, the user's authorization must be required. The vendors classify these interfaces into several authorization *scope* such as location, address, camera, etc. This permission scheme exists in all the host apps we analyzed. Specifically, WeChat defines a *scope* called "scope.camera". In contrast, Ali defines a *scope* called "my.scan".

C. Adversary Model

In this paper, we assume that the adversary aims to steal all kinds of users' privacy by exploiting the improper management of the mini-program's permission. The research makes the following assumptions. *(i)* We assume that the adversary is the mini-programs or the developers of the mini-programs. They want to steal the data without the user's awareness. *(ii)* The adversary can obtain all the official APIs and uses them effectively. *(iii)* The adversary does not need malicious apps installed on mobile phones and does not need privileges to execute malicious codes. *(iv)* The adversary does not need Android rooting or iOS jailbreaking. *(v)* The adversary is very familiar with the directory structure of mini-programs.

D. Potential Vulnerabilities

Followed by our security principles, we conduct a large-scale analysis of permission-related APIs from different host apps. A total of 2,580 APIs are detected through manual or semi-automatic [25] programming. Then we identify the following six categories of potential security vulnerabilities in mini-programs' sensitive permission request process.

1) Reuse Cache Files: When a user quits or deletes the mini-programs, the cache files in the corresponding path should also be deleted to avoid being reused. However, when we close a mini-program, either on iOS or Android, we find that some host apps do not completely empty the cache files. Although other mini-programs cannot access this cached file, the malicious mini-program can still reuse this cached file after another user logs in. This is not a high-level risk. However, since reading these cache files does not require privileges, the attackers can obtain the different users' private data that is stored in the cache file easily without the user's awareness. Hence, improper management of the cache files still brings privacy risks to the user.

2) Permission Encapsulation in Leaked APIs: Some APIs related to sensitive permissions do not encapsulate permissions well (we refer to these APIs as PEL-API). This means that any mini-program can directly call these APIs to obtain the corresponding permissions, ignoring the need to request permission within the host app. The security issue incurred during the sensitive permission request process is illustrated in Figure 6. When the host app obtains a dangerous permission B from the OS, since some APIs are not encapsulated well by host app vendors, in some cases, the mini-programs can obtain the permission without giving the user a notification. Also, in some cases, the mini-program can still get permission when the user rejects it. We divide these vulnerabilities into four

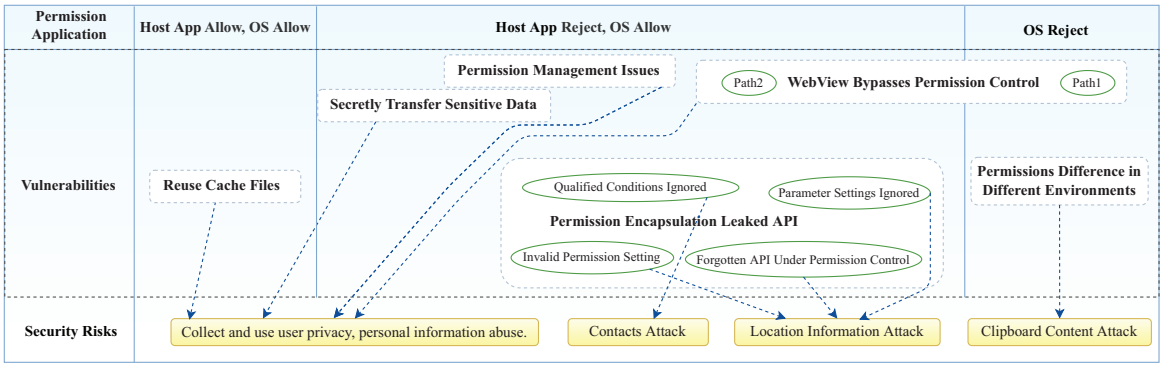

Fig. 5. The vulnerabilities and attacks in the mini-programs' permission granting process.

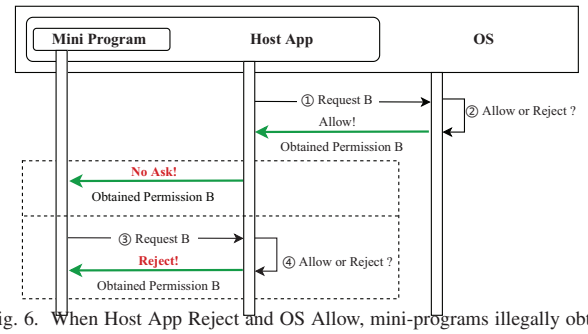

Fig. 6. When Host App Reject and OS Allow, mini-programs illegally obtain the process of sensitive permissions.

categories according to the reasons and methods of getting permission.

Qualified Conditions Ignored. According to the description in Section III-B, when the user must manually operate the sensitive permissions like view, select, or transmit private data, the mini-program will consider that the user has allowed the use of rights in default, so that there will be no pop-up window. In other words, these APIs do not belong to any *scope* (mentioned in Section II-C). And as long as the OS gives the relevant permissions to the host app, the mini-programs can call these APIs directly and get data at will. It is not a design fail since the user is aware of reading or processing private data. At this time, sensitive data can only be used when the user interacts with the mini-program, which means the user controls his/her sensitive information by him/her-self. Mini-programs cannot steal the user's sensitive information through this type of APIs.

