Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs

Yue Zhang*
The Ohio State University

Yuqing Yang
The Ohio State University

Zhiqiang Lin
The Ohio State University

ABSTRACT

Mobile mini-programs in WeChat have gained significant popularity since their debut in 2017, reaching a scale similar to that of Android apps in the Play Store. Like Google, Tencent, the provider of WeChat, offers APIs to support the development of mini-programs and also maintains a mini-program market within the WeChat app. However, mini-program APIs often manage sensitive user data within the social network platform, both on the WeChat client app and in the cloud. As a result, cryptographic protocols have been implemented to secure data access. In this paper, we demonstrate that WeChat should have required the use of the "appsecret" master key, which is used to authenticate a mini-program, to be used only in the mini-program back-end. If this key is leaked in the front-end of the mini-programs, it can lead to catastrophic attacks on both mini-program developers and users. Using a mini-program crawler and a master key leakage inspector, we measured 3,450,586 crawled mini-programs and found that 40,880 of them had leaked their master keys, allowing attackers to carry out various attacks such as account hijacking, promotion abuse, and service theft. Similar issues were confirmed through testing and measuring of Baidu mini-programs too. We have reported these vulnerabilities and the list of vulnerable mini-programs to Tencent and Baidu, which awarded us with bug bounties, and also Tencent recently released a new API to defend against these attacks based on our findings.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; **Mobile and wireless security**.

KEYWORDS

Mobile Security; Mobile Super Apps; Miniprogram Security; Credentials Leakage

ACM Reference Format:

Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* 15 pages. https://doi.org/10.1145/3576915.3616591

*This author is now with Drexel University.

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3616591

1 INTRODUCTION

A mini-program, which is a small program restricted in size, is gaining popularity on social network platforms such as WeChat, which is the third most popular messenger app with 1.2 billion monthly active users [1]. These mini-programs have greatly expanded the capabilities of super-apps, such as WeChat, which has almost become an "operating system" offering more than 900 APIs [6] for executing more than 4 million mini-programs [5]. These mini-programs cater to various daily needs of users, including online shopping in virtual stores and scan-to-buy in physical stores. Among the popular mini-programs running on the WeChat platform, `PinDuoDuo`, a group-buying app, is such an example.

While many social network platforms, like Facebook [14], have provided APIs for third-party developers to access and use the social network information collected, super apps such as WeChat have taken it further and made it even more open with their mini-program paradigm. Mini-programs can access more generalized user-specific data directly in the platform, maintained by both the local host-app and remote cloud, through uniformed APIs. For instance, by invoking `getPhoneNumber`, a WeChat mini-program can retrieve a user's telephone number. A telephone number in mobile super apps is a crucial identifier used to bind the mini-program account to a specific user, and the super apps even allow mini-programs to authenticate their users solely through the phone number without any passwords. Furthermore, in addition to local resources and user-specific data, WeChat provides online paid services like Optical Character Recognition (OCR), which converts images of text to electronic text, to enrich the functionalities of the mini-programs without adding heavy barriers.

Super apps have implemented access control mechanisms and cryptography protocols to secure services access and data transmission to protect privacy-sensitive data and paid services from potential data leaks and abuse by mini-programs. For instance, when accessing sensitive user records stored in WeChat servers, WeChat encrypts the data with a user-specific encryption key (`EK` or `session key` in Tencent's terminology), which the mini-program back-end requests with the mini-program's master key (`MK` or `AppSecret`), the mini-program ID, and the user's login token (`LT`). The mini-program back-end then decrypts the data on the back-end using the retrieved `EK`. Similarly, when accessing paid services provided by WeChat, the mini-program back-end requests an access token with `MK`, mini-program ID, and uses the obtained access token to invoke the services. However, not all developers

understand the implications of the untrustworthiness of the mini-program front-end, and some mistakenly distribute MKs to the front-end of the mini-program, as confirmed with our manual analysis of several well-known mini-programs that store their MKs in the mini-program code, allowing an attacker to easily obtain their MKs.

This study systematically examines MK leaks in WeChat to understand their prevalence, root causes, and consequences. Our findings can be applied to other super apps, given the similar mechanisms they share. In particular, we show that MK leaks can result in severe consequences, including user account hijacking, promotion abuse, and service theft. For example, in WeChat, a common practice to index users is through their phone numbers. With the leaked MK of a vulnerable mini-program, an attacker can query the EK from the WeChat servers to decrypt any encrypted data delivered from the servers. This enables access to all data maintained by the victim's account, such as the user's citizen ID and shipping address. Additionally, the attacker can cause financial losses to the victim, such as gaining cashback and coupons from vendors by manipulating sensitive data, such as group chat information. Finally, with the obtained MK, the attacker can consume paid services purchased by the victim for free.

To gauge the severity and prevalence of the MK leak among mini-programs, we have conducted a measurement study in this paper. Specifically, by scanning a dataset of 3,450,586 mini-programs, we discover 40,880 mini-programs that had leaked their MKs. Our evaluation also reveals that vulnerable mini-programs are not limited to unknown developers, but include popular mini-programs from high-profile vendors such as *Nestle*, *HP*, and *Tencent*. We have reported the MK leakage vulnerabilities and the list of vulnerable mini-programs to *Tencent* and been awarded with bug bounties. Currently, *Tencent* is actively working with developers to fix this vulnerability, and some mini-programs have already removed MKs from their code. Furthermore, *Tencent* recently released a new API to defend against attacks based on our findings [7].

Contributions. We make the following contributions:

- **Systematic Understanding (§3).** We are the first to systematically examine the sensitive resource access protocols of mini-programs, resulting in the discovery of MK leakage vulnerabilities across multiple platforms such as WeChat.
- **Empirical Measurement (§4).** We develop a measurement tool and evaluate it with a large set of 3,450,586 mini-programs. The result shows that currently around 40,880 of mini-programs that contain MK leakage vulnerability.
- **Practical Attacks (§5).** We demonstrate two types of novel attacks with leaked MKs: attacks against sensitive resources and attacks against cloud services. We show that these attacks can have devastating impacts in the super app ecosystem, such as hijacking user's account, manipulating user's sensitive data, or consuming cloud services for free.
- **Countermeasures (§6).** In addition to the responsible disclosure of this vulnerability as well as the list of 40,880 mini-programs that leaked MKs, we also shed light on the possible mitigations and preventions, particularly on how super app vendors could have fixed this issue.

Name	API	Encrypted?	Price
Sensitive Data			
Phone number	`getPhoneNumber()`	✓	-
User Info	`wx.getUserProfile()`		
└Gender	├`info.gender`	✓	-
└Nick name	├`info.nickName`	✓	-
└Avatar	└`info.avatarURL`	✓	-
Shared Info	`wx.getShareInfo()`	✓	-
Promoting Messages	`wx.authPrivateMessage()`	✓	-
Group Chat Info	`wx.getGroupEnterInfo()`	✓	-
WeRundata	`wx.getWeRunData()`	✓	-
Cloud Services			
AI services			
OCR Services			
├Bank account	`openapi.ocr.bankCard`	✗	~$ 1,000
├Business License	`openapi.ocr.businessLicense`	✗	~$ 1,000
├Drive License	`openapi.ocr.driveLicense`	✗	~$ 1,000
├National ID	`openapi.ocr.idCard`	✗	~$ 1,000
├Regular Text	`openapi.ocr.plainText`	✗	~$ 1,000
└License plate	`openapi.ocr.vehicleLicense`	✗	~$ 1,000
AI Chat Bot	`openapi.ans_node_name`	✗	0
AI Products Classification	`goodclass2`	✗	0
Translation	`multilingualMT`	✗	0
Jokebot	`jokebot`	✗	0
Products Info Extraction	`goodinfo`	✗	0
Security Services			
├Black Market Report	`weixinSecintelligenceresp`	✗	0
├Fraud Detection	`weOpensecRiskservice`	✗	~$ 5,000
└User Risks Detection	`weOpenSecuseracctRiskLevel`	✗	~$ 5,000
Map Services			
├Poi Search	`poisearch`	✗	~$ 260
├Address Resolution	`geoc`	✗	~$ 260
├Coords Conversion	`coordTrans`	✗	~$ 260
└Poi Suggestion	`poiSuggestion`	✗	~$ 260

Table 1: Summary of sensitive resources managed by WeChat. The price is for 1 million's invocations of the services [32].

2 BACKGROUND

2.1 Sensitive Resource Access by Mini-programs

There are two types of resource that can only be accessed by mini-programs in WeChat: (1) sensitive data and (2) cloud services:

- **Sensitive Data Access**. Sensitive data refers to data generated by end-users and collected by mini-programs. For example, by examining the APIs provided by WeChat and their official documentation [6], we have identified multiple privacy-sensitive data, as shown in Table 1 (note that other super-apps such as Baidu have similar APIs that can be invoked by their mini-programs to access sensitive data). This data includes user information associated with a particular user, such as their nickname and phone number, which is kept on WeChat servers. When accessed by third-party mini-programs through APIs, WeChat will first encrypt the data and allow them to be decrypted only with a decryption key that is fetched from the MK on the mini-program's back-end.
- **Cloud Service Access**. Cloud services are the services provided by WeChat to mini-program developers for freeing them from re-implementing complex functionalities such as Optical Character Recognition (OCR) services. Some services are not provided for free (e.g., fraud detection services

	Mini-programs	Mobile apps	Web apps
Platforms store users' data on cloud for API use?	✓	✗	✗
API for social network resources?	✓	✗	✗
Nature support of encryption based access protocol?	✓	✗	✗
Nature support of token based access protocol?	✓	✗	✗
Nature support of key management protocol?	✓	✗	✗
Apps need to be vetted?	✓	✓	✗

Table 2: Comparison of Mini-Programs, Mobile Apps, and Web Apps.

provided by WeChat cost around $5,000 per million invocations). Having investigated the services provided by WeChat on the online developer dashboard, we find that those services can be grouped into 3 categories as shown in Table 1: (1) AI services (e.g., for AI chat bots); (2) Security services, which detects potential user risks as well as fraud risks; and (3) Map services (e.g., for searching and resolving locations). When they are accessed by 3rd-party mini-programs through APIs, WeChat requires a correct access token obtained from MK at the mini-program's back-ends.

