Measuring the Leakage and Exploitability of Authentication Secrets in Super-apps: The WeChat Case

Supraja Baskaran
Concordia University
Montreal, Canada
su_baska@ciise.concordia.ca

Lianying Zhao
Carlton University
Ottawa, Canada
lianying.zhao@carleton.ca

Mohammad Mannan
Concordia University
Montreal, Canada
mmannan@ciise.concordia.ca

Amr Youssef
Concordia University
Montreal, Canada
youssef@ciise.concordia.ca

ABSTRACT

Super-apps such as WeChat and Baidu host millions of mini-apps, which are very popular among users and developers because of the mini-apps' convenience, lightweight, ease of sharing, and not requiring explicit installation. Such ecosystems involve several entities, such as the super-app and mini-app clients, the super-app backend server, the mini-app developer server, and other hosting platforms and services used by the mini-app developer. To support various user-level functionalities, these components must authenticate each other, which differs from regular user authentication to the super-app platform. In this paper, we explore the mini-app to super-app authentication problem caused by insecure development practices. This type of authentication allows the mini-app code to access super-app services on the developer's behalf.

We conduct a large-scale measurement of developers' insecure practices leading to mini-app to super-app authentication bypass, among which hard-coding developer secrets for such authentication is a major contributor. We also analyze the exploitability and security consequences of developer secret leakage in mini-apps by examining individual super-app server-side APIs. We develop an analysis framework for measuring such secret leakage, and primarily analyze 110,993 WeChat mini-apps, and 10,000 Baidu mini-apps (two of the most prominent super-app platforms), along with a few more datasets to test the evolution of developer practices and platform security enforcement over time. We found a large number of WeChat mini-apps (36,425, 32.8%) and a few Baidu mini-apps (112) leak their developer secrets, which can cause severe security and privacy problems for the users and developers of mini-apps. A network attacker who does not even have an account on the super-app platform, can effectively take down a mini-app, send malicious and phishing links to users, and access sensitive information of the mini-app developer and its users. We responsibly disclosed our findings and also put forward potential directions that could be considered to alleviate/eliminate the root causes of developers hard-coding the app secrets in the mini-app's front-end code.

RAID '23, October 16–18, 2023, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-0765-0/23/10…$15.00
https://doi.org/10.1145/3607199.3607236

CCS CONCEPTS

• **Security and privacy → Software and application security**; **Authentication**.

KEYWORDS

Authentication, Mini-app Security, WeChat, Hard-coded Secrets

ACM Reference Format:
Supraja Baskaran, Lianying Zhao, Mohammad Mannan, and Amr Youssef. 2023. Measuring the Leakage and Exploitability of Authentication Secrets in Super-apps: The WeChat Case. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3607199.3607236

1 INTRODUCTION

Full-featured apps such as WeChat [62] and Baidu [64], with a monthly user base of over one billion [5, 37], have created an ecosystem to accommodate payments, media, online stores, developers, etc [22, 25, 72]. Such popularity and self-contained ecosystem have enabled them to become "super-apps", serving as the hosting platform of millions of mini-apps (also known as mini-programs). Mini-apps, together with their super-apps, are seeing increasing demands in different countries because of their convenience, light weight, ease of sharing and no need to install [70]. In addition, mini-apps do not need a custom backend server from developers, but use a well-constructed set of APIs provided by their super-app to allow straightforward access to backend data and system resources that are provided by the super-app platforms.

Unsurprisingly, security and privacy issues also start to surface in these super-app platforms, e.g., users' PII (name, national ID, date of birth, and facial data) collection by WeChat mini-apps [42]. More extensive security-focused work in this domain includes (details in Sec. 9): analysis of mini-app permission models [68], identity confusion attacks [69], and cross mini-app request forgery attacks [67].

In contrast to existing work, we focus on the mini-app's authentication secret (developer secret or app secret) leakage, i.e., the secrets used to authenticate any part of the mini-app code to the super-app server that it is the mini-app it claims to be. This is complicated by several factors, including: lack of a human user presenting secrets at

the time of authentication (hence the need for somewhere to store the mini-app to super-app authentication secret), mini-app developers not following security guidelines from super-app platforms, failure of super-app platforms to enforce necessary security restrictions, multiple entities being involved in a super-app ecosystem (e.g., mini-app client, cloud functions, server-side APIs, and data storage provided by a super-app server, mini-app developer server), which also authenticate each other–either explicitly or implicitly.

A common way for the super-app to authenticate a mini-app is to use app secrets. According to the WeChat documentation [58], an app secret is specific to a developer account, and it is used to authenticate any part of the mini-app (representing the developer) to the super-app's server. Therefore, the app secret should be treated as a sensitive piece of data and should not be exposed (e.g., in the mini-app source code). Due to the absence of a human user in such authentication scenarios, app secrets have to be stored instead of being entered by a user. WeChat expects mini-apps to offload such a burden to the developer's server which stores app secrets and performs necessary communication with the super-app server, i.e., not to hard-code in the mini-app code. For example, in the case of WeChat mini-apps, the mini-app packages can be extracted from a rooted/jail-broken device, or using the PC client, with common reverse-engineering techniques without any special privilege [15] and the app secrets, if hard-coded, can be easily obtained by anyone. If the developers still use such hard-coding, since the super-app server has complete control over the super-app ecosystem, it can easily spot such hard-coding and prevent it either before releasing a mini-app or at run-time. Our work is motivated by the observation that it is not the case in reality.