However, we found that the host app designer may not put the sensitive APIs into the *scope*. As a result, the mini-program can obtain sensitive data directly without requesting permission. For example, the host app vendors continue to upgrade their software, and some new APIs will be added during these upgrades. These newly added APIs may be extensions of an old API or have similar functions. If the old API does not belong to any *scope* for the above reason, it is possible that these host app vendors do not put this new API into *scope*, too. However, by this new API, the mini-program may obtain sensitive data without the user's interactive operation, that is, the user's awareness. So this new API becomes a new privacy risk. We believe these APIs ignore the qualification of sensitive permission (whether private data will be stolen without the user's awareness).

Forgotten API Under Permission Control. The host apps handle whether the sensitive information can be passed to the mini-program. For example, almost all host apps consider obtaining "location information" as dangerous permission. When the user refuses the permission requests, the mini-program itself cannot locate the user's specific location, regardless of whether the host app has obtained the permission. However, the host app vendor neglected that some APIs should ask for authorization before sending the location information to the mini-programs. In other words, these APIs do not belong to any *scope* and can be called without users' authorization.

Parameter Settings Ignored. From the point of view of its functions, some APIs may not actively obtain the user's sensitive permissions. However, the host app vendor neglects the parameter settings in API, which will also steal users' private data. For example, *my.chooseCity* in the Alipay mini-programs is an API to open the city selection list. The parameter *showLocatedCity* indicates whether to display the currently located city. If it is set to true, the user's current city will be directly located regardless of whether the host program grants the location permission for the mini-programs. If the user does not do anything, we cannot see any content about the location information in the background. However, as long as the user selects the area located by the system, the host app will return the current city, latitude, and longitude to the mini-program in the background, even if the mini-program does not have the location permission at this time.

Invalid Permission Setting. Some APIs' permission settings are inconsistent with their official documents' descriptions. To some extent, it shows that the API related to sensitive permissions in mini-programs does not encapsulate permission well. For example, *wx.choosePoi* in mini-programs hosted by WeChat implements the function of opening the map and selecting the location. It is indicated in the document that the invocation of this API requires the authorization of *scope.userLocation*. However, in the actual test, we found

that the location can be selected without users' authorization; and such a case is inconsistent with the official document description. When the user chooses precise positioning, the host app will return the latitude and longitude data of the current user in the background.

3) Stealthily Transferring Sensitive Data: **Vertical**: Different mini-programs under the same host app transmit sensitive data. The request process for this vulnerability is shown in Figure 6. When the mini-programs' developer has some relationships with the host app vendor, like acquired by the host app vendor, the mini-program can obtain the users' data directly from the host apps without popping up permission request. Then, the method shown in Figure 6 can bypass user authorization for stealthily transmitting a user's sensitive data. **Horizontal**: Different mini-programs developed by the same company may share user information. All the user's sensitive permissions acquired by the mini-programs should be made visible to the user, and the user should have the right to disallow the mini-program to acquire the sensitive permissions. However, our empirical study found that some mini-programs obtain and leverage the user information in their associated mini-programs by default, including account information, shipping address, etc. This type of mini-programs skips the permission request step, and the permission settings pop-up is blank so that the user cannot revoke the relevant permissions.

4) Permission Management Issues: A mini-program may continue to use sensitive permissions to collect users' private information even if a user wants to revoke the permissions after using it. In particular, it can be divided into the following three situations.

Permission Setting Page Disappears. The permission setting page allows users to manage the permissions of mini-programs. However, some mini-programs may obtain permanent authorization after a one-time authorization due to the disabled or disappeared permission setting page. In this case, users cannot view what permissions they have granted to a mini-program, and they cannot cancel the previously authorized permission. As long as the host app is not uninstalled, the permission authorization will remain valid. Thus, a mini-program can use the previous authorization to continuously gather and use the user's personal information.

Permission Cannot be Revoked. Regarding the validity period of authorization of mini-programs, once a user explicitly agrees or rejects the authorization, such an authorization relationship will be recorded in the background until the user actively deletes the mini-programs. However, the permissions of some mini-programs may not be able to be revoked, and it can cause harmful consequences. These mini-programs will be able to use the previous authorization to continue collecting and using the user's personal information.

Unable to Completely Revoked Permissions. The permission settings of mini-programs should be revoked if the user actively removes the mini-programs. However, some host apps forget to clear these permission settings. So permission settings of some sensitive personal information (such as ID number) will be retained after the mini-program is re-installed.

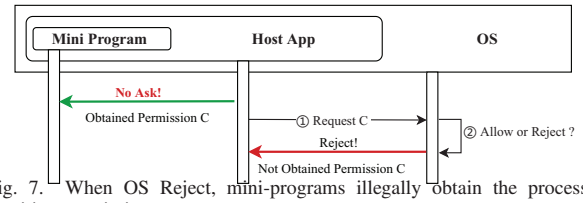

Fig. 7. When OS Reject, mini-programs illegally obtain the process of sensitive permissions.

However, when the mini-program is re-installed again, the setting options of related permissions will not be on the mini-programs' settings page, and the user cannot completely remove the permission.