2.2 Comparison of Mini-Programs, Mobile Apps, and Web Apps

As shown in **??**, WeChat's mini-programs function like an "operating system" but differ from traditional mobile OS and web browsers. Unlike mobile apps, mini-programs don't need installation. Super apps store user data in the cloud, letting mini-programs access it via APIs, while mobile OS and browsers use locally stored data and third-party services like Amazon's cloud. Mini-programs have built-in encryption, token-based access, and key management protocols, whereas other apps can add these independently. Both mini-programs and mobile apps undergo platform vetting, but web apps don't, as anyone can create web pages.

3 UNDERSTANDING THE ATTACK SURFACE

3.1 Parties Involved

Sensitive data access is complicated in WeChat mini-programs, since multiple parties including WeChat client, WeChat server, mini-program front-end, and mini-program back-end all need to be involved, as shown in Figure 1.

- WeChat **Client (WC)** provides a JavaScript runtime environment that enables developers to write and run their mini-programs using JavaScript. Moreover, it also offers a range of APIs that support a wide range of needs in mini-program development, including UI rendering, resource access, and network accessing.
- WeChat **Server (WS)** is crucial in the WeChat mini-program ecosystem. They vet submitted mini-programs (similarly to how Google Play vets Android apps), and securely store and deliver sensitive resources (e.g., services and data) to trusted mini-programs upon request.
- **Mini-program's Front-end (MF)** handles UI rendering, user requests processing, and communicates with the mini-program back-end to execute specific business logic like online shopping and ride-hailing. Unfortunately, MF is developed using JavaScript, making it susceptible to reverse engineering.
- **Mini-Program's Back-end (MB)** is used to store user specific data related to the mini-program. These MBs are the backbones of mini-programs that handle both user data generated from front-end, and user sensitive resources delivered by WeChat severs.

Definition	Length (bits)	Generated by	Involved Parties	Validation Period
Master Key (MK)	256	WS	MB, WS	∞
Encryption Key (EK)	128	WS	MB, WS	User-specific
Login Token (LT)	128	WS	WS, MB, WC, MF	5 mins
Access Token (AT)	512	WS	WS, MB	120 mins

Table 3: Summary of various keys used in WeChat ecosystem.

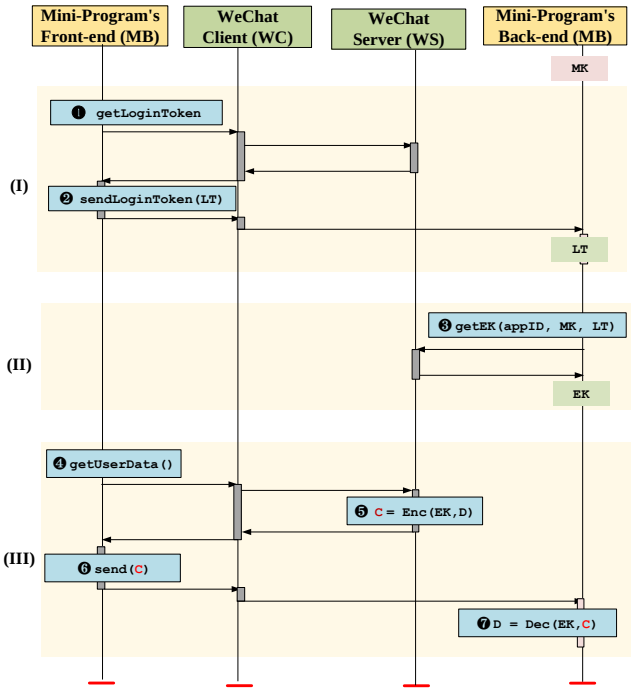

Figure 1: Sensitive data access protocols in WeChat mini-programs

3.2 Cryptographic Access Control

The MK is a vital key for cryptographic access control in mini-programs, generated by WeChat upon authentication. WeChat offers two MK-based protocols for accessing resources: encryption-based for sensitive data and token-based for sensitive services.

Encryption-based Access Control. To safeguard sensitive user data from abuse or manipulation, WeChat encrypts it when accessed by mini-programs through APIs. The encrypted data is sent back to the mini-program's back-end for decryption and processing, following a three-phase process shown in Figure 1:

Keys	Data	Services	Feasible
MK	Manipulating Data (✓)	Consuming Services for Free (✓)	✓
EK	Manipulating Data (✓)	N/A	✓
LT	Collecting Other's Data (✗)	N/A	✗
AT	N/A	Consuming Services for Free (✓)	✓

Table 4: Summary of possible attacks. "✓" means the corresponding attack works, while "✗" means does not.

(I) **Login Token (LT) Acquisition.** WeChat assigns a unique appID and MK to each mini-program to verify their identity and developers. However, since sensitive data is linked to users rather than mini-programs, appID and MK alone are insufficient. To ensure users only access their data, WeChat generates a user-specific LT when they log in, which is combined with the appID and MK to obtain the EK (**Step ❶**). As shown in Table 3, LT is a 128-bit hex string valid for only five minutes [18], generated during user login to the WeChat server. MF delivers it to MB for querying EK within the time window (**Step ❷**). This design prevents unauthorized collection of user data at scale by MB.

(II) **Encryption Key (EK) Fetching (❸).** EK encrypts user data and requires appID, MK, and LT for retrieval. To balance protection against enumeration attacks and mini-program data processing performance, the EK has a dynamic expiration period, set to five minutes by default [18]. Its expiration time may be extended or reduced based on API usage frequency. Although LT is discarded after obtaining the EK, there's a possibility of two separate EK acquisitions returning the same EK within the expiration period, despite different LTs.

(III) **Sensitive Information Encryption and Decryption.** MF uses data fetching APIs to retrieve user data (*e.g.*, `getPhoneNumber`) from WS (**Step ❹**). WS identifies the user and mini-program from WC's request, encrypts sensitive data using the corresponding EK from stage-II, and sends encrypted data to WC (**Step ❺**). WC forwards the encrypted data to its back-end (**Step ❻**). Decryption should only occur at the back-end using the EK obtained in stage-III, since the front-end is untrusted. The back-end decrypts the data (**Step ❼**) to obtain the plaintext of sensitive data. The MB may then use this data for specific business logic, such as retrieving app-specific data based on decrypted phone number.

Token-based Access Control. WeChat platform has provided various sensitive services that encapsulate complex functionalities such as AI chat bot or OCR for mini-program developers to purchase and use. The use of these services is quite straightforward, which is a two-step process: (i) the MB provides the appID and the mini-program MK to the WS in exchange for API access token AT (Step ①); (ii) then, when invoking services provided by WeChat, the MB has to attach both AT and the data sent to WS (Step ②). As shown in Table 3, AT is 512 bits long and valid for 2 hours.

3.3 Threat Model of WeChat

WeChat has defined four types of keys, each with a distinct purpose, and WeChat has made different assumptions about the security of these keys.

- MK serves as the root of WeChat security and is used for querying EK and AT. WeChat assumes that the MK should be strictly held within the MS and never disclosed to MF or any other parties.
- EK is responsible for encrypting and decrypting sensitive information, such as phone numbers. It is queried by MS and kept there until it expires. WeChat assumes that only MS, and not MF, can retrieve the EK and that all encrypted data must be consumed at MS using the EK.
- **LT** is crucial for authenticating a user, as it is user-specific and only the data owner can produce a valid LT based on the user's credentials. Additionally, LT is used as the index for querying the session key. WeChat assumes that attackers cannot obtain LT, as doing so would require them not only to install malware, but also to obtain root privileges to access WeChat's data.
- **AT** is a critical key for enabling the consumption of paid online services. It is queried by MS and remains stored there until it expires, typically after two hours. WeChat assumes that the consumption of services using AT is carried out by MS instead of MF.

3.4 Attack Surface Analysis

While developers are responsible for managing the MK, some may not adhere to WeChat's development guidelines [2]. This can result in developers hardcoding or distributing keys to the wrong party, such as distributing MS keys to MF, leading to key leaks to attackers. Violation of the underlying assumptions can result in possible attacks, as shown in Table 4.

- **With the Leak of MK.** Developers may hardcode the MK in MF, creating opportunities for attackers to obtain it through unpacking the MF. If the MK is leaked, attackers can always obtain the EK to encrypt and decrypt data, compromising confidentiality and integrity. They can also use the MK to obtain an access token and consume services for free. As such, the first assumption made by WeChat may not always hold true.
- **With the Leak of EK.** It could be possible that careless mini-program developers can accidentally send the EK obtained on MB to MF. If attackers obtain the EK, again, the attacker can launch data manipulation attacks, although this attack can last for a short period of time in a particular session and the attacker has to keep refreshing EK from the mini-program servers. In addition, in the event that the MK is leaked, the attacker is able to obtain EK through the encryption-based access control protocol, though this would be limited to the attacker's own EK. As a result, the second assumption made by WeChat may not always hold true.
- **With the Leak of LT.** Since LT is linked to an individual user, attackers can create a malicious mini-program with their own MK to retrieve and decrypt all sensitive information of a user who leaks their LT. However, obtaining the LT is difficult, as user authorization is required explicitly. Though attackers can try to hack into the user's account or break WeChat's mechanism to generate a valid LT associated with a specific

user, these methods are not practical. Therefore, while attackers can obtain their own LT, they cannot obtain others' LT. Third assumption made by WeChat remains valid.

- **With the Leak of AT.** If attackers obtain an AT, they can misuse it for up to two hours before it expires. Within this timeframe, they can exploit the compromised AT to consume services that may not be free, resulting in financial charges against the developers. Although it may not be practical for attackers to obtain an AT by compromising the back-end of the mini-program, developers may mistakenly distribute ATs to MF, and attackers can also query ATs using a leaked MK. As a result, the fourth assumption made by WeChat can also be compromised.

4 MEASURING THE MK LEAKS

Our investigation confirmed that the keys could be leaked in the front-end of a mini-program, as revealed by the attack surface (§3.4). However, it remains unclear how widespread key leaks are among mini-programs. Therefore, we have conducted a measurement study to answer the following research questions:

- **RQ1:** What are the categories of the MK leaked mini-programs?
- **RQ2:** What are their ratings?
- **RQ3:** What are the their accessed resources?
- **RQ4:** Who are their developers?
- **RQ5:** When are their latest update?
- **RQ6:** Are there any high profile MK leaked mini-programs?