When such mini-app to super-app authentication is compromised due to hard-coded app secrets, one can intuitively imagine how things may go wrong afterwards, e.g., the attacker will be able to impersonate the legitimate mini-app (developer) and manipulate/abuse its resources (e.g., images or order info). To better understand this problem, we examine how app secrets are used to achieve the authentication: it is the super-app server that authenticates a remote party claiming to be the mini-app and provides subsequent services all through a set of exposed APIs over the network [59]. Therefore, these APIs eventually become the target of the authentication compromise, i.e., whether they can be invoked in an unauthorized manner. The super-app platforms often have certain security guidelines to ensure proper mini-app to super-app authentication. For instance, WeChat discourages having app secrets in the mini-app, recommends the use of *IP address whitelisting* for such APIs (not allowing calls from non-listed addresses, configured at the developer portal), and disallows directly calling such APIs from within the mini-app (as opposed to doing it from the developer's server) [58]. Still, enforcement of these guidelines remains a question, leading to insecure development practices.

The mini-app to super-app authentication mechanisms vary across super-app platforms, and thus the ways of managing authentication secrets vary. For example, Douyin [12] (the Chinese version of Tiktok [13]), uses a similar authentication technique like WeChat and Baidu. The mini-app to super-app authentication system, which relies on `"client_credential"` grant type to generate access tokens, is significantly affected by the mini-app developers' mishandling of the secret. In contrast, Alipay employs a dynamic authorization token for the generation of access token which involves no static secrets (`"authorization_code"` grant type), thus providing a better authentication with the super-app. Our analysis targets major popular super-app ecosystems that have such authentication secret leakage problems, although primarily focuses on WeChat as it is the largest among these platforms in terms of mini-app count [37].

Our objective is to measure the extent to which the aforementioned insecure mini-app development practices are found along the timeline of recent years (assuming the awareness is improving), and how such practices could have led to potential unauthorized calls to the super-app server APIs. We also explore the security consequences of such calls in the super-app ecosystem, e.g., business/personal resources in various application scenarios. To do so, we develop an analysis framework that automatically detects app secrets in the mini-app's code through static analysis and verifies if authentication bypass is possible to call unauthorized super-app server APIs, confirming the validity of the identified secrets and the lack of IP whitelisting. For ethical reasons, we do not make calls to all the super-app server-side APIs, but instead, divide them into Get (can only read data) and Modify (can update/delete data) APIs. The framework automatically makes calls to only the necessary Get APIs, which provides the required parameters to perform a "callability" analysis of Modify APIs. Also, we analyze and categorize the security consequences of the unauthorized super-app server-side API access and shed light on the ways forward for improvement.

Summary of contributions and notable findings.

(1) We examine the mishandling of authentication secrets in WeChat and Baidu mini-apps, two leading super-app providers. The identified issue eventually leads to unauthorized super-app server-side API calls by any network attacker, allowing access to various mini-app-owned resources. Our methodology is designed to facilitate *automated* analysis of mini-apps from super-app platforms that have a similar mini-app directory structure (like WeChat [56] and Baidu [10]), and use secret-based mini-app authentication.

(2) We conduct a large-scale automated measurement to assess the extent of these insecure practices in WeChat and Baidu. Out of 110,993 WeChat mini-apps that we could successfully decrypt and unpack (from a total of 115,392, crawled in 2021) and 10,000 Baidu mini-apps, we found a large number of WeChat mini-apps (36,425, 32.8%) and a few Baidu mini-apps (112) leak their developer secrets, which can cause authentication bypass. The use of IP whitelisting, a WeChat security feature, which could restrict exploitation of such secret exposures, is also very limited–only 33 out of 110,993 mini-apps have enabled it (7 mini-apps with app secret). We also automatically check data leakage through the available Get APIs, and callability of Modify APIs that can directly interfere with a mini-app functionality. We test the effects of dangerous Modify APIs only on our own mini-app.

(3) From our responsible disclosure, we learned that WeChat is aware of app secret hard-coding (independent of our reporting), and has enforced a new requirement that publishing such mini-apps are disallowed (which we also confirmed by submitting a new mini-app with hard-coded app secret–it was rejected). To check the effectiveness of this new requirement, and to see how developer behaviours have evolved over the years, we performed

a few additional measurements. From 9,824 mini-apps crawled in Feb. 2023, 2,572 (26.2%) have valid app secrets, and from the 36,425 mini-apps (crawled in 2021, with valid app secrets), 36,293 (99.6%) still have valid app secrets—making recent enforcement by WeChat largely ineffective against existing vulnerable mini-apps (which were made public before Mar. 2023).

(4) We conduct an in-depth attack feasibility analysis for individual APIs and mini-apps, and categorize the security consequences of such attacks. The consequences vary a lot with the semantics of the involved APIs, the configuration and functions of individual mini-apps, and several types of consequences are high-impact and affect a large number of mini-apps.

(5) We discuss the root causes of the hard-coding app secrets, and suggest several recommendations for design and enforcement considerations. We will make our tool available to any super-app platforms, who can identify and measure security consequences for different apps and take appropriate mitigating actions.