5) WebView Bypasses Permission Control: The mini-programs can use the WebView component to carry H5 webpage. In this process, the loaded H5 page needs to manually import JS files (*i.e.* a web development toolkit based on the host apps for web developers) provided by each platform. In this way, developers can use the capabilities provided by the mobile phone, such as camera, Bluetooth, and GPS, to bring users a better experience. After the empirical study, as shown in Figure 7, we find that WebView component can bypass the specified API when the mini-programs or host apps do not give a pop-up for permission request or the user refuses permission request after the notification.

There are two scenarios to illegally obtain sensitive permissions for mini-programs after using the WebView component. **Scenario I**: the mini-programs may completely ignore the OS's permission control over host apps and host apps' permission control. They can directly access any sensitive permissions without notifying a user. Even when OS rejects sensitive permission requests from a host app, the mini-programs can still obtain such permissions through the WebView component. **Scenario II**: only the OS's permission control over host apps is considered, while the host apps' permission control over mini-programs is ignored. The specific process is shown in Figure 6. In this case, if the OS's permission control on host apps is disabled, the mini-programs will not obtain the corresponding sensitive permissions. If the OS's permission control on the host app is enabled, a user allows host apps to use certain sensitive permissions. Hence, any mini-program in host apps can obtain sensitive permissions. We would like to remark that both scenarios may result in the disclosure of user privacy.

6) Permission Issues in Different Environments: The mini-program framework uses the same running codes for iOS and Android. However, the processing details for some APIs are not the same between different OS and versions. Since the framework cannot handle these APIs differently, the same operations or program codes will have different results under different running environments. For example, apps can read the clipboard contents without the user's manually selecting "Paste" when the user copies something. This is by design. Nevertheless, if the user copies sensitive information and leaves it on the clipboard, all apps can capture it and send it to a remote server. Copying private data from a clipboard is risky. Different versions of OS have different feedback on this. The Android or old version of iOS will not inform the

TABLE I
THE LIST OF COLLECTED 9 HOST APPS.

Company	Host App	Monthly Active Users
Tencent	WeChat	1.31 billion [3]
	Tencent QQ	574 million [26]
Alibaba	Alipay	668 million [5]
ByteDance	TouTiao	731 million [27]
	TouTiao speed Edition	
	TikTok	1+ billion [4]
Baidu	Baidu	634 million [28]
Multi Vendors	QuickAPP	130 million* [29]
China UnionPay	UnionPay	10.6 million [30]

* Only contains Huawei's data.

user when an application reads the clipboard. Many host apps will also be silent when the mini-program reads the clipboard. If OS or host app does not consider the clipboard permission dangerous, mini-programs with access to the clipboard can steal the clipboard information of users in the background.

7) Clipboard Content Attack: Since the mini-program's framework does not restrict the apps from reading the clipboard, developers only need a few lines of code to see what users have just copied. If users copy an online banking password to paste and leave this private information on the clipboard, a malicious mini-program can read it in the background and see that data directly. The same goes for other sensitive data like names, addresses, credit card numbers, or even private photos. Mini-programs can capture everything from a user's clipboard and do whatever they want with it. The copied texts could be sent to a remote server without a user's awareness. More details can be found in Section V-C.

IV. EMPIRICAL STUDY

This section presents our empirical study for analyzing the current mini-programs permissions and examines the APIs for potential vulnerabilities according to Section III. Our goal is twofold. First, we collect the current popular mini-program platforms (Table I), and then expose potential vulnerabilities as outlined in Section III. We also exhibit the security issues exposed in the real world through detection. Second, we conduct case studies to show some real attacks based on improper permission management of mini-programs, to prove that the revealed vulnerabilities may cause severe consequences in real-world use.

A. Mini-programs Ecosystem

We select 9 popular host apps developed by 6 companies, which are listed in Table I. Each host app has its development tools. We use the respective development tools to test on different host apps. Through empirical study, we discuss their vulnerabilities and list them in Figure 8. The orange blocks indicate that the host app has corresponding vulnerabilities; green blocks indicate that the host app has fixed vulnerabilities; gray block indicates that the host app does not have such a vulnerability; and light yellow blocks indicate that it is uncertain if there exists such a vulnerability.

B. Vulnerability Analysis

1) Vulnerable Caching Mechanism: Our study discovered that in WeChat, when a user closes the used mini program but not being deleted from the *recent use*) list, and reopens the mini-program, the previously cached temporary files still exist. Because the local temporary file path can be obtained in the background, and reading this file does not require privileges, the temporary file can be reused without the user's awareness before being recycled. In QQ and ByteDance, as long as the user exits the mini programs or reopens the previously used mini-programs, the previous temporary files will be automatically deleted. Hence, they do not have this vulnerable caching concern. Meanwhile, we are not sure if this vulnerability exists in other host apps since we did not find the cache files of the mini-programs in the local folder or if the mini-program does not support personal testing. So this vulnerability has not been found temporarily in mini-programs of other host apps. We will keep an eye on this problem. (See Figure 8 "Cache-related issues".)

TABLE II
THE LIST OF COLLECTED PEL-APIS.