4.1 Scope and Methodologies

Scope. In this study, we focus on systematically studying MK leaks to understand their prevalence, root causes, and consequences. We do not aim to detect attacks caused by leakage of the EK, LT, or AT. This is because as outlined in §3.3 and §3.4, a leaked MK can result in the exposure of EK and AT, and it is not possible for LT to be leaked. We use the example of WeChat to explain MK-based access control protocols, but note that similar protocols are used on other mini-program platforms, such as Baidu [4] and Alipay. Therefore, we focus on WeChat in particular due to its popularity and support for both sensitive data access and cloud service access.

Collecting Mini-Programs. We used the `innersearch` API (obtained through reverse engineering of WeChat) to search for and download mini-programs, and the `waVerifyInfo` API to collect developer information [3], if available (as shown in Table 5). We employed 1,000 commonly used Chinese characters and 1,000 commonly used English words as seed keywords, expanding them based on the names and descriptions of the collected mini-programs, resulting in 14,020 keywords that retrieved 3,450,586 mini-programs. Note that the WeChat market has about 4 million mini-programs [5], making our dataset likely to cover the majority of them.

Identifying the MK Leaked Mini-Programs. We checked if the mini-programs had leaked their MK(s) by identifying all 256-bit hexadecimal digit strings and pruning them based on the MK validation API (the fourth API in Table 5). We used a regular expression with a 32-byte hex-string format of `[a-f0-9]32` to search for possible

API	Description
Mp.wx.qq.com/wxa-cgi/innersearch	Mini-program Searching API
Parameters	
├query	A keyword to search the mini-programs
├client_version	The WeChat client version
├time	The timestamp of this query
├cookie	WeChat client token
└ ...	Other omitted parameters
Return value	
├appID	The app ID of the mini-program
├label	The category of the mini-program
├rating	The user rating of the mini-program
└ ...	Other omitted fields in the return value
Wx.nativgateToMiniProgram()	Mini-program Downloading API
Parameters	
├appID	The app ID of the mini-program
└version	The version (released or debug)
Return value	The packed file of the mini-program
Mp.weixin.qq.com/mp/waVerifyinfo	Developer Information Downloading API
Parameters	
├appID	The app ID of the mini-program
├deviceType	The device type of mobile (e.g., Android)
└netType	Network Connection Information (e.g., WiFi)
Return value	
├developerInfo	The developer information of the mini-program
└updateDate	The relase date of the mini-program
Api.weixin.qq.com/cgi-bin/token	A MK Validation API
Parameters	
├appID	The app ID of the mini-program
├secret	A MK
└grant_type	The access type
Return value	An access token or error message
Api.weixin.qq.com/sns/jscode2session	EK query API (`getEK()`)
Parameters	
├appID	The app ID of the mini-program
├secret	The Master key (MK)
└jscode	A login token (LT)
Return value	The encryption key (EK)

Table 5: The five important APIs used in our study.

MKs in the mini-program code and validated them using the MK validation API. The API returns the actual AT if the MK is correct, else it returns an error code. All mini-programs' backend, by default, has its corresponding AT due to free cloud services like `Translation` as reported in Table 1. With the downloaded 3,450,586 mini-programs, we identified 40,880 mini-programs that leaked their MKs.

4.2 The Results

With the detected 40,880 mini-programs, we next seek to answer the research questions set out earlier by analyzing their category, ratings, accessed resources, their developers, and last update. In the following, we provide these results in greater details.

(RQ1) Categories of the MK Leaked Mini-programs. Most mini-programs in the WeChat mini-program store have been classified into specific categories, as shown in Table 6. We observed that the percentage of MK leaked mini-programs and the data or services they accessed (e.g., phone number, user info, share info, werun info, AI, security, and location) was diverse, indicating that MK leakage was widespread across mini-program categories. However, some categories (e.g., lifestyles and shopping) had a slightly higher number of vulnerable mini-programs due to the larger number of mini-programs in those categories. On the other hand, tool, government, and games had a higher percentage of vulnerable mini-programs. Unfortunately, all these mini-programs are crucial for users' daily lives as they often provide essential services (e.g., QR code scanning), involve real user identity information and critical

Category	Sensitive Data												Cloud Services								
	Phone number (A1)			User info (A1)			Share info (A2)			Werun info (A2)			AI (A3)			Security (A3)			Location (A3)		
	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln
Business	0	242	0.59	8	644	1.59	0	49	0.12	0	9	0.02	0	5	0.01	10	855	2.12	0	0	0.00
Education	1	775	1.90	114	3,065	7.78	15	635	1.59	0	33	0.08	0	6	0.01	132	3,957	10.00	0	1	0.00
e-Learning	0	39	0.10	2	103	0.26	2	22	0.06	0	9	0.02	0	0	0.00	37	134	0.42	0	0	0.00
Entertainment	0	42	0.10	6	234	0.59	3	74	0.19	0	2	0.00	0	0	0.00	11	268	0.68	0	0	0.00
Finance	0	33	0.08	0	74	0.18	0	10	0.02	0	3	0.01	0	2	0.00	0	96	0.23	0	0	0.00
Food	0	511	1.25	8	1,369	3.37	0	158	0.39	0	6	0.01	0	0	0.00	16	1,846	4.55	0	0	0.00
Games	1	23	0.06	133	422	1.36	92	170	0.64	0	3	0.01	0	0	0.00	250	473	1.77	0	0	0.00
Government	0	141	0.34	24	524	1.34	0	47	0.11	0	7	0.02	0	3	0.01	24	700	1.77	0	0	0.00
Health	0	161	0.39	4	448	1.11	1	47	0.12	1	11	0.03	0	4	0.01	4	559	1.38	0	0	0.00
Photo	0	2	0.00	1	32	0.08	0	4	0.01	0	1	0.00	0	0	0.00	1	36	0.09	0	0	0.00
Job	0	101	0.25	1	275	0.68	0	36	0.09	0	1	0.00	0	2	0.00	2	390	0.96	0	0	0.00
Lifestyle	5	1,440	3.53	93	4,183	10.46	4	590	1.45	0	22	0.05	0	9	0.02	124	5,658	14.14	0	1	0.00
Shopping	0	1,885	4.61	41	7,174	17.65	6	1,434	3.52	1	65	0.16	0	9	0.02	52	9,504	23.38	0	1	0.00
Social	1	55	0.14	13	282	0.72	0	62	0.15	0	6	0.01	0	3	0.01	15	320	0.82	0	0	0.00
Sports	1	38	0.10	6	224	0.56	1	25	0.06	3	44	0.11	0	0	0.00	8	253	0.64	0	0	0.00
Tool	8	761	1.88	181	3,332	8.59	33	432	1.14	8	43	0.12	0	17	0.04	466	4,327	11.72	2	2	0.01
Traffic	1	161	0.40	6	465	1.15	0	33	0.08	0	2	0.00	0	4	0.01	8	665	1.65	0	1	0.00
Travelling	0	31	0.08	1	137	0.34	0	12	0.03	0	0	0.00	0	0	0.00	2	191	0.47	0	0	0.00
Uncategorized	0	0	0.00	0	3	0.01	0	1	0.00	0	0	0.00	0	0	0.00	0	5	0.01	0	0	0.00
TOTAL	18	6,441	15.80	642	22,990	57.81	157	3,841	9.78	13	267	0.68	0	64	0.16	1,162	30,237	76.81	2	6	0.02

Table 6: Statistics of the identified vulnerable mini-programs. Note that ind. stands for individual developers and com. stands for enterprise developers (i.e., by company).

Category	Ratings distribution																	
	1.0-1.9			2.0-2.9			3.0-3.9			4.0-4.9			5.0			Unrated		
	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln	# ind.	# com.	% vuln
Business	0	0	0.00	0	1	0.00	0	4	0.01	0	38	0.09	0	4	0.01	12	812	2.02
Education	0	1	0.00	0	4	0.01	0	38	0.09	7	216	0.55	0	15	0.04	126	3,714	9.39
e-Learning	0	0	0.00	0	2	0.00	0	4	0.01	0	11	0.03	0	1	0.00	37	118	0.38
Entertainment	0	2	0.00	0	7	0.02	2	17	0.05	1	23	0.06	0	0	0.00	8	226	0.57
Finance	0	0	0.00	0	2	0.00	0	2	0.00	0	11	0.03	0	0	0.00	0	82	0.20
Food	0	0	0.00	0	1	0.00	0	2	0.00	0	49	0.12	0	2	0.00	16	1,795	4.43
Games	0	0	0.00	18	23	0.10	63	124	0.46	9	37	0.11	0	0	0.00	161	294	1.11
Government	0	0	0.00	0	2	0.00	0	7	0.02	0	48	0.12	0	1	0.00	24	645	1.64
Health	0	0	0.00	0	1	0.00	0	3	0.01	0	49	0.12	0	2	0.00	4	510	1.26
Photo	0	1	0.00	0	0	0.00	0	2	0.00	1	7	0.02	0	0	0.00	0	27	0.07
Job	0	0	0.00	0	1	0.00	0	10	0.02	0	29	0.07	0	1	0.00	2	351	0.86
Lifestyle	0	0	0.00	0	1	0.00	0	37	0.09	3	227	0.56	1	19	0.05	120	5,398	13.50
Shopping	0	2	0.00	0	0	0.00	2	23	0.06	10	299	0.76	0	26	0.06	42	9,205	22.62
Social	0	0	0.00	0	3	0.01	0	5	0.01	1	38	0.10	0	0	0.00	14	279	0.72
Sports	0	0	0.00	0	0	0.00	0	2	0.00	0	33	0.08	1	6	0.02	7	213	0.54
Tool	0	3	0.01	3	24	0.07	4	48	0.13	19	325	0.84	4	20	0.06	441	3,955	10.75
Traffic	0	0	0.00	0	5	0.01	0	16	0.04	0	71	0.17	0	3	0.01	8	574	1.42
Travelling	0	0	0.00	0	0	0.00	0	3	0.01	0	12	0.03	0	1	0.00	2	175	0.43
Uncategorized	0	0	0.00	0	0	0.00	0	0	0.00	0	0	0.00	0	0	0.00	0	5	0.01
TOTAL	0	9	0.02	21	77	0.24	71	347	1.02	51	1,523	3.85	6	101	0.26	1,024	28,378	71.92

Table 7: Distribution of the detailed ratings of mini-programs

administrative services (e.g., government tools), or involve payment (e.g., shopping) and in-game trading.