2 BACKGROUND

We first briefly explain the various terms/entities involved in mini-app ecosystems, specifically in WeChat and Baidu.

Super-app. A *super-app client* is the host mobile app that features a selection of independent services, all contained within a single app. The *super-app server* is the key authority of the super-app platform, managing identity and operations of the mini-apps, and providing necessary services. The super-app server exposes a standard set of APIs to all mini-apps, which we refer to as *server-side APIs* hereafter.

Mini-app. The *mini-app client* is the client-side code that runs on top of the super-app client (also called mini-program, smart-program, micro-app). It is created with the corresponding super-app's devtools and shipped as a package (JavaScript, XML, JSON, and CSS). Every mini-app has an *app ID*, which is a unique identifier (in a given super-app) with random alphanumeric characters, often used for the mini-app's requests to the super-app server. Every mini-app developer has a random secret (in WeChat and Baidu, called *app secret*, 32-character long), assigned to their account, which is used to authenticate the developer of mini-app for calling the super-app server-side APIs. The app secret is regarded as a sensitive piece of information. The *developer server* is the backend server of a specific mini-app, set up and maintained by the mini-app developer (not controlled by the super-app platform). In the case of WeChat, the developer server must have a valid Internet Content Provider (ICP [65]) licensed domain name. Without specifically mentioning the server or client/package, hereafter by *mini-app* we refer to the entire mini-app, any code representing the corresponding developer or business.

Access token. To use a super-app's server-side APIs, and access resources associated with a mini-app, an *access token* is required. This is an ephemeral secret (valid for 2 hours for WeChat and 30 days for Baidu, renewable anytime), issued by the super-app server through an API call, e.g., `getAccessToken` [54] in WeChat, which requires an app ID and app secret as request parameters.

OpenID. In WeChat and Baidu mini-apps, the *OpenID* is a user identifier that is unique for each mini-app. When a user logs into a mini-app, the mini-app sends a request to the super-app server to obtain the user's openID, which is based on the user's super-app account. In WeChat, it is an encrypted value of the user's WeChat ID and the app ID of the mini-app, and remains the same for the same user-mini-app combination.

WeChat cloud base functionalities. Mini-apps can take advantage of the cloud base [50], an option in WeChat that enables the mini-apps to utilize some basic cloud functionalities without setting up a dedicated server. The cloud base has a range of features, e.g., cloud functions, databases, storage and cloud call. A *cloud function* allows developers to execute their server-side (JavaScript) code. These functions can be typically triggered by specific events, such as a user action or any change in data. Within the mini-app, cloud functions can be triggered with the mini-app's regular API (termed as JSAPI) `wx.cloud.callFunction`. The cloud base also offers the *cloud call* capability to call the server-side APIs from cloud functions. This is recommended by WeChat apart from calling the server-side APIs using developer server or Tencent cloud hosting [61]. Cloud calls are implicitly authenticated (no need to supply the app secret or access token). The JSON *cloud base database* can be queried by the mini-app to retrieve or update data using cloud functions. This database can be called using either JSAPIs, or the server-side APIs from the developer server (with an access token). The *cloud base storage* provides a storage space for mini-apps to store their files, accessible by dedicated APIs, cloud functions, or the developer server.

WeChat IP whitelisting. The calls to server-side APIs can be restricted to originate from only a list of IP addresses (no domain names), configured at the WeChat mini-app developer portal. If enabled by the developer (disabled by default), only these IP addresses can call the server-side APIs, i.e., no other hosts can obtain access tokens even if they have valid app secrets. Note that IP whitelisting applies to all the server-side APIs (i.e., not for specific APIs).

WeChat plug-in. A WeChat mini-app plug-in is a package of pre-built custom components or libraries that can be integrated into a mini-app. Developers can request to integrate a plug-in (developed by WeChat and third-parties) from the developer portal.

3 OBJECTIVES AND THREAT MODEL

In this section, we formulate the secret leakage problem we consider, state our objectives, and define the threat model.

Authentication Secret leakage problem. Mini-apps are quite different from regular mobile apps, in terms of their reliance on the super-app platform as a runtime authority for authentication, access control and other services through the server-side APIs. There are mainly three types of authentication involved in the super-app environment: *mini-app user to super-app*, when the user initially logs into the super-app (usually persistent across reboots as in WeChat); *mini-app user to mini-app*, each time (or some cases the first time) the user uses the mini-app and clicks to consent to identity sharing; and *mini-app to super-app*, which is often neglected as it happens behind the scene. Our work deals with this last case (as the other two involve the human mini-app user which is an orthogonal authentication problem). There exists also *mini-app to mini-app* authentication [67] via the super-app, which can also be considered as mini-app to super-app.