Vulnerabilities	Host App	API
Ignore	WeChat	*wx.searchContacts*
Forgotten	QQ	*MapContext.moveToLocation*
		MapContext.getCenterLocation
Parameter	Alipay	*my.chooseCity*
Invalidation	WeChat	*wx.choosePoi*

2) Vulnerable APIs: As discussed in Section III-D2, this kind of issue comes from those APIs that are related to sensitive permissions but cannot encapsulate permissions well. In order to find APIs with such vulnerabilities, according to the security principles mentioned in Section III-A, we set two criteria for analysis: 1) Whether the permission request needs to be granted by the user under the Android and iOS permission policy. 2) Whether there is human interaction in the process of private data acquisition. Suppose the user must manually operate to view, select or transmit private data. In that case, the mini-programs will consider that the user is aware of this process and allowed permission without notification. Otherwise, a permission popup should be displayed. Table II summarizes the APIs that we have found so far that are related to sensitive permissions but do not encapsulate permissions well. (See Figure 8 "Ignore, "Forgotten", "Parameter", "Invalidation".)

3) Vulnerable Permissions Transfer: **Vertical**: The mini-program "Amap" in the Alipay host app can be opened directly to precisely locate the user, ignoring the mini-program's request for the user's location permission. Although Alipay and AMap have reached an in-depth cooperative relationship (both belong to Alibaba Group) [31], this does not mean that their operations can bypass the user's willingness. No abnormality is found in the public test code of "official demo" provided by Alipay, but using this mini-program alone can directly locate users precisely. It shows that there may be an inconsistency between the source code of "demo" and the public test code provided. The attackers may use other ways to bypass the user authorization to transmit the user's location secretly.

Horizontal: Some companies may share user information among different mini-programs. For example, after logging in to the "Pinduoduo" mini-program in WeChat, the same

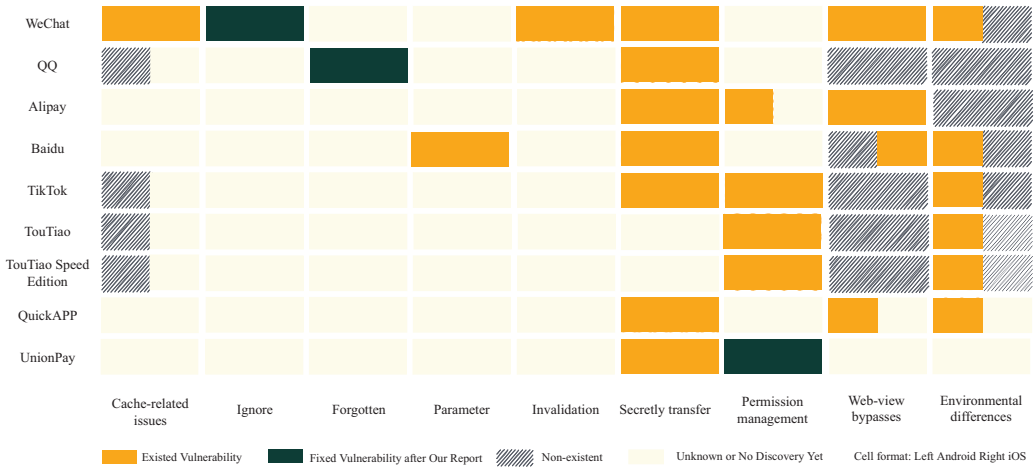

Fig. 8. The vulnerabilities distributions in the collected 9 host apps, where the vertical axis lists the names of host apps and the horizontal axis lists the vulnerabilities that we have discussed in Section 3. The left part of the cell is the analysis result on Android, and the right is iOS.

login information is displayed directly when the user first uses another mini-program named "Pinduoduo Coupon", which is from the same company. This type of mini-program skips the permission request, and there is no configurable option on the permission setting. So the user cannot revoke the permission to access these data. Thus, users cannot fully control the dissemination of their personal information. It will lead to the continuous accessibility of personal information and pose personal information at risk of being used in unknown circumstances. The issue can be called cross-mini-program authorization.

It's common to share the login credential between mobile apps like Single sign-on (SSO). However, we found that the information shared between mini-programs from the same developer/company includes not only login credentials but also user information, even data that requires access permissions, such as location. Although we did not discuss their isolation in this paper, the information sharing still needs to notify the user. (See Figure 8 "Secretly transfer".)

4) Insecure Permission Management: The permission management of UnionPay mini-programs is rather messy. Since the security of UnionPay is mainly related to user identity information and payment, it is not very strict to consider other sensitive permissions such as microphones, geographic locations, cameras, and photo albums. On the privacy settings page of the UnionPay APP, we can see the permission setting options for mini-programs. However, only the authorization of the phone number and identity information (name, ID number) is displayed. In contrast, permissions on other sensitive information such as location, access to mobile phone albums, and the camera are ignored, and there is no permission setting page inside mini-programs. When a user grants mini-programs hosted by UnionPay sensitive permission, such as a microphone, geographical location, camera, and photo album, as long as the UnionPay app is not uninstalled, the other mini-programs hosted by UnionPay can always access these permissions.

We found that, in ByteDance, the permission of mini-programs cannot be removed with the deletion of the mini-programs. In ByteDance's official document, there is no clear explanation of the validity period of the authorization and whether the authorization will remain valid after deleting mini-programs. Hence, mini-programs hosted by ByteDance may use previous authorizations to continuously access and collect user personal information.