(RQ2) Ratings of the MK Leaked Mini-programs. Compared to traditional platforms like Google Play, the mini-program store does not furnish information about the total number of downloads for each mini-program. Nevertheless, we can estimate its popularity by evaluating its rating. It is worth noting that the rating becomes available only after a sufficient number of users have employed and rated the corresponding mini-program [47]. By utilizing this rating, we can gauge the popularity of a mini-program and link it to the corresponding mini-program category, as demonstrated in Table 7. The table illustrates that mini-programs with lower ratings are more likely to have a greater number and percentage of vulnerable mini-programs (i.e., MK leaked mini-programs). This is likely because developers of mini-programs with higher ratings may devote more effort to ensuring security and privacy to maintain their competitive edge.

(RQ3) Accessed Resources of the MK Leaked Mini-programs. As shown in §5, attackers can use different attacks depending

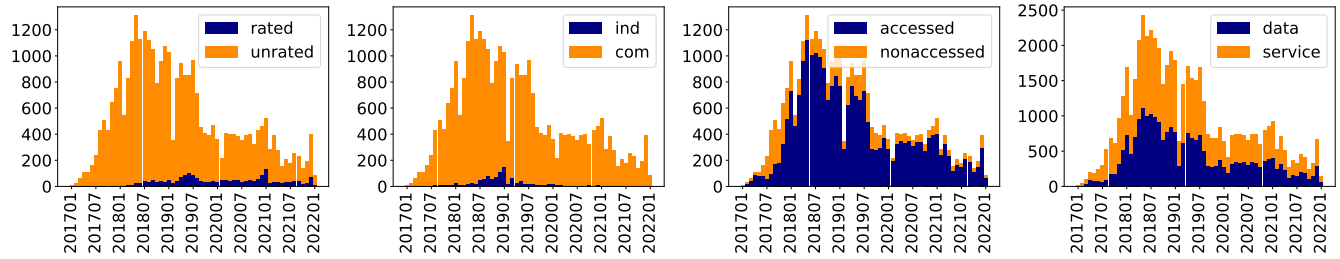

Figure 2: The trending of all vulnerable mini-programs w.r.t rating, developer, accessed resource type, and accessed data type.

# Associated Apps	Rated		Unrated		Total	
	# com	# apps	# com	# apps	# com	# apps
>=20	0	0	16	417	16	417
19	0	0	2	38	2	38
18	0	0	2	36	2	36
17	0	0	1	17	1	17
16	0	0	3	48	3	48
15	1	15	2	30	3	45
14	0	0	1	14	1	14
13	1	13	4	52	5	65
12	3	36	6	72	9	108
11	0	0	1	11	1	11
10	0	0	9	90	9	90
9	3	27	9	81	12	108
8	1	8	24	192	25	200
7	1	7	29	203	30	210
6	1	6	26	156	27	162
5	6	30	69	345	75	375
4	7	28	155	620	161	648
3	21	63	333	999	353	1,062
2	100	200	1,300	2,600	1,388	2,800
1	1,624	1,624	22,357	22,357	23,805	23,981
TOTAL	1,769	2,057	24,349	28,378	25,781	30,435

Table 8: Numbers of vulnerable mini-programs associated with the same software development enterprises.

on the sensitive resources accessed by mini-programs. To understand the impact of MK leaks in these vulnerable mini-programs, we present statistics in Table 6 based on APIs used. In particular, we examined the API names directly in the code to identify mini-programs that utilized sensitive data. Moreover, we employed `api.weixin.qq.com/cgi-bin/get_current_selfmenu_info` to inspect the specific services utilized by a mini-program (without invoking these services to verify their availability). Our findings show that 24,701 mini-programs accessed sensitive data and accessed at least one cloud service, making them vulnerable to attacks such as account hijacking (A1), promotion abuse (A2), and service theft (A3) (The details of those attacks will be introduced in §5). Interestingly, we also discovered that there are 6,907 vulnerable mini-programs that do not access any MK-related resources at all. However, these mini-programs may still be vulnerable since they could potentially be utilized to access MK-related resources in the future.

(RQ4) The Developers of the MK Leaked Mini-programs. We now investigate the developers of these vulnerable mini-programs and their frequency of errors. There are two types of developers: individual and enterprise. *Tencent* provides enterprise name and tax ID for enterprise developers, but not for individual developers. Thus, we can only present statistics for vulnerable mini-programs developed by companies, excluding 4,676 developed by individuals. For companies, we collected developer information if available, and Table 8 shows the statistics of associated mini-programs. Most companies developed only one vulnerable mini-program, but some, such as "Suzhou Yutianxia", developed 30.

(RQ5) Latest Update of the MK Leaked Mini-programs. The mini-program's metadata includes the latest updated timestamp, allowing us to determine how long the MK leakage vulnerability has existed. We also analyze the resources accessed, ratings, and developer information grouped by the latest update month to observe trends over time (Figure 2). Our observations are as follows: (i) MK leakage vulnerability has been there since the first year when *Tencent* introduced the mini-program into WeChat (e.g., a Calendar mini-program leaked the MK in January 2017 and was never updated). (ii) Over time, the number of vulnerable mini-programs decreased regardless of the accessed data resource or rating, but those developed by companies still constitute a significant portion of the vulnerable mini-programs.

(RQ6) High Profile MK Leaked Mini-programs. Lastly, we investigate the implications of our attacks by identifying the vulnerable mini-programs. We found 107 mini-programs developed by Fortune top-500 companies, and those with multiple vulnerable mini-programs are listed in Table 9. Notably, (i) some high-profile companies made the same mistake across multiple mini-programs (e.g., Nestle); (ii) Tencent, the WeChat provider, is the top-ranked company with MK leaks in their mini-programs; (iii) some mini-programs from high-profile companies only leaked their MK without accessing sensitive information. As discussed, while these mini-programs cannot be attacked currently, they remain vulnerable if new functionalities are added (*e.g.*, phone number fetching).

5 EXPLOITING THE MK LEAKS

We have found vulnerable mini-programs that exposed their app secrets, allowing for two types of attacks: (i) attacks against sensitive data (§5.1), where the attacker aims to modify sensitive data; and (ii) attacks against cloud services (§5.2), where the attacker uses the victim mini-program's paid cloud services without cost.

5.1 Attacks Against Sensitive Data

Attack Workflow. In our threat model, the attacker can obtain the mini-program front-end package and manipulate network packets. This allows them to unpack the program and trigger sensitive data

Company	Ratings	Category	Data	Attacks
Tencent	5.0	games	①	A1
	5.0	tool	①	A1
	5.0	entertainment	①,②	A1, A2
	4.6	health		
	5.0	entertainment	①	A1
	4.8	tool	①	A1
	4.7	education		
	4.6	games	①	A1
	4.7	social	①	A1
	4.6	tool	①	A1
	5.0	tool		
	5.0	e-learning	③	A2
	5.0	health	①	A1
	5.0	games	①	A1
		games	①	A1
		education		A3
	5.0	games	②	A2
Nestle	4.3	food	③,①	A1, A2
	5.0	tool	③,①	A1, A2
		lifestyle	③,①	A1, A2
	4.9	food	③	A2
	4.8	shopping	③,①	A1, A2
	4.9	shopping	③,①	A1, A2
Sanofi	4.7	health	③,①	A1, A2
	5.0	business	③,①	A1, A2
	5.0	business	③,①	A1, A2
		business	③,①	A1, A2
China Unicom	5.0	finance	③	A2
	5.0	finance	③,①	A1, A2
	4.8	tool		
	4.1	shopping	③	A2
Bank Of China	4.3	shopping	①	A1
	4.3	shopping	①	A1
		business	①	A1
Hitachi	5.0	tool	①	A1
	5.0	tool	①	A1
	5.0	business	①	A1
Shou Gang		tool		A3
	5.0	lifestyle		
China Mobile	4.3	finance		
	4.7	finance		
ICBC	5.0	lifestyle	③	A2
		traffic	①,②,④	
Volkswagen	5.0	shopping	①	A1
		shopping	①,②	

Table 9: The top-10 Fortune 500 companies that have the most number of mini-programs leaked the MK. ① user info; ② shared info; ③ phone number; ④ werun data.

communication for retrieval or manipulation. The attack workflow involves two steps as shown in Figure 3:

(I) **Obtaining Attacker's Encryption Key (EK).** To start the attack, assume an attacker *Eve*, he needs to first get his own LT, denoted LT_{eve}, from the WeChat server (**Step ❶**). Next, he provides the leaked MK, the mini-program appID (easily obtained in the meta-data of a mini-program), and LT_{eve} to the WeChat server (**Step ❷**). Since MK, appID, and LT_{eve} are all valid, the WeChat server will send his EK (*i.e.*, EK_{eve}) as what it does typically for the mini-program's back-end to *Eve*. Note that prior to this login, the server may already have a copy of EK_{eve} if *Eve* has logged in before the mini-program server. Also, if WeChat server has refreshed the EK_{eve}, *Eve* will get a new one, and the mini-program back-end will detect the EK change as well (at **Step ❻**) and request for this new EK_{eve} from the WeChat server by asking *Eve* to provide his LT_{eve} again (details are omitted here for brevity). Therefore, it does not matter whether the mini-program server has a copy of EK_{eve} or not.

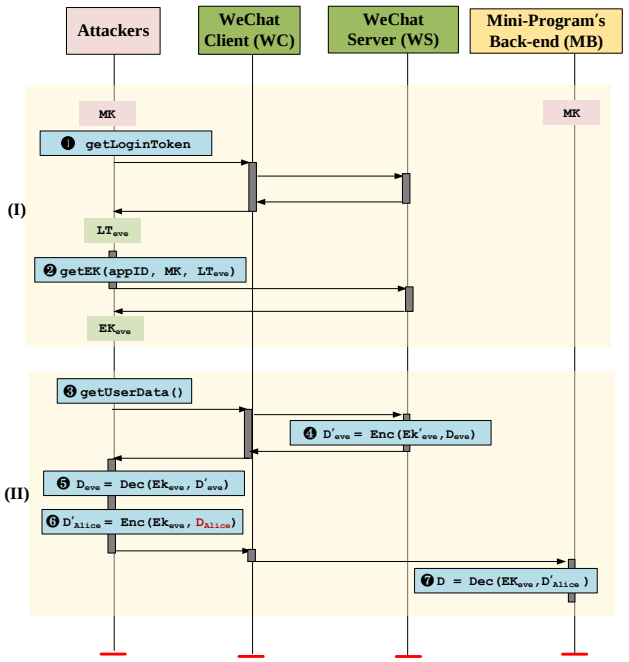

Figure 3: Attacks Against Sensitive Data

(II) **Sensitive Data Retrieval and/or Manipulation.** In this stage, the attacker *Eve* initiates a request for sensitive data (*e.g.*, getPhoneNumber) and prompts WeChat client to retrieve his or her own data (Step ❸). The WeChat client sends a request to its server, which encrypts the requested data such as *Eve*'s Phone number D_{eve} (Step ❹). The server returns the encrypted data, represented by D'_{eve}, to *Eve*, who can then choose to discard it and manipulate the information. Nevertheless, the attacker must first decrypt the cipher to examine the packet format since they are unaware of the data's structure (Step ❺). Only then can the attacker create falsified data. Next, the attacker re-encrypts the fake data using the same EK (**Step ❻**). For example, *Eve* can change his phone number to Alice's (*i.e.*, D_{Alice}). Since the mini-program's backend has the same EK as the attacker, it can decrypt the cipher and obtain the modified data (*i.e.*, phone number D_{Alice}) (**Step ❼**). If there is no consistency check between the phone number and the EK (likely true at this moment according to our experiment), the attacker can break into Alice's account.