As explained in Sec. 2, a mini-app uses its app secret (and the app ID) to obtain an access token from the super-app server which

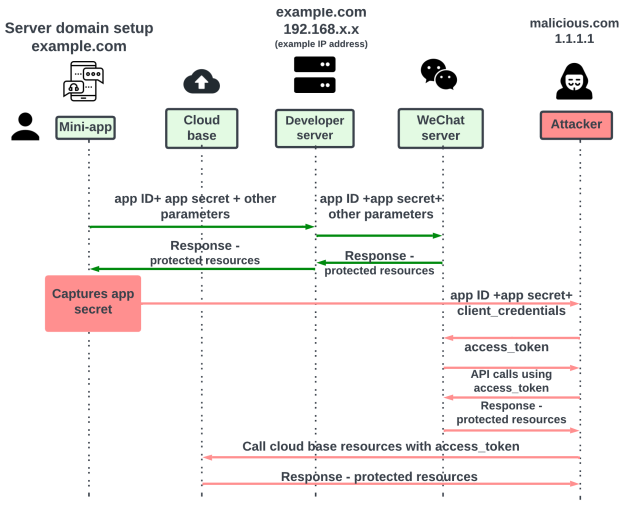

Figure 1: Example scenario of an attacker making use of the hard-coded app secrets, by reverse-engineering the mini-app binary, and using them to successfully access WeChat's server-side APIs.

can be used for all subsequent server-side API calls. This means any improper practices enabling unauthorized calls to the server-side APIs will compromise the mini-app to super-app authentication, leading to various security issues or outright attacks.

The mini-app authentication secret leakage problems we consider are usually reflected in the following aspects (1) insecure but common practices of the mini-app developers; and (2) inherent design flaws and failure to enforce their own security guidelines by super-app platforms. Taking WeChat as an example, 1) the app secret should not be included directly within the mini-app package; 2) any API involving the app secret as its request parameter should be called by the developer server only, not from within the mini-app; 3) IP whitelisting should be configured. Mini-apps developed by not following one or multiple of such recommendations can be insecure, leading to unauthorized server-side API calls; see Figure 1.

Objectives. We center our study around the issues of mini-app secret leakage and its exploitation as presented above, i.e., unauthorized calls to the server-side APIs, caused by insecure development practices. We aim to 1) find out the extent to which the insecure development practices are identified from a large number of mini-apps, despite the warnings in documentations over time; 2) analyze the feasibility of the attacker being able to actually make unauthorized calls to individual server-side APIs, for mini-apps with the identified insecure practices; 3) understand the security consequences of such unauthorized server-side API calls, in a given super-app platform.

Threat model. We assume that the mini-apps are benign. The (source) code of the mini-app is integrity-protected by the super-app. Also, the developer servers as well as the super-app server used by the mini-apps are trusted and the communication between the mini-app (via the super-app) and its developer server is through HTTPS, and is thus assumed to be secure. These assumptions are in line with the day-to-day uses of mini-apps and what has been assumed by the mini-app service providers.

Attacker requirements and capabilities. An attacker with a regular super-app account can obtain (sometimes on a large-scale) the binary package of any mini-app that is publicly visible. This can be achieved by several means, e.g., installing the super-app on a rooted device or its PC client with the help of some reverse-engineering tools (e.g., Frida [49]). Then, certain open-source scripts (e.g., [20, 21] for WeChat) can be used to extract/unpack the content (JS and resource files). The attacker can read and change the code of the reverse engineered mini-app locally, but will not be able to re-publish it for the same app ID (enforced by the super-app). The attacker can view the code of the mini-app, but will not have access to the mini-app's cloud base. They can thus read the mini-app's code for hard-coded secrets, and other information such as business logic, and use the obtained information to attack the mini-app in different ways. To access the developer server-side APIs and super-app server-side APIs, the attacker does not need to possess any special privileges, or even a regular super-app account (e.g., WeChat or Baidu). They just need access to an OS terminal capable of dealing with web requests or a REST client such as Postman [36], if the IP whitelisting for the mini-app is disabled.

Scope. Our study is primarily focused on WeChat mini-apps, with certain analysis extended to Baidu mini-apps. WeChat's other open platform features such as official accounts, WeChat's SSO, mobile and web development are out-of-scope. Other types of authentication which do not involve the explicit usage of app secrets, e.g., mini-app user to super-app (e.g., the attacker being able to log into someone's WeChat account), mini-app user to mini-app, and mini-app to mini-app (e.g., one mini-app impersonating another, see [67]), are excluded. We only examine the (reverse-engineered) mini-app packages, and the response to server-side API calls from the super-app/developer server. We do not perform traffic analysis and do not consider JSAPIs, and other APIs, e.g., Tencent's cloud hosting APIs, third-party APIs, WeChat's payment APIs, and Baidu's cloud APIs.

4 METHODOLOGY

In this section, we discuss our methodology for conducting a stepwise analysis of WeChat mini-apps for potential unauthorized server-side API calls. The same steps apply to Baidu mini-apps as well, except for decrypting and unpacking, and IP whitelisting validation. Our methodology is designed to facilitate the analysis of mini-apps from other platforms that have a similar mini-app directory structure (like WeChat and Baidu) and rely directly on secret-based mini-app authentication. A large-scale measurement study on WeChat and Baidu using this methodology is presented in Sec. 5. We build our tool following this methodology, which is fully automated, except for separating the Get vs. Modify APIs from the super-app documentations; we manually label these APIs to avoid calling the Modify APIs, which may interfere with the mini-apps' functionality.