Although the authorized sensitive permission of Alipay mini-programs can be deleted after a user actively deletes the mini-program, the sensitive personal information involved before (such as ID number) will be retained by certain mini-programs. We have canceled the authorization and removed the mini-program, but the relevant information can still be accessed when entering the mini-programs again after a certain period. At this time, the permission setting of the mini-program becomes empty, and the user cannot set the permission on the setting page. This situation indicates that after the user revokes the permission, the related mini-programs do not update the permission information in time and may continue to collect and use the user's sensitive information. (See Figure 8 "Perimission management".)

5) Insecure WebView Component: Some mini-programs can carry webpages through the WebView component. The communication between the webpage and mini-programs is implemented by the interface provided by the webpage development kit, which is based on host apps provided by each platform.

In WeChat, for Android, the user's stored data can be obtained using the WebView components provided by the host app. The mini-programs can completely bypass the normal permission control under Scenario I described in Section III-D5. Different from that, for iOS, the mini-programs can obtain the user's album permission under Scenario I and can access camera data under Scenario II. In Alipay, no

Fig. 9. Examples of getting clipboard contents in Android. Only ② and ③ will pop up a window to the user when the mini-programs get the contents of the user's clipboard.

matter for Android or iOS, the mini-programs can bypass the standard permission control under Scenario II, to obtain the data such as the user's camera, photo album, location, etc. When users reject these sensitive permissions, the data can still be obtained by using the WebView component. In Baidu, the mini-programs can obtain the camera permission to take pictures for iOS under Scenario II. And the mini-programs of QuickApp in Huawei can obtain the users' stored data (pictures, audio, videos, documents, etc.) under Scenario II. The malicious mini-programs in QuickApp will get users' precise locations without their awareness. (See Figure 8 "Webview bypass".)

6) Vulnerable Clipboard Mechanism: After the iOS 14 upgrade, a pop-up will inform the user when an app reads the clipboard's contents. Hence, there will be a notification when the mini-programs read the clipboard, no matter which host apps they run inside.

Android is more complex since there are multiple versions. For example, Xiaomi MIUI is a third-party mobile phone OS, which is deeply optimized, customized, and developed by Xiaomi based on Android. One version of it, Xiaomi MIUI12, divides the access permission of the clipboard into finer granularity, and users can monitor each request's reading and writing behavior. However, in other Android phones (such as Huawei, Vivo, etc.), users cannot involve the clipboard's permission control because they will not get a notification. Under these OS versions, we test the clipboard permission management on different host apps in the Android environment, and check their corresponding prompts, with results shown in Figure 9. To exhibit that the mini-programs have obtained the contents of a user's clipboard, we display a modal dialog box. We found that 6 out of the 8 host apps (cases 1, 4, 5, 6 in Figure 9) did not give any prompt to the user when obtaining the clipboard information, which may be vulnerable to the clipboard content theft described in Section V-C. Cases 2 and 3 in Figure 9 show a design example that the mini-programs will prompt when obtaining a user's clipboard contents. However, for Android and iOS, case 2 only reminds the user that the mini-programs have obtained the clipboard's contents through a pop-up, but the user cannot block the accessing to the clipboard. In case 3, obtaining the clipboard's content is set as the user's permission to proceed. The user must select "Allow" before the mini-program can obtain the corresponding information. We consider these designs with the best security usability practice.

It should be noted that, although it is set as the user's permission to operate in case 3, there is still an issue with Alipay. In version 10.2.26.8000, Alipay will pop up a window to ask the user "Request to obtain the contents of your clipboard". However, the clipboard contents have been pasted on the page before clicking "Reject" or "Allow" (this is equivalent to the mini-program still being able to obtain the clipboard information after the user refuses the permission authorization). While in version 10.2.23.7100, users cannot get any content after clicking "Reject". (See Figure 8 "Environmental differences".)

V. Case Studies

In this section, we present our case study of some representative APIs and privacy issues in mini-programs.

A. Stealing Location Information

In the QQ mini-program, the index.js file uses *qq.createMapContext* to create a *MapContext* object. After the user use *MapContext.moveToLocation* to move the map center to the current location, the malicious mini-program can use *MapContext.getCenterLocation* to get the latitude and longitude of the current map center. In the whole process, the user's geographic location, latitude, and longitude can be accurately obtained in the background without the user's authorization. When using Tencent's location service [32], the attacker can obtain the user's precise location. We confirmed that the QQ mini-programs could obtain the center point of the current location without the user's authorization, regardless of the Android or iOS system, and successfully transfer the data, specific location, and other information to the mini-program. Once the mini-program associates the location information with the account information, the user's personal information will be completely exposed. The malicious mini-program can further obtain the user's location continuously and then calculate the user's physical traces easily.

Hence, the QQ example is not a missing authorization check. The main reason is the vendor of QQ, Tencent, did not put some location-related APIs in the *scope*. If the mini-program uses these APIs, without permission management since it is not in *scope*, it can obtain the local directly without the user's authorization.