Now, the attacker can directly obtain and modify the information of interests. It turns out such data manipulation can cause severe consequences. In the following, we would like to demonstrate the implication of these attacks by using two concrete examples unique in the super apps: account hijacking and promotion abuse.

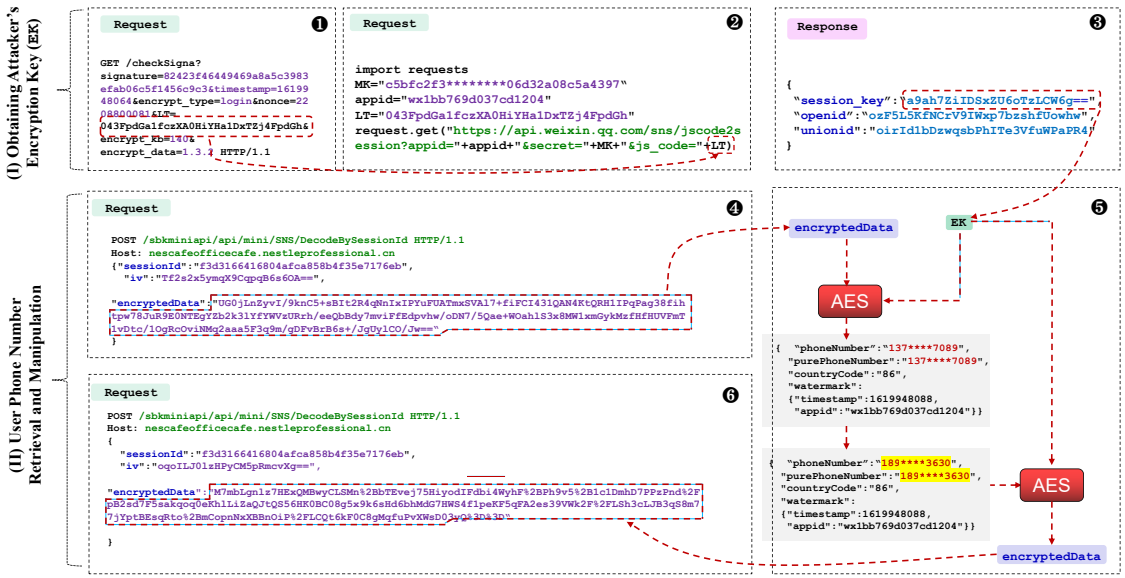

Figure 4: An excerpt of attack traffic traces

(A1) Account Hijacking Attacks. Since a mini-program's back-end usually needs to maintain a database to keep user's record and this database needs to be indexed, the mini-program back-ends usually use predictable data such as the phone number to index each user (this is particularly true in China since most citizen has a cell phone number, and this number has been used almost as an identity). As such, with MK, the attacker can retrieve the victim's information by changing the predictable data (*e.g.*, the phone number) to a victim's, logging into other users' accounts for illicit purposes. For instance, he or she can steal the discount offers or even free offers given to high-value loyal customers, such as hotel free night certificates or rewards points.

We now present such a concrete attack against a mini-program named "We Proudly Serve" (WPS) from *Nestle* to demonstrate its consequences. Note that this mini-program has fixed the vulnerability after we made the responsible disclosure. Specifically, to carry out our attack, we first registered two accounts, a victim account and an attack account, and then executed the following steps:

(I) **Obtaining Attacker's Encryption Key (EK).** In this stage, we first got LT from WeChat server, and used the obtained LT, mini-program appID, and the leaked MK to query the encryption key EK (**Step ❶**). To get the LT from WeChat server, we used a well-known MITM proxy Burp Suite [39] to inspect the traffic between the WeChat client and its server. By default, when a victim mini-program invokes wx.login, the LT will be delivered to the front-end and require the front-end to send it to the back-end. In our attack, we intercepted the LT (i.e., the encrypt_code in Figure 4) without sending it to the back-end as shown in Figure 4. Next, we programmed a python script to query the EK from WeChat server, and the obtained LT was then fed to the getEK() API (**Step ❷**). Specifically, getEK() takes three inputs, including the public accessible appID, the secret (*i.e.*, MK), and Js_code (*i.e.*, the LT obtained in **Step ❶**). As shown in Figure 4, WeChat server returned the encryption key EK, which is encoded using Base64 (**Step ❸**).

(II) **Phone Number Retrieval and Manipulation**. In this stage, we intentionally triggered WPS's front-end to invoke getPhoneNumber, and manipulated the encrypted phone number sent from the WeChat server to trick the back-end of WPS into believing it is interacting with the victim's account. More specifically, we first triggered WPS to invoke getPhoneNumber. getPhoneNumber is a callback function, which is called when we click the corresponding button. Then we intercepted the cipher that is about to be sent to the mini-program back-end (**Step ❹**), and this cipher was initially received by the mini-program from WeChat server when invoking the getPhoneNumber API. According to WeChat's official document [18], WeChat uses the standard cryptographic algorithms, and therefore, we easily implemented our own script to decrypt and encrypt the encryptedData using the obtained EK, and inspected its format, so that we can forge a fake phone number in the following steps (**Step ❺**). After inspecting the format of the decrypted plain-text, we then replaced the attacker's phone number (*i.e.*,137****7089) with our victim's (*i.e.*, 189****3630"), and encrypted the modified phone number using EK. At this time, the attacker can send the encrypted data (the plaintext of which is the modified phone number) to the back-end of WPS (**Step ❻**). As discussed, the EK used by the backend of WPS to decrypt the phone number and EK used by the attacker to encrypt the phone number are the same, and therefore, our modified phone number was successfully decrypted by the back-end of WPS, and used by WPS to retrieve the user information. It is interesting to note that much of the retrieved information is collected and maintained by WPS,

meaning that if we hijack multiple accounts by enumerating all the phone numbers, we can harvest a large amount of WPS's user information.

Even with root access to a phone, attackers cannot manipulate encrypted data without the MK. While plaintext phone numbers can be changed during user registration, they are protected by SMS authentication. Later on, all phone numbers are encrypted and sent directly to the WeChat server, never exposing plaintext to the front-end of the mini-program or WeChat app. Without MK, attackers won't be able to mount the account hijacking.

(A2) Promotion Abuse Attacks. Attackers can manipulate sensitive data other than phone numbers, such as `WeRunData` and `ShareInfo`. In the mini-program paradigm, promoting services to more users is crucial for success, and convenient access to social networks empowers mini-programs to promote products more effectively. Some mini-programs offer incentives, such as sharing promotions via group chats, which can range from trial products to real cash. However, attackers who manipulate encrypted `WeRunData` or `ShareInfo` can easily exploit these promotions and earn rewards without actually sharing promotion information to multiple group chats by manipulating group chat identifiers.

Case I: Promotion Abuse via `WeRunData` Manipulation. Promotion abuse attacks can be achieved through manipulating `WeRunData` data. For example, we discovered that a banking mini-program from Industrial and Commercial Bank of China (ICBC) that uses the `WeRunData` for promotion where users gain the "energetic value" with the steps they walked each day, and such a value can be used to redeem for gifts in the shop. By inspecting the unpacked JavaScript code, we found that `WeRunData` is obtained with `getWeRunData` and then the encrypted data is sent to the back-end via AJAX request. However, as the `MK` is leaked, the attacker can abuse these promotions by arbitrarily setting the daily steps walked across a week and sending the request to the mini-program back-end. Specifically, the attacker can first use `EK` to decrypt `t.encryptedData` and change the original daily steps from, for example, zero to a hundred thousand. Given that the attacker can have multiple accounts, theoretically, the attacker can perform the attack at a scale to harm the vendors.

Case II: Promotion Abuse via `ShareInfo` Manipulation. Aside from `WeRunData`, promotion abuse attacks can also be achieved by manipulating `ShareInfo`. We discovered that numerous mini-programs reward their users with red packets, a type of monetary gift, when they share the mini-program with their group chats. This is accomplished by fetching the group chat ID returned by `getShareInfo`. For example, as shown in Figure 5, we found that there is a mini-program that rewards users with red packets for sharing with a unique group chat. The mini-program's back-end verifies the encrypted data returned by `getShareInfo` to ensure that the sharing is genuine and prevent users from cheating. However, similar to the previous case, with the leaked `MK`, an attacker can exploit this security measure by repeatedly sharing the program and modifying the group chat ID. We have tested this mini-program by playing it normally and noticed that it gives 1 to 10 cents of random red packets to users when sharing this mini-program with a unique group. Given that it takes around 3 seconds at most to share a mini-program to a groupchat, an attacker can theoretically get $20,000 dollars worth of red packets in a week.

```
onShareAppMessage: function(a){
  var e=parseInt(a.target.dataset.info), return {
    success: function(){
      wx.getShareInfo({
        shareTicket: t,
        success: function(a){
          this.shareJudge(t, a.encryptedData, a.iv);
        }
      })
    }
  }
}
shareJudge: function(a, e, i){
  s=setConfig.url+"/api/enve/shareGetTimes", l={
    token: this.globalData.token,
    shareTicket: a,
    encryptedData: e,
    iv: i
  };
  t.request(s, l, function(a){
    2e4 == a.data.code ? (wx.showToast({
      title: "share success"
    })) : wx.showToast({
      title: "please share to differerent groupchat"
    })
  }
}
```

Figure 5: Promotion Abuse via `ShareInfo` Manipulation

5.2 Attacks against Paid Cloud Services

Attack Workflow. Since `MK` is also used to access the cloud services provided by WeChat, the attacker may use the leaked `MK` to fetch its corresponding access token AT and consume the paid services for free. The attacker takes place with the role of mini-program's back-end, and launches the attacks with the following two steps:

(I) **Obtaining the API Access Token (AT).** To obtain the API Access Token, the attacker just needs to send the `MK` and the victim mini-program's `appID` to the WeChat server, which will reply the corresponding AT if both the `MK` and `appID` are legitimate, allowing attacker to gain access to all the services enabled by the victim (**Step** ①).