Overview. The analysis is composed of the following steps: preparation of mini-app files, detection of insecure development practices, reviewing the list of server-side APIs, and testing of potential unauthorized server-side API calls; see Figure 2. We consider the

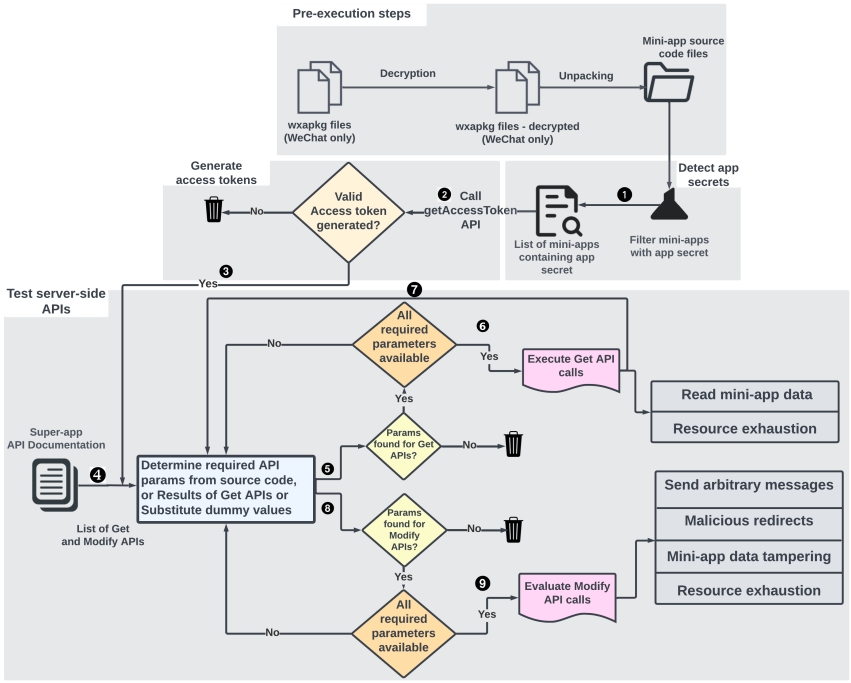

Figure 2: Overview of our analysis methodology for WeChat and Baidu mini-apps.

following insecure development practices: hard-coded app secrets in the mini-app package, absence of IP whitelisting, and direct invocation of server-side APIs. First, we decrypt and unpack the WeChat mini-apps (using [21] and [20], respectively). This step is not needed for Baidu mini-apps. Then, with static code analysis of the unpacked mini-apps, we detect hard-coded app secrets and direct invocation of server-side APIs. Next, we use the identified app secrets to obtain the access tokens from the super-app server in preparation for next steps. Meanwhile, we review (once per super-app) the list of server-side APIs, categorize them and analyze the requirements for invoking them. When it comes to testing potential unauthorized server-side API calls, we make use of the access tokens from the previous step to evaluate the callability of these APIs by an attacker, which at the same time also checks if IP whitelisting is enabled or not.

Detection of hard-coded secrets. We first manually analyzed 100 WeChat mini-apps, and observed that although such app secrets are eventually used as request parameters to standard API calls (e.g., code2session API, see Appendix A.1 and Figure 4), they are also used to make calls to custom APIs of a mini-app developer server (see Figure 5 in Appendix), or even a few obfuscated functions, which will invoke the standard APIs on the server side. To find and capture the app secret patterns, we designed regular expressions for WeChat and Baidu to identify such secrets in the unpacked mini-app code (following similar approaches based on pattern, keywords, and entropy [29, 30, 41]). Some parts of the source code of the reverse-engineered mini-app are highly obfuscated, making it difficult to analyze. However, the variable names used for identifying the app ID and app secret are not obfuscated as observed in our initial manual analysis. Any JSON data, including API request parameters, will not be obfuscated as the server interprets the received data depending on the variable names. Similarly, static string matching is also used against a set of 84 server-side APIs in total to detect calling of these APIs directly from the mini-app. Note that we do not distinguish between the case where a mini-app discloses its own secret and the case where it discloses the secret of another mini-app (which may happen due to code cloning). All our identified app secrets are validated by calling the `getAccessToken` API [54] (see the discussion below about obtaining the access token), ensuring that all the identified secrets are valid and there are no false positives.

Preparing the list of server-side APIs. We study WeChat and Baidu mini-app's server-side API documentation [8, 59, 60] and divide these APIs into two categories: *Get APIs* that only read information, and *Modify APIs* that can cause updates (Table 5 and Table 6 in Appendix). The category (Get/Modify) is determined based on the API's intended operation and its request parameters. To further determine their callability without actually invoking them, we manually analyze the request parameters of each API and confirm if all the required request parameters can be obtained. Only when an API is confirmed to be callable, we include it in our analysis. In our measurement study, finding these request parameters and verifying the callability of Modify APIs are carried out using our automated framework. It can be inferred from Table 5 that most of the Modify APIs can be called with the help of the response returned from the corresponding Get API, which refers to the special subset we need to call. To evaluate this predetermined list of server-side APIs in both WeChat and Baidu, we use the classification specified by

the respective super-apps. In most cases, all APIs that belong to one category are used for the same mini-app feature. Thus, we can easily assess a Modify API using the response of a Get API from that same category, and identify features susceptible to attacks.

Obtaining the access token and the presence of IP whitelisting. To prepare for testing the server-side APIs, as well as to verify the validity of the detected hard-coded secret from a given mini-app, we need to call the access token generation API. If the API returns a valid access token, it can be inferred that the mini-app is still actively present in the respective super-app platform. We use the app ID and app secret parameters from the static analysis along with the authorization grant type parameter set to a constant value `"client_credential"`. The corresponding super-app server verifies the app ID and app secret and returns a valid access token. If the app ID or app secret is incorrect, the API will return an error message which confirms the invalidity of the credentials. Note that an app secret confirmed to be invalid presently does not mean it was invalid at the time of being hard-coded in the mini-app package.