B. Stealing Contacts

The API provided by the WeChat mini-programs, *wx.searchContacts*, is to find the contacts and match a similar mobile phone number. This API does not specify the number of calls within a period. When the *phoneNumber* parameter (i.e., the number to be searched) is written in a loop, most information in the user contacts can be obtained by traversing in sequence. Specifically, if a mini-program sets a button in *index.wxml* to bind an event and writes the API *wx.searchContacts* into this event in the *index.js* file. When a user clicks this button, the mini-program will get the data about the user's contacts in the background. If the host app (i.e., WeChat) has already obtained the contacts' permission, the mini-programs can obtain partial contact information without the user's awareness. After testing, we have confirmed that WeChat mini-programs can use this method to obtain the users' contacts and successfully transmit the obtained data to the mini-programs' background process, whether in Android or iOS, without the user's authorization and the user's awareness. Attackers can take advantage of this vulnerability and bind *wx.searchContacts* to an inductive button to entice users to click. Sensitive data in the user's address book may be read and uploaded in the background, resulting in information leakage.

C. Stealing Clipboard Information

Take the WeChat mini-programs as an example. During the test, we write *wx.getClipboardData* in the onload event in the JS code so that the clipboard content can be easily obtained without the user's awareness. Even if it is not written in the onload event, binding this API to the button control event can also trigger it. In the real world, mini-programs can write some inductive slogans on the button to induce the user to click and then obtain the user's clipboard contents. After a large number of tests, we have confirmed that many host apps (e.g., WeChat, ByteDance, Baidu, QuickAPP) can obtain user clipboard information and successfully transfer the obtained data into the mini-programs in the background without the user's authorization and awareness. Suppose the copied content is not destroyed after the user pastes it into an application. In that case, the content can still be obtained when the user opens a mini-program, thus causing the leakage of the user's sensitive information. For example, when a user copies the name of a certain product, after opening a shopping mini-program, it can read the user's clipboard content and upload the private information in the background. The developers of this mini-program will know the user may want to buy this product and push similar commodities or analyze the user's behavior to push advertisements precisely.

D. Responsible Disclosure

To ensure that the mini-programs' mechanism of different host apps had sufficient time to fix the vulnerabilities, we contacted them individually about the vulnerabilities several months before submitting this manuscript. This allowed several different host apps to finish patching the reported vulnerabilities. We reported the results of our investigation to the Tencent Security Response Center as well as to CVE (CVE-2021-33057, CVE-2021-40180).

VI. Discussion

A. Seriousness of the Vulnerabilities

Some vulnerabilities look like bugs and are easy to be fixed by the vendor. Take the disappearance of the setting page (IV-B4) as an example. It is a bug; meanwhile, we also think it is a serious vulnerability because it is a design flaw in the mini-program's framework, and all the mini-programs operated by this host app will be affected. The mini-program users can do nothing but watch the malicious apps steal the data.

B. Limitations and Future Works

We conducted a series of mini-programs' permission tests based on personal accounts. In other words, the tested APIs are all for individuals. According to our statistics, the number of APIs open to non-individual developers is 4.6% of the number of the total APIs. This type of interface usually includes obtaining the user's mobile phone number, motion data, etc. Different host apps have different attitudes towards such interfaces. For example, the WeChat mini-programs allow individual developers to obtain users' motion data, while in the Alipay mini-programs, this interface is only open to corporate users. Generally, those interfaces that are only open to enterprise users will be more strictly managed by host apps since these can obtain more data. Therefore, it is very difficult to examine the APIs used by enterprise accounts. In the future, we will cooperate with enterprises to test and research these APIs.

C. Mitigation Measures

Our empirical analysis aimed to draw people's attention to the neglected security issue of the improper use of sensitive permissions in mini-programs. First, the lack of isolation between mini-programs for some specific permissions like reading/writing clipboard is one of the design flaws. Besides, due to inheritance, the permissions of mini-programs are the subset of the host App's permissions. When the host App is not able to manage the permission properly, the mini-program may obtain more permissions than it needs. This violates the principle of ***Least Privilege*** in the security area. We call on the host app's vendors to consider privacy protection and follow the security principles when designing the mini-programs' APIs and access control framework. Users should also increase their security awareness to protect their personal information when using mini-programs, and should be vigilant against mini-programs of unknown origin. Everyone should not quickly authorize their private information to the mini-programs to prevent it from being illegally collected and leaked.

VII. Related Work

There has not been much research relevant to mini-programs in the current academic circles regarding the privacy issues related to permissions. Lu *et al.* [33] propose their study on the resource management in mini-program and reveal some high-impact security flaws. Zhang *et al.* [25] research the identity confusion problem in the mini-program ecosystems.

Since mini-programs are built and run on mature mobile applications (such as WeChat, Alipay, etc.), there are a lot of studies on security related to mobile application permissions for us to learn from [34]–[37]. In this section, we introduce them from the following two aspects.

A. Permission Management Concerns in Mobile Apps

Almomani *et al.* [38] demonstrated, discussed, and compared the latest technologies in the field of Android permissions, and conducted the latest research on Android permissions, revealing that Android permissions face various security issues. Fang *et al.* [39] investigate the arising issues in Android security, including coarse granularity of permissions, incompetent permission administration, insufficient permission documentation, over-claim of permissions, permission escalation attack, and TOCTOU (Time of Check to Time of Use) attack and put forward several methods to further reduce Android security risks. Reardon *et al.* [13] searched for sensitive data being sent over the network for which the sending app did not have permissions to access it by mechanisms to monitor the application's runtime behavior and network traffic. They found that apps can circumvent the permission model and gain access to protected data without user consent by using both covert and side channels and determined how this unauthorized access occurs. Mujahid *et al.* [40] implements a technique in a tool called PERMLYZER, which automatically detects permission issues from apps APK.