(II) **Service Invocation.** In this stage, the attacker just needs to send a request to the WeChat server with the obtained AT, victim's `appID`, and the JSON object containing parameters to invoke the enabled services. The server will return the result as invoked by the victim mini-program (**Step** ②). An online OCR API charges $1,000 for every 1 million invocations. Attackers could exploit the service by creating their own mini-program or query interface that consumes the OCR service for free using the leaked `MK`, potentially causing a loss of thousands of dollars.

(A3) Service Theft Attacks. All vulnerable mini-programs using cloud services were found to be susceptible to service theft, allowing attackers to invoke paid services for free once they obtain the corresponding access token using the leaked `MK`. We investigated a few examples, including Tencent's medical appointment platform, which used location services for searches, and an online game requiring identity verification using OCR (in China, an online game user may not be able to register for a game account unless he or she presents a valid identity, which prevents children from 18 years and younger from game addiction). Additionally, several mini-game

programs enabled WeChat's security service to identify user risk levels. Exploiting the leaked MK, an attacker could use these services to build an online portal for commercial use without paying for anything to Tencent.

6 DISCUSSIONS

6.1 Mitigation and Prevention

Fundamentally, the MK leakage problem is a cryptogrphic key management problem. Since the WeChat client cannot be trusted, we must rely on both the WeChat server and the mini-program back-end to properly manage the keys. At a high level, there could be three possible approaches to mitigate or even prevent MK leaks.

- **Vetting the mini-programs systematically.** One mitigation solution is to use the detection method introduced in §4.1 to vet mini-programs. In particular, when a mini-program is submitted, WeChat must extract the byte code from the mini-program, and detect all possible MK leaks. A mini-program that fails to meet the vetting requirements will not be allowed to be released. Since WeChat controls the life cycle of the MK, it would be easier for them to perform such a detection. For example, in our analysis, to confirm whether the grepped hex strings are MKs, we have to send each MK to WeChat server, and analyze the responses. However, WeChat can fetch these keys directly from their databases.
- **Removing the encrypted data forwarding from the untrusted client when requesting sensitive data.** According to the sensitive data access protocols described in Figure 1, WeChat should have redesigned Step ❺-❼ by removing the data forwarding step at Step ❻. This is because the current design provides an opportunity for attackers to modify the ciphertext sent by the WeChat server. Consequently, WeChat should let `getUserData` API communicate to the mini-program backend first, and let the back-end directly get the data from the WeChat server, instead of allowing the client to request and then forward. However, this defense may require a completely new mechanism to make it secure. Currently, AT is directly fetched by the MS from WS, since AT does not need the user's authorization (e.g., AT is only used to consume the services of WS). However, if we directly let the two back-ends exchange the encrypted data (without the user's authorization, i.e., LT, sent from the front end), the MS may try to crawl all the users' sensitive data for malicious purposes.
- **Using signing and integrity checking.** Since the data manipulation attacks we demonstrated involve the modification of the ciphertext of the messages sent by the WeChat server, WeChat can then first sign the encrypted message using *Tencent*'s private key, and then ask the mini-program back-end to verify the integrity of the message with *Tencent*'s public key, though this can have additional computation overhead. Encouragingly, we have noticed that WeChat has released a new API `auth.checkEncryptedData` [7] to check the data integrity based on this recommended defense.

It is important to note that the latter two defense can only fix the attacks against sensitive data. To mitigate against attacks against paid cloud services is relatively easy, and we can adopt traditional defenses in web security. For example, in the regular case, the service request is from different IP addresses and different accounts. As such, the back-end may set up the rate limit to prevent the services from being abused. In cases where the attacker with the leaked MK switches accounts and changes IP address, developers may have to update MK to trade off security and usability.

Super App	Name	Company	Back-end Stores MK	Data Access	Service Access	Download API	Verify API
WeChat [18]	APPSECRET	Tencent	✓	✓	✓	✓	✓
WeCom [18]	APPSECRET	Tencent	✓	✓	✓	✓	✓
QQ [33]	APPSECRET	Tencent	✓	✓	✓	✓	✓
Baidu [4]	CLIENT_SECRET	Baidu	✓	✓	✓	✓	✓
Alipay [8]	AES_KEY	Alibaba	✓	✓	✗	✓	✗
Taobao [8]	AES_KEY	Alibaba	✓	✓	✗	✓	✗
DingTalk [31]	APPSECRET	Alibaba	✓	✗	✓	✓	✓
Tiktok [13]	APPSECRET	Bytedance	✓	✓	✓	✓	✓
JINRI Toutiao [13]	APPSECRET	Bytedance	✓	✓	✓	✓	✓

Table 10: The list of super-apps that used cryptographic access control

6.2 Generality of our Study

We are confident that our findings on the WeChat mini-program paradigm can be applied to other platforms that have a similar design, such as QQ, Alipay, Baidu, Taobao, and TikTok. As shown in Table 10, these platforms may have different names for their app secrets, but they all have sensitive data or services for their mini-programs to access and are required to store their MK on their back-end systems. Thus, a compromised app secret could potentially lead to similar attacks on these mini-programs. We verified our findings by testing on Baidu [4], which confirmed that the attacks and the methods we used to detect MK leakage worked as expected. Particularly, we focused on Baidu for three reasons: (i) Baidu mini-programs are developed by Baidu Inc., which is independent from Tencent. (ii) Baidu has over 150,000 mini-programs, making it more popular than other platforms like DingTalk with only around 20,000 mini-programs. (iii) Baidu provides sensitive data and cloud services, unlike platforms such as Alipay that only offer access to sensitive data without service access.

First, our attacks are applicable to Baidu as they use a similar architecture to WeChat to protect sensitive data and cloud services. Particularly, Baidu encrypts sensitive data using a user-specific encryption key, which is later decrypted at the back-end using the retrieved encryption key. Similar to WeChat, Baidu uses the CLIENT_SECRET (i.e., MK) to query access tokens to consume their services. We tested our attacks on 10 Baidu mini-programs that leaked their CLIENT_SECRET, and all 10 were vulnerable to our attacks (we choose not to disclose their names due to ethics concerns). For example, we have been able to execute account hijacking and service theft attacks with success. However, due to the unavailability of APIs to access `ShareInfo` and `WeRunData` on Baidu, we are unable to carry out promotion abuse attacks. The attack details are similar to those used in WeChat, so we have omitted them for the sake of brevity. It's noteworthy that even though Baidu's documentation mentions that sensitive data is signed, they did not provide an API to verify the signature on the mini-programs' back-ends [4], which may explain why the tested mini-programs do not verify the signature and are vulnerable to attacks.

Second, our methodology for collecting and verifying the correctness of the MK for Baidu mini-programs is similar to that used

for WeChat. Specifically, `navigateToSmartMiniprogram` API can be used to download Baidu mini-programs by providing their appIDs, making our designed crawler easily consumable for Baidu mini-programs. Additionally, Baidu also has an API to verify the correctness of the MK. This API is necessary since encryption keys or tokens cannot be delivered to untrusted mini-program front-ends. Indeed, our investigation revealed that, besides Baidu and WeChat, all other platforms have mini-program downloading APIs, and 7 out of 9 platforms have an API to verify the MK. This suggests that the methodology for collecting mini-programs and verifying their secrets is similar across other platforms, in addition to WeChat and Baidu.

Third, the problem of MK leakage also exists in Baidu. We collected 171,989 Baidu mini-programs and used a regular expression to search for possible MKs in the mini-program code (`[a-zA-Z0-9]32`), which we validated using the MK validation API. Our findings show that 7,476 Baidu mini-programs (4.35%) have leaked their MKs. For readers interested in more details, we have included a detailed measurement for Baidu in our Appendix-A.

6.3 Ethics Concerns

During our experiments, we strictly adhered to community practices to prevent harm to victims or super apps. We respected rate limits defined by WeChat and Baidu servers throughout our study. While downloading mini-programs, we maintained a maximum of six requests per minute, taking over six months to collect them all. In the verification of MKs, we followed the same rate limits, and for WeChat mini-programs, we reported leaked MKs to Tencent, conducted experiments with approved methods, and reported all such cases. Our testing was conducted within controlled parameters, solely involving our accounts, devices, and servers, avoiding attacks on third-party entities. We responsibly disclosed our findings, earning bug bounties from Tencent and Baidu, who confirmed and addressed the vulnerabilities. Certain mini-programs, like those from Toyota, Nestle, and Cisco, have rectified vulnerabilities by eliminating MKs. We noted some vendors mistakenly updated MKs instead of removing them. Additionally, WeChat's introduction of the `auth.checkEncryptedData` API, based on our recommended defense, was acknowledged during our interactions with Tencent, reflecting the significant impact of our study on the vendors.

7 RELATED WORK

Mini-Program Studies. Lu et al. [22] examined the security of the mini-program paradigm by focusing on access control mechanisms. More recently, Zhang et al. [45] explored identity confusion in WebView-based super apps, while Yang et al. [44] discovered a new cross mini-app request forgery attack. Other efforts have been made to understand this novel paradigm. For instance, Zhang et al.[47] developed MiniCrawler to download mini-programs and conducted a large-scale measurement, while Hao et al. [17] studied the system architecture and key technologies used by WeChat mini-programs. Liu et al. [20] created a dynamic analysis framework for WeChat mini-programs and evaluated their tool on 152 open-source mini-programs. Wang et al. [35] introduced TaintMini, a solution for tracking sensitive data flow in mini-programs via a data flow graph. They also uncovered hidden APIs in super apps [37], revealing exploit potential. In APIDiff [36], they identified API execution variations in WeChat across platforms by auto-generating test cases, revealing API discrepancies in presence, permissions, and outcomes. Yang et al. [43] systematically studied security measures, threats, and trade-offs of the super apps. Our work investigates the misuse of cryptographic keys, specifically the MK, by mini-programs and their potential attack consequences. Baskaran et al. [11] also examined the secret leakage. However, their exploration and analysis primarily target cloud services. Their definition of vulnerabilities differs as well; they don't confirm if the leaked app secret is genuinely self-leakage. Instead, they view both self-leakage and the leakage of other mini-apps' app secrets as vulnerabilities.