Being able to obtain an access token in the previous step can also confirm the absence of IP whitelisting for a mini-app (applicable only for WeChat mini-apps). If the app secret is invalid, the response from the server indicates that the app ID or app secret is invalid; if IP whitelisting is enabled, the response mentions that the IP address of the request origin is not in the whitelist. We use such explicit error messages to check the adoption of whitelisting among all the WeChat mini-apps, even when we do not have the corresponding app secret (in which case, we use a random 32-character value).

Testing the server-side APIs. Using the valid access tokens obtained from the previous step, we attempt to call a subset of the Get server-side APIs that do not involve any form of modification of the mini-app data or operations (listed in Tables 5 and 6). Calling this subset is necessary to enable analysis for the Modify APIs, and we only retrieve metadata to check if the required request parameters for calling other APIs are available without storing any data returned in the response. Note that this does not require possessing a WeChat or Baidu account. All the available server-side API requests have a calling quota and hence, we avoid making arbitrary API calls. If a Get API is successful for a particular category, then, we determine if the Modify API from the same category can be successfully called or not. If a Get API call returns a success response, our framework can automatically determine the callability of a corresponding Modify API. This is done by making use of a combination of: the response returned by the Get API, inserting attacker-controlled dummy inputs (based on the API documentation), searching the mini-app source code for pertinent values. We confirm the callability of a Modify API if all the required parameters are available. Whenever we need any verification of the behaviour of Modify APIs, or resource-exhaustive calls to the Get APIs, we test them only with our own mini-app.

5 MEASUREMENT

To achieve the three objectives put forward in Sec. 3, we quantify what can be observed from a large number of mini-apps, and for individual APIs in their respective contexts. Also, we consider the historical aspect of such insecure development practices since the extent (prevalence) we measure can also be temporal, e.g., as the warnings or recommendations in the documentation [58] can be dated back to the early days of mini-apps, what has been the prevalence over time?

5.1 Datasets

An early collection of WeChat mini-apps we were able to obtain is a dataset of 115,392 WeChat mini-apps (`"wxapkg"` files), crawled between July 2021 and Dec. 2021 (`DATASET1`). We also used a dataset of 10,000 Baidu mini-apps crawled in Jan. 2023 (`DATASET2`), 10,000 additional WeChat mini-apps, collected on Jan. 25, 2023 (`DATASET3`). These datasets were crawled using MiniCrawler [70]. Note that this choice of datasets can naturally cover an important case where older versions of a mini-app hard-coded the app secret which got leaked but the app secret still remains valid now, enabling attacks. Then, it is possible that by examining the current version of the same mini-app package, no hard-coded app secret can be detected, hence not drawing attention and misleading both the developers and WeChat into a false sense of security. For this same-mini-app temporal comparison, we also randomly picked 100 mini-apps from `DATASET1`, and analyzed them twice – in Jan. 2023 and in Mar. 2023, before and after our report to WeChat respectively (more in Sec. 5.3). We analyzed `DATASET1` in Dec. 2022 and `DATASET2` in Feb. 2023.

After decrypting the WeChat mini-apps files from `DATASET1`, we ended up with 115,244 successfully decrypted mini-apps, consuming 265 GB disk space. From these mini-apps, we could successfully unpack 110,993 in total (434 GB in size), which we used for further analysis. For Baidu mini-apps in `DATASET2`, we extracted the zip files and used them as is (59 GB in size, no decryption or unpacking needed). Among the recent 10,000 WeChat mini-apps (`DATASET3`), we managed to decrypt and unpack 9,994 mini-apps and 9,824 mini-app respectively. For an overview of our datasets and the corresponding app secrets and generated access tokens, see Table 1.

Mini-apps	**DATASET1**	**DATASET3**	**DATASET2**
# total	115,392	10,000	10,000
# decrypted	115,244	9,994	-
# unpacked	110,993	9,824	-
# hard-coded secrets	43,377	2,894	112
# access tokens	36,425	2,572	112

Table 1: WeChat (DATASET 1 and DATASET 3) and Baidu (DATASET 2) mini-app datasets used in our measurement. "# hard-coded secrets": the number of unique mini-apps with hard-coded app secrets; "# access tokens": mini-apps where access tokens were generated successfully.

5.2 Measurement Results

5.2.1 *Insecure Development Practices.*

Hard-coded app secret and IP whitelisting. Our regular expressions to match the app secrets resulted in hits in 43,337 out of the 110,993 (successfully decrypted and unpacked from `DATASET1`) WeChat mini-apps (about 39%), and 112 out of the 10,000 Baidu mini-apps (about 1%). When we attempted to generate access tokens from these app secrets, we found a total of 36,425 (approximately 33%) WeChat mini-apps with valid hard-coded secrets (i.e., successfully generated access tokens), violating the security guidelines by