B. Inappropriate Permission Detection in Mobile Apps

In Android, the most common is the permission management mechanism of Android [34], [35], [41], [42]. DroidNet [12] is an Android permission control and recommendation system, which is an Android permission control framework based on crowdsourcing. It provides recommendations on whether to accept or reject the permission requests based on decisions from peer expert users, which can help users implement low-risk resource access control for untrusted applications and protect users' privacy. HybridGuard [43], a framework based on the subject authority and fine-grained policy execution for web mobile applications, can accurately monitor all web codes to ensure the security of mobile applications, in which an interception and policy code is implemented in a single JavaScript file, and whether to intercept them is determined by wrapping API about device resource access and DOM operation and checking the policy. M-Perm [11] is a detection tool that combines string analysis and static analysis to identify normal, dangerous, and third-party permission requests in applications to detect permission abuse. Cusper [44] is a new modular design in the Android permission model, which separates the management of system permissions from custom permissions declared by untrusted third-party applications. It introduces backward compatible naming conventions for custom permissions to systematically eliminate and prevent the loopholes of custom permissions.

The mainstream approach for enhancing the Android permission mechanism is to identify over-declared permissions requested by an app [45]–[48], and recommend appropriate permissions for an app [49], [50]. TERMINATOR [16] provides a safe, reliable, yet non-disruptive approach to protect mobile users against permission misuses. Liu *et al.* proposed a Personalized Privacy Assistant (PPA) for mobile applications, which can manage mobile permissions of mobile applications and predict the privacy settings that users want by asking some questions, and proposed a method to learn the privacy profile of permission settings [14]. Bao *et al.* also proposes two novel approaches to realize permission recommendations [51] .

VIII. Conclusion

The mini-program is a new mobile application format that runs inside a mobile app. Although these mini-programs are taking over the traditional mobile OS and have become the way to do almost everything in China, there is little research on these mini-programs, especially regarding their potential security and privacy issues. In this paper, we conducted a large-scale analysis of mini-programs in different host apps for the first time. We have conducted empirical research on 9 currently popular host apps, revealing the security issues corresponding to the 6 types of potential security vulnerabilities we have discovered in the real world. We propose corresponding attack methods to analyze these potential weaknesses to exploit the discovered vulnerabilities. In addition, we also showed three real attacks on the mini-program's permissions to prove that the revealed vulnerabilities may cause severe consequences in real-world use. Following the practice of responsible disclosure, we have also reported newly discovered vulnerabilities to relevant security platforms, among which the more severe vulnerabilities obtained CVE numbers. Lastly, we put forward a series of suggestions for the future deployment of mini-programs to protect users' privacy.

References

[1] "Mini program platforms 2021: Wechat vs. alibaba vs. baidu," https://www.chinainternetwatch.com/30749/mini-program-platforms/.

[2] "Miniapp standardization white paper," https://www.w3.org/TR/mini-app-white-paper/.

[3] "Number of monthly active wechat users from 2nd quarter 2011 to 3rd quarter 2022," https://www.statista.com/statistics/255778/number-of-active-wechat-messenger-accounts/#statisticContainer.

[4] "Tiktok statistics – updated jan 2023," https://wallaroomedia.com/blog/social-media/tiktok-statistics/.

[5] "Number of monthly users of alipay mini programs in china from september 2020 to september 2022," https://www.statista.com/statistics/1359311/china-number-of-alibaba-alipay-mini-program-monthly-users/.

[6] "Number of monthly active facebook users worldwide as of 3rd quarter 2022," https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/.

[7] D. Barrera, H. G. Kayacik, P. C. Van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 73–84.

[8] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *IEEE security & privacy*, vol. 7, no. 1, pp. 50–57, 2009.

[9] X. Ma, "App store killer? the storm of wechat mini programs swept over the mobile app ecosystem," *Retrieved November*, vol. 15, p. 2019, 2019.

[10] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 217–228.

[11] P. Chester, C. Jones, M. W. Mkaouer, and D. E. Krutz, "M-perm: A lightweight detector for android permission gaps," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 217–218.

[12] B. Rashidi, C. Fung, A. Nguyen, T. Vu, and E. Bertino, "Android user privacy preserving through crowdsourcing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 773–787, 2017.

[13] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 603–620.

[14] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. A. Zhang, N. Sadeh, Y. Agarwal, and A. Acquisti, "Follow my recommendations: A personalized privacy assistant for mobile app permissions," in *Twelfth Symposium on Usable Privacy and Security*, 2016, pp. 27–41.

[15] I. Mohamed and D. Patel, "Android vs ios security: A comparative study," in *2015 12th International Conference on Information Technology-New Generations*. IEEE, 2015, pp. 725–730.

[16] A. Sadeghi, R. Jabbarvand, N. Ghorbani, H. Bagheri, and S. Malek, "A temporal permission analysis and enforcement framework for android," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 846–857.

[17] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau, and S. Weisgerber, "On demystifying the android application framework: Re-visiting android permission specification analysis," in *25th {USENIX} security symposium ({USENIX} security 16)*, 2016, pp. 1101–1118.

[18] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android permissions remystified: A field study on contextual integrity," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 499–514.

[19] L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang, "Upgrading your android, elevating my malware: Privilege escalation through mobile os updating," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 393–408.