API Misuse. API misuse is a critical topic in software development, as highlighted in several studies [9, 16, 19, 25, 30, 46]. Such misuse can result in security vulnerabilities, as evidenced by the discovery of API misuse in third-party mobile payment systems that allowed attackers to bypass payment processes [42], and the exploitation of Android fingerprint APIs to launch attacks on vulnerable apps [12]. Researchers have developed various tools and techniques to detect and prevent API misuse. For instance, `SAFEWAPI`[9] uses Web API specifications to identify API misuses in JavaScript web applications, while ARBITRAR[19] leverages active learning to classify valid and invalid usages of target API methods. Ren et al.[25] construct a knowledge graph from API reference documentation to enhance API misuse detection, while MuDetect[30] employs a graph-based representation and ranking strategy to detect API misuses. Additionally, Gorski et al. [16] propose an API-integrated security advice approach to help software developers write more secure code for difficult-to-use cryptographic APIs. ExampleCheck [46] analyzed over 217,818 Stack Overflow posts and discovered that 31% of them potentially contained API usage violations.

Credentials Leakage and Detection. Credentials may be hardcoded in open-source projects, which can lead to security vulnerabilities. Sinha et al. [29] and Singh et al.[28] discuss the issue of API key leaks, in which malicious users steal API keys from public code repositories. Shi et al. [27] studied leaked payment credentials for Cashiers serving over one billion users using `PayKeyMiner`. Wen et al. [40] developed a real-time, large-scale comprehensive secret scanner for GitHub called `SecretHunter`. Michael et al. [23] detected potential credential leakage (such as SSH keys and API tokens) from GitHub repositories with a hybrid approach that involves both automatic detection and manual validation. Basak et al. [10] studied current key management practices by analyzing key leakage in GitHub. Lounici et al. [21] and Saha et al. [26] detect key leakage in GitHub using machine learning. `PassFinder` [15] employs deep neural networks to effectively detect user password leakage from public repositories on GitHub. Compared to previous studies that focused on key leakages in Github, our study is unique in that we focus on mini-programs instead. Also, unlike other studies that aim to improve detection methods, we use simple methods to detect keys and ensure no false positives by validating the correctness of keys using API.

Credentials may also be hardcoded in the front-end of mobile apps, which can lead to security vulnerabilities. Viennot et al. [34]

performed the first study of secret authentication key usage and its problems in Android apps using `PlayDrone`. Zhou et al. [48] developed a tool called `CredMiner`, which can programmatically identify and recover developer credentials from native Android apps. They identified 302 email credentials and 58 Amazon AWS credentials over 36,561 native apps. Zuo et al. [49] conducted a study on cloud service credentials in mobile apps and discussed their root causes and impacts. Wen et al. [41] scanned 100 popular iOS apps and identified that 48 of them had misused credentials, including credential leakage, using `iCredFinder`. Nan et al. [24] developed a semantics-driven solution that utilizes NLP to automatically discover credentials in modern mobile apps. Moreover, Wang et al. [38] recover cloud credentials from apps, infer their capabilities in the cloud, and verify if the capabilities exceed the legitimate needs of the apps. Once again, the focus of other works is primarily on detection methods, such as using machine learning or NLP. In contrast, our focus is on the problem space, where we demonstrate how credentials can lead to various attacks in mini-programs.

In summary, our work stands apart from previous efforts for three main reasons. First, we detect `MK` leaks in the innovative mini-program paradigm, contrasting with prior instances found in Github or mobile app code. Second, our study diverges from others that primarily enhance credential detection methods. Instead, we delve into comprehending mini-program development practices, unique credential-related protocols, and the security implications of leaked credentials. Third, the protocols we scrutinize significantly differ from earlier investigations. Mini-programs incorporate distinct key management, encryption-based resource access, and token-based resource access protocols, absent in mobile apps or web browsers. These protocols, including those from third-party services like Amazon or Facebook, are novel and understanding them aids in grasping security challenges in the novel mini-program paradigm. Furthermore, our attacks yield fresh security consequences. While prior attacks might involve leaked keys for service consumption, akin to our service theft attack, novel mini-programs introduce new features, exploited in attacks like promotion abuse, where vendors harness unique werun data to attract users.

8 CONCLUSION

We have presented the first systematic study of the cryptographic access control in mini-programs, and found that WeChat has made the developers heavily involved in the security related implementation including key management, encryption, and decryption. As such, the developers can mistakenly leak their master keys to the untrusted front-end, leading to various attacks such as account hijacking and promotion abuse. With a large-scale measurement study, we have discovered that 40,880 mini-programs leaked their `MK`s. In addition to the responsible disclosure of this vulnerability and also the list of the vulnerable mini-programs to *Tencent*, we have also discussed the possible countermeasures, particularly on how *Tencent* could have fixed this issue. So far, *Tencent* has provided a new API to mitigate the attacks based on our recommendations.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their insightful feedback. This research was supported in part by NSF award 2330264. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of NSF.

REFERENCES

[1] "Most popular messaging apps," https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/, (Accessed on 08/26/2023).
[2] "Security guide," https://developers.weixin.qq.com/miniprogram/dev/framework/security.html.
[3] "Wechat developer information," https://developers.weixin.qq.com/community/develop/doc/0002ae37438a800cb88a73ef151400, (Accessed on 08/26/2023).
[4] "Baidu Documentation," https://smartprogram.baidu.com/docs/develop/function/login_process/, 03 2020, (Accessed on 08/26/2023).
[5] "Decoded: WeChat Mini Programs For Cultural Destinations," https://jingculturecommerce.com/decoded-wechat-mini-programs-for-cultural-destinations-part-one/, 03 2020, (Accessed on 08/26/2023).
[6] "WeChat API categories," https://developers.weixin.qq.com/miniprogram/en/dev/api/, 03 2020, (Accessed on 08/26/2023).
[7] "Tencent Documentation (auth.checkEncryptedData)," https://developers.weixin.qq.com/miniprogram/dev/api-backend/open-api/user-info/auth.checkEncryptedData.html, 03 2022.
[8] Alipay, "Alipay mini-program documentation," https://opendocs.alipay.com/common/02mse3.
[9] S. Bae, H. Cho, I. Lim, and S. Ryu, "Safewapi: Web api misuse detector for web applications," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 2014, pp. 507–517.
[10] S. K. Basak, L. Neil, B. Reaves, and L. Williams, "What are the practices for secret management in software artifacts?" in *2022 IEEE Secure Development Conference (SecDev)*. IEEE, 2022, pp. 69–76.
[11] S. Baskaran, L. Zhao, M. Mannan, and A. Youssef, "Measuring the leakage and exploitability of authentication secrets in super-apps: The wechat case," in *26nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2023)*, 2023.
[12] A. Bianchi, Y. Fratantonio, A. Machiry, C. Kruegel, G. Vigna, S. P. H. Chung, and W. Lee, "Broken fingers: On the usage of the fingerprint api in android." in *NDSS*, 2018.
[13] Bytedance, "Bytedance mini-program documentation," https://microapp.bytedance.com/docs/zh-CN/mini-app/develop/server/interface-request-credential/get-access-token.
[14] facebook, "Facebook for developers," https://developers.facebook.com/.
[15] R. Feng, Z. Yan, S. Peng, and Y. Zhang, "Automated detection of password leakage from public github repositories," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 175–186.
[16] P. L. Gorski, L. L. Iacono, D. Wermke, C. Stransky, S. Möller, Y. Acar, and S. Fahl, "Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic {API} misuse," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 2018, pp. 265–281.
[17] 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.
[18] T. inc., "Wechat mini-program's official document (data verification and encryption)," https://developers.weixin.qq.com/miniprogram/en/dev/framework/openability/signature.html, 2020.
[19] Z. Li, A. Machiry, B. Chen, M. Naik, K. Wang, and L. Song, "Arbitrar: User-guided api misuse detection," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1400–1415.
[20] Y. Liu, J. Xie, J. Yang, S. Guo, Y. Deng, S. Li, Y. Wu, and Y. Liu, "Industry practice of javascript dynamic analysis on wechat mini-programs," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 1189–1193.
[21] S. Lounici, M. Rosa, C. M. Negri, S. Trabelsi, and M. Önen, "Optimizing leak detection in open-source platforms with machine learning techniques." in *ICISSP*, 2021, pp. 145–159.
[22] 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.
[23] M. Meli, M. R. McNiece, and B. Reaves, "How bad can it git? characterizing secret leakage in public github repositories," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019.
[24] Y. Nan, Z. Yang, X. Wang, Y. Zhang, D. Zhu, and M. Yang, "Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in mobile apps." in *NDSS*, 2018.
[25] X. Ren, X. Ye, Z. Xing, X. Xia, X. Xu, L. Zhu, and J. Sun, "Api-misuse detection driven by fine-grained api-constraint knowledge graph," in *2020 35th IEEE/ACM*