Server-side APIs	Required parameters	# miniapps	[A]	[B]	[C]	[D]	[E]	Impact
clearQuotaByAppSecret	appID, appSecret	36,425			✓		✓	High
clearQuota	AT, appID	36,425			✓		✓	High
managePlugin	AT, pluginAppID	7,242	✓		✓		✓	High
deleteNearbyPoi	AT, poiID	2,927			✓		✓	High
setShowStatus	AT, poiID	2,927			✓		✓	High
managePluginApplication	AT, appID	772	✓		✓		✓	High
invokeCloudFunctions	AT, cloudFunctionName	202	✓	✓	✓		✓	High
databaseCollectionGet	AT, cloudEnv	24	✓				✓	High
databaseCollectionAdd	AT, cloudEnv, CollectionName	24			✓		✓	High
databaseCollectionDelete	AT, cloudEnv, CollectionName	24			✓		✓	High
databaseAdd	AT, cloudEnv	24			✓		✓	High
databaseDelete	AT, cloudEnv	24			✓		✓	High
databaseUpdate	AT, cloudEnv	24			✓		✓	High
databaseQuery	AT, cloudEnv	24	✓		✓		✓	High
setUpdatableMsg	AT	17		✓	✓	✓	✓	High
uploadTempMedia	AT	36,425			✓		✓	Medium
getApiQuota	AT, cgi_path	36,425	✓				✓	Medium
getDomainInfo	AT	33,795	✓				✓	Medium
getFeedback	AT	18,475	✓				✓	Medium
customerServiceMessage.send	AT, openID	18,224		✓		✓	✓	Medium
getQcloudToken	AT	11,786	✓				✓	Medium
getAllDelivery	AT	8,622	✓				✓	Medium
getPrinter	AT	312	✓				✓	Medium
updatePrinter	AT, openID	312			✓		✓	Medium
createActivityId	AT	36,425	✓				✓	Low
getNearbyPoiList	AT	2,927	✓				✓	Low
addTemplate*	AT	112			✓		✓	Medium
submitResource*	AT	112			✓		✓	Medium
submitSitemap*	AT	112			✓		✓	Medium
interfaceSubmission*	AT	112			✓		✓	Medium
submitsku*	AT	112			✓		✓	Medium
createCoupon*	AT	112			✓		✓	Medium
submitcoupon*	AT	112			✓		✓	Medium
ManageCoupon*	AT	112			✓		✓	Medium
getTemplateList*	AT	74	✓				✓	Medium
deleteMessageTemplate*	AT	74			✓		✓	Medium

Table 2: Statistics of unauthorized callable WeChat (`DATASET1`) and Baidu (`DATASET2`) server-side APIs, including their required parameters. Items marked with * denote Baidu APIs (at the lower part of the table). [A]: Read Mini-app Data; [B]: Send Arbitrary Messages; [C]: Data Tampering; [D]: Malicious Redirects; [E]: Resource Exhaustion; AT: Access Token; ✓ denotes the possibility of the attack using the corresponding server-side API. Impact: the impact of the attacker's invocation of the API on a mini-app and its users — determined based on the CVSS calculator (see Appendix A.2).

WeChat in terms of both hard-coding the app secrets and not configuring the IP whitelisting. On the other hand, all the Baidu mini-apps with hard-coded app secrets generated valid access tokens, meaning all the identified secrets are valid.

The access token API returned failure responses for 6,959 WeChat mini-apps, in which for 3,578 mini-apps, the app secret was invalid. For another 3,374 mini-apps, the API returned a `50002` error code stating that "the user is limited." We did not find enough information about this error code, except that, as per the documentation, the user is not authorized to use this API [53]. For the remaining 7 mini-apps, the API returned that the requesting IP address is not in the list of whitelisted IPs. The use of IP whitelisting is indeed very limited as we found out by testing all the 110,993 WeChat mini-apps, using dummy app secrets for the ones without hard-coded secrets, that only 33 mini-apps have IP whitelisting configured.

Direct invocation of server-side APIs. Attempting to call server-side APIs within the mini-app directly is not recommended by WeChat or Baidu, not because the direct invocation itself causes security issues, but because to make such calls successful the developer must involve both hard-coding app secret, and disabling IP whitelisting (in WeChat), as a mini-app client can be run from any IP address. Therefore, we checked the prevalence of such practice. We found 4,098 occurrences of direct invocations of server-side APIs from 2,317 mini-apps out of the 110,993 unpacked WeChat mini-apps. We further classify each API call based on the category [59, 60] as shown in Table 3 in Appendix. We detected a much lower number of direct invocations in Baidu mini-apps compared to WeChat. In total, there are 43 occurrences of direct server-side APIs invocations in 11 mini-apps in Baidu (mostly `getSessionKey` and `getTemplateList` APIs).

5.2.2 Unauthorized Invocation of Server-side APIs. With a valid access token, all the server-side APIs should be callable for a given mini-app. However, in practice, each API has its semantics and the mini-app's functionality and current state determines whether a specific API can be called or supported at a given time. Therefore, next, we examine individual APIs' callability for the chosen server-side APIs under each category (see Tables 5 and 6, the category names are from the official WeChat and Baidu documentation [8, 59]). We present the statistics of the successful server-side API calls, based on the chosen 26 WeChat server-side APIs, and 10 Baidu server-side APIs we tested in Tables 2, and Table 7 (in appendix). We also discuss selected per-category results for WeChat below (see Appendix A.3 for Baidu).