[20] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 224–238.

[21] G. Petracca, Y. Sun, A. Atamli-Reineh, P. D. McDaniel, J. Grossklags, and T. Jaeger, "Entrust: Regulating sensor access by cooperating programs via delegation graphs." in *USENIX Security Symposium*, 2019, pp. 567–584.

[22] L. Hao, F. Wan, N. Ma, and Y. Wang, "Analysis of the development of wechat mini program," in *Journal of Physics: Conference Series*, vol. 1087, no. 6. IOP Publishing, 2018, p. 062040.

[23] "What is instant app (google android instant app)?" https://searchmobilecomputing.techtarget.com/definition/instant-app.

[24] "Permissions on android," https://developer.android.google.cn/guide/topics/permissions/overview.

[25] L. Zhang, Z. Zhang, A. Liu, Y. Cao, X. Zhang, Y. Chen, Y. Zhang, G. Yang, and M. Yang., "Identity confusion in webview-based mobile app-in-app ecosystems," in *In 31st USENIX Security Symposium*, 2022, pp. 1597–1613.

[26] "Number of monthly active qq users from 3rd quarter 2019 to 3rd quarter 2022," https://www.statista.com/statistics/227352/number-of-active-tencent-im-user-accounts-in-china/.

[27] "Number of monthly active users of popular short video apps in china in november 2022," https://www.statista.com/statistics/910633/china-monthly-active-users-across-leading-short-video-apps/.

[28] "Baidu q3 2022 on ai, autonomous driving; baidu app mau up 5%," https://www.chinainternetwatch.com/31413/baidu-quarterly/.

[29] "The monthly active users of quick app," https://twitter.com/Huawei_devs/status/1398270846481965063.

[30] "Top payment apps in china in 2021," https://ecommercedb.com/news/top-payment-apps-in-china-in-2021/3238.

[31] "Introduce of autonavi software," https://en.wikipedia.org/wiki/AutoNavi.

[32] "Tencent location service," https://lbs.qq.com/service/webService/webServiceGuide/webServiceGcoder.

[33] H. Lu, L. Xing, Y. Xiao, Y. Zhang, X. Liao, X. Wang, and X. Wang, "Demystifying resource management risks in emerging mobile app-in-app ecosystems," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security*, 2020, pp. 569–585.

[34] A. Sadeghi, H. Bagheri, J. Garcia, and S. Malek, "A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software," *IEEE Transactions on Software Engineering*, vol. 43, no. 6, pp. 492–530, 2016.

[35] D. J. Tan, T.-W. Chua, and V. L. Thing, "Securing android: a survey, taxonomy, and challenges," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–45, 2015.

[36] M. Diamantaris, E. P. Papadopoulos, E. P. Markatos, S. Ioannidis, and J. Polakis, "Reaper: real-time app analysis for augmenting the android permission system," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, 2019, pp. 37–48.

[37] J. Xiao, S. Chen, Q. He, Z. Feng, and X. Xue, "An android application risk evaluation framework based on minimum permission set identification," *Journal of Systems and Software*, vol. 163, p. 110533, 2020.

[38] I. M. Almomani and A. Al Khayer, "A comprehensive analysis of the android permissions system," *IEEE Access*, vol. 8, pp. 216 671–216 688, 2020.

[39] Z. Fang, W. Han, and Y. Li, "Permission based android security: Issues and countermeasures," *computers & security*, vol. 43, pp. 205–218, 2014.

[40] S. Mujahid, R. Abdalkareem, and E. Shihab, "Studying permission related issues in android wearable apps," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 345–356.

[41] E. Alepis and C. Patsakis, "Unravelling security issues of runtime permissions in android," *Journal of Hardware and Systems Security*, vol. 3, no. 1, pp. 45–63, 2019.

[42] Y. Zhang, M. Yang, G. Gu, and H. Chen, "Rethinking permission enforcement mechanism on mobile systems," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 10, pp. 2227–2240, 2016.

[43] P. H. Phung, A. Mohanty, R. Rachapalli, and M. Sridhar, "Hybridguard: A principal-based permission and fine-grained policy enforcement framework for web-based mobile applications," in *2017 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2017, pp. 147–156.

[44] G. S. Tuncay, S. Demetriou, K. Ganju, and C. Gunter, "Resolving the predicament of android custom permissions," 2018.

[45] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 1025–1035.

[46] J. Wang and Q. Chen, "Aspg: Generating android semantic permissions," in *2014 IEEE 17th International Conference on Computational Science and Engineering*. IEEE, 2014, pp. 591–598.

[47] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1354–1365.

[48] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "{WHYPER}: Towards automating risk assessment of mobile applications," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 527–542.

[49] K. Huang, J. Han, S. Chen, and Z. Feng, "A skewness-based framework for mobile app permission recommendation and risk evaluation," in *International Conference on Service-Oriented Computing*. Springer, 2016, pp. 252–266.

[50] Z. Liu, X. Xia, D. Lo, and J. Grundy, "Automatic, highly accurate app permission recommendation," *Automated Software Engineering*, vol. 26, no. 2, pp. 241–274, 2019.

[51] L. Bao, D. Lo, X. Xia, and S. Li, "Automated android application permission recommendation," *Science China Information Sciences*, vol. 60, no. 9, pp. 1–17, 2017.