International Conference on Automated Software Engineering (ASE). IEEE, 2020, pp. 461–472.
[26] A. Saha, T. Denning, V. Srikumar, and S. K. Kasera, "Secrets in source code: Reducing false positives using machine learning," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2020, pp. 168–175.
[27] S. Shi, X. Wang, K. Zeng, R. Yang, and W. C. Lau, "An empirical study on mobile payment credential leaks and their exploits," in *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II 17*. Springer, 2021, pp. 79–98.
[28] V. Singh and S. Pandey, "Revisiting cloud security attacks: Credential attack," in *Rising Threats in Expert Applications and Solutions: Proceedings of FICR-TEAS 2020*. Springer, 2021, pp. 339–350.
[29] V. S. Sinha, D. Saha, P. Dhoolia, R. Padhye, and S. Mani, "Detecting and mitigating secret-key leaks in source code repositories," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 396–400.
[30] A. Sven, H. A. Nguyen, S. Nadi, T. N. Nguyen, and M. Mezini, "Investigating next steps in static api-misuse detection," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 265–275.
[31] D. Talk, "Dingding talk mini-program documentation," https://open.dingtalk.com/document/orgapp-server/how-to-call-apis.
[32] Tencent, "The price of ai services," https://developers.weixin.qq.com/miniprogram/dev/platform-capabilities/service-market/intro.html.
[33] ——, "Qq mini-program documentation," https://q.qq.com/wiki/develop/miniprogram/server/open_port/port_use.html.
[34] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, 2014, pp. 221–233.
[35] C. Wang, R. Ko, Y. Zhang, Y. Yang, and Z. Lin, "Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.
[36] C. Wang, Y. Zhang, and Z. Lin, "One size does not fit all: Uncovering and exploiting cross platform discrepant apis in wechat," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.
[37] ——, "Uncovering and exploiting hidden apis in mobile super apps," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.
[38] X. Wang, Y. Sun, S. Nanda, and X. Wang, "Credit karma: Understanding security implications of exposed cloud services through automated capability inference," in *31st {USENIX} Security Symposium ({USENIX} Security 23)*, 2023.
[39] S. Wear, *Burp Suite Cookbook: Practical recipes to help you master web penetration testing with Burp Suite*. Packt Publishing Ltd, 2018.
[40] E. Wen, J. Wang, and J. Dietrich, "Secrethunter: A large-scale secret scanner for public git repositories," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2022, pp. 123–130.
[41] H. Wen, J. Li, Y. Zhang, and D. Gu, "An empirical study of sdk credential misuse in ios apps," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 258–267.
[42] W. Yang, Y. Zhang, J. Li, H. Liu, Q. Wang, Y. Zhang, and D. Gu, "Show me the money! finding flawed implementations of third-party in-app payment in android apps." in *NDSS*, 2017.
[43] Y. Yang, C. Wang, Y. Zhang, and Z. Lin, "Sok: Decoding the super app enigma: The security mechanisms, threats, and trade-offs in os-alike apps," *arXiv preprint arXiv:2306.07495*, 2023.
[44] Y. Yang, Y. Zhang, and Z. Lin, "Cross miniapp request forgery: Root causes, attacks, and vulnerability detection," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3079–3092.
[45] 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 *31st {USENIX} Security Symposium ({USENIX} Security 22)*, 2022.
[46] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 886–896.
[47] Y. Zhang, B. Turkistani, A. Y. Yang, C. Zuo, and Z. Lin, "A measurement study of wechat mini-apps," in *SIGMETRICS '21: ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, Virtual Event*, 2021.
[48] Y. Zhou, L. Wu, Z. Wang, and X. Jiang, "Harvesting developer credentials in android apps," in *Proceedings of the 8th ACM conference on security & privacy in wireless and mobile networks*, 2015, pp. 1–12.
[49] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *40th IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 1296–1310.

A MEASURING THE MK LEAKS IN BAIDU

In a similar manner to our measurement of MK leaks in WeChat, we also measured MK leaks in Baidu. However, as Baidu currently does not have the last update timestamp in its metadata, we conducted a measurement study to answer just the following five research questions:

- **RQ1:** What are the categories of the MK leaked mini-programs?
- **RQ2:** What are their popularity?
- **RQ3:** What are their accessed resources?
- **RQ4:** Who are their developers?
- **RQ5:** Are there any high profile MK leaked mini-programs?

Category	UserInfo (#)	%	PhoneNumber (#)	%
Automobile	81	1.42%	40	1.01%
Business	620	10.84%	398	10.01%
Charity	1	0.02%	1	0.03%
E-commerce	26	0.45%	15	0.38%
Education	261	4.56%	168	4.22%
Efficiency	175	3.06%	92	2.31%
Entertainment	62	1.08%	33	0.83%
Finance	5	0.09%	2	0.05%
Food	44	0.77%	24	0.60%
Government	45	0.79%	29	0.73%
Health	1	0.02%	0	0.00%
Information	289	5.05%	114	2.87%
IT tech	21	0.37%	12	0.30%
Lifestyle	367	6.42%	253	6.36%
Medical	47	0.82%	5	0.13%
News	2	0.03%	2	0.05%
Post service	30	0.52%	23	0.58%
Real estate	340	5.95%	440	11.06%
Shopping	1,793	31.35%	1,222	30.72%
Social	53	0.93%	38	0.96%
Sports	25	0.44%	19	0.48%
Tool	11	0.19%	2	0.05%
Traffic	29	0.51%	22	0.55%
Travelling	636	11.12%	607	15.26%
Uncategorized	755	13.20%	417	10.48%
Total	5,724	100%	3,996	100%

Table 11: Statistics of the identified vulnerable Baidu mini-programs w.r.t their categories. The percentage represents the proportion of vulnerable mini-programs accessing a particular resource, compared to the total number of mini-programs accessing that resource.

In total, we found that 7,476 of the 171,989 Baidu mini-programs (4.35%) we collected have leaked their MKs. In the following, we provide the answers for these five RQs with respect to these mini-programs.

(RQ1) Categories of the MK Leaked Mini-programs. Similar to WeChat, most Baidu mini-programs have been classified into specific categories, as shown in Table 11. Since Baidu does not permit individual developers to release their mini-programs, all of them are developed by different companies. It is noticeable that shopping, business, and traveling mini-programs have a slightly higher number of vulnerable mini-programs due to the larger number of mini-programs in those categories. As these mini-programs typically involve payments, attacking them through their leaked MK may have broad security implications.

Download Times	UserInfo (#)	%	PhoneNumber (#)	%
(0, 1,000]	1,479	25.86%	1,002	25.19%
(1,000, 2,000]	952	16.65%	647	16.26%
(2,000, 3,000]	785	13.73%	522	13.12%
(3,000, 4,000]	430	7.52%	319	8.02%
(4,000, 5,000]	545	9.53%	408	10.26%
(5,000, 10,000]	867	15.16%	634	15.94%
(10,000, 100,000]	644	11.26%	436	10.96%
(100,000, 1M]	16	0.28%	9	0.23%
> 1M	1	0.02%	1	0.03%
Total	5,719	100%	3,578	100%

Table 12: Statistics for vulnerable Baidu mini-programs based on downloads. Note that some lack download times, leading to a lower total count of mini-programs with user info and phone access than in Table 11.

Service Name	Description	Enabled by default
Login	Services for users to Login	✓
Book Shelf	Online bookshelf services for e-learning apps	✗
Coupon	Services for developers manage coupons	✓
Resource Management	Services for developers manage online resources	✓
Template Messages	Services for manage template messages	✓
Customer Messages	Services for manage customer messages	✓
Risk Detection	Services for detecting malicious users	✓
QR Code	Services for creating QR code	✓
Comment Management	Services for managing comments	✓
Content Security	Services for detecting illegal content	✗

Table 13: Summary of the Baidu-managed cloud services.

Number of Apps	Downloaded <= 5000	Downloaded > 5000	Total
1	3981	1422	5403
2	183	126	309
3	55	35	90
4	27	19	46
5	23	21	44
6	20	7	27
7	9	7	16
8	10	7	17
9	6	5	11
10	5	4	9
> 10	20	10	30

Table 14: Numbers of vulnerable mini-programs associated with the same software development companies. Note that not all apps have their developer information available.

(RQ2) Popularity of the MK Leaked Mini-programs. As illustrated in Table 12, the statistics of identified vulnerable Baidu mini-programs with respect to their download times, which signify their popularity, are reported. It is evident that the majority of vulnerable Baidu mini-programs exhibit relatively low download times, with over 50% of them residing within the (0, 1,000] and (1,000, 2,000] ranges. This finding implies that less popular mini-programs are more susceptible to security vulnerabilities. Nevertheless, it is crucial to emphasize that even in the (100,000, 1M] and >1M ranges, instances still exist. This observation serves as a reminder that security issues can persist, regardless of the high download rates of certain mini-programs.

(RQ3) Accessed Resources of the MK Leaked Mini-programs. Both Table 11 and Table 12 display the accessed resources of vulnerable Baidu mini-programs. It can be observed that they access more `UserInfo` than `PhoneNumber`. The discrepancy could suggest that developers exercise greater caution when handling sensitive data, such as phone numbers. Nevertheless, a significant number of mini-programs still access this information. Meanwhile, we would like to note that since Baidu lacks APIs for accessing `ShareInfo` and `WeRunData`, we only provide statistics on the number of mini-programs that utilize `UserInfo` and `PhoneNumber`. In contrast to WeChat, Baidu does not offer APIs such as `api.weixin.qq.com/cgi-bin/get_current_selfmenu_info` to verify the enabled services of specific mini-programs. The sole method to determine the availability of services is to invoke them, which can raise ethical concerns. Consequently, we cannot showcase the enabled services of these mini-programs.

However, as demonstrated in Table 13, we notice that numerous services are enabled by default, indicating that attackers can launch service theft with success. It is evident that some of these services are employed to manage essential online resources (e.g., images, videos) within the mini-programs. An attacker could potentially delete these resources by exploiting service theft attacks. Additionally, certain APIs can send messages (e.g., template message services and customer message services) directly to mini-programs running on users' devices, potentially tricking users into believing and clicking on phishing websites (i.e., the attacker may masquerade as an official party to send phishing messages). Interestingly, many of these APIs (e.g., resource management services) have a rate limit, meaning that if an attacker actively utilizes these services, developers may lose the opportunity to invoke them.

(RQ4) The Developers of the MK Leaked Mini-programs. As shown in Table 14, the data is categorized into the number of vulnerable mini-programs and the number of companies, which are separated into two groups: those with mini-programs downloaded less than or equal to 5,000 times, and those with mini-programs downloaded more than 5,000 times. Fewer companies have multiple vulnerable mini-programs. For instance, there are 309 companies with 2 vulnerable mini-programs, 110 companies with 3 vulnerable mini-programs. This trend could suggest that having multiple vulnerable mini-programs is relatively rare. There are also 30 companies with more than 10 vulnerable mini-programs. For example, a company named "ShangFang (ShenZhen) Tech Inc." developed 87 vulnerable mini-programs.

(RQ5) High Profile MK Leaked Mini-programs. We also discovered that 21 mini-programs were developed by Fortune 500 companies, including many prominent names such as Sony, HP, Amazon, and Toyota. These findings indicate that even large, well-established companies can make mistakes when it comes to developing secure mini-programs. Among the 21 mini-programs, 16 of them accessed the user's phone number or personal information. In addition to these well-known companies, Baidu, the provider, developed 39 vulnerable mini-programs. Of these, 33 accessed the user's phone number or personal information. This highlights the importance of addressing security concerns across organizations of all sizes and reputations in order to protect user privacy and maintain trust.