Customer service messages. We test this category of APIs to determine if an attacker can send messages to mini-app users and insert arbitrary media to user messages. For this category, we evaluate two APIs, `uploadTempMedia`, and `customerServiceMessage.send`. We observe that if the Customer Service Message feature is enabled for a mini-app, then `uploadTempMedia` is callable, as it requires only the media file, form data and the access token as its request parameters. The `customerServiceMessage.send` API is callable only when the openID is present. Our framework identified that openIDs are disclosed for 312 (<1%) mini-apps via the `getPrinter` API, and for 18,475 (50.7%) mini-apps via the `getFeedback` API; hence customer service messages can be forged for such mini-apps. Here, we do not consider other potential sources for obtaining openIDs such as cloud functions which may depend on the mini-app's business logic.

Cloud base HTTP API. If the API `invokeCloudFunction` is callable, for the mini-apps containing the access token and the cloud function calls in the mini-app code, our framework statically searched through each mini-app directory to identify the cloud function calls and collect their names. Among the total 36,425 mini-apps that we tested, 254 (<1%) mini-apps use cloud base, and our framework collected 202 (<1%) mini-apps with cloud functions having valid access tokens, containing a total of 992 distinct cloud functions. As several cloud functions involved update, delete operations in the cloud level, we did not invoke any of the collected cloud functions.

We also test the cloud database CRUD (Create, Read, Update, Delete) APIs to verify if an unauthorized access to the database is possible. Out of the total 11 database APIs available, we directly test only one of them – `databaseCollectionGet`, which returns only the table names from the cloud database. With these names, we only evaluate the remaining 6 chosen APIs related to cloud database based on their callability. Through statically searching the mini-app source code, we see that out of 254 (<1%) mini-apps that use cloud base, 179 (<1%) mini-apps use the cloud database. We identify only for 24 (<1%) mini-apps out of 179 mini-apps database APIs are callable, as the API requires cloud environment ID as its request parameter. The remaining 230 (<1%) mini-apps either did not have a valid cloud environment ID in the mini-app's code or the APIs returned with an error stating that the the number of requests exceeded the quota for the mini-app. We further evaluated other database APIs considering the 24 mini-apps. We find that the `databaseAddCollection` and `databaseDeleteCollection` can be performed for all the 24 mini-apps. Additionally, adding a record to the database collection, updating and deleting are also possible for these 24 mini-apps. We do not attempt to access or download any of the identified database tables apart from evaluating the callability by assessing the request parameters for the corresponding server-side APIs.

We then test for the Tencent cloud credentials API [43]. Out of the total number of mini-apps tested, we obtain the Tencent cloud API calling credentials for 11,786 (32.3%) mini-apps. For the remaining 24,639 (67.6%) mini-apps, the QCloud token API returned an error message stating that the mini-app has no cloud base privilege, meaning that the mini-app does not use Tencent cloud base. We did not test the cloud storage APIs, as when evaluated, we see that the upload link, download link and batch delete APIs require the file path and value where, in most cases, it is unknown to an unauthorized user. If the storage paths are revealed in some part of the code, it is easier to upload arbitrary files, download files and delete files from the cloud storage. In our analysis, we did not find any hard-coded storage path in the tested mini-apps.

Plug-in management. We test this category of APIs to verify if an attacker will be able to misuse the plug-in-related functionalities of a mini-app. The API `getPluginList` (`managePlugin` with `"list"` as parameter in v2 API documentation) is to retrieve the current in-use plug-ins for any mini-app. No error code was received for 17,433 (47%) mini-apps when calling `getPluginList` API. For the remaining 18,992 (52.1%) mini-apps, the API returned with an error stating that accessing the API is unauthorized, meaning that the mini-app has not configured plug-in related permissions in the developer portal. Since the plug-in app ID is public information for any plug-in, the `applyPlugin` (`managePlugin` with "apply") API is callable for these 17,433 mini-apps, as identified by our framework. This API takes in only the access token and the plug-in app ID as request parameters. For 7,242 (19%) mini-apps that make use of the plug-ins (returning a non-empty list), the `unbindPlugin` (`managePlugin` with "unbind") API will be callable.

openAPI management. We verify openAPI management APIs to determine if the API management details can be obtained and modified by the attacker. We selected one Get API `getAPIQuota` and two Modify APIs `clearQuota` and `clearQuotaByAppSecret`. Using our analysis framework, we make calls to the `getAPIQuota` API for all the mini-apps with valid access tokens. The `clearQuota` API and `clearQuotaByAppSecret` API can also be called for the mini-app with valid hard-coded app secrets and access tokens generated. Therefore, the attacker can view the used quota for every server-side API and reset the quota at any time.

Mini-apps nearby. This is a feature for business mini-apps to show up in WeChat (under mini-apps nearby) when a user is in proximity to their business location. We test this category to check if an attacker will be able to add or modify the nearby points of interest of a mini-app. We found 4,918 mini-apps (13%) had this feature enabled (via `nearbyPoi.getList`); 2,927 (8%) mini-apps had configured some POIs, for which an attacker will be able to call the `deletePOI` and `setShowStatus` APIs to delete or change the visibility of the POIs, respectively. 8,110 (22%) mini-apps returned an error stating that the mini-apps are personal mini-apps (as opposed to business mini-apps). For 23,397 (64%) mini-apps, the API returned an error stating that the nearby features are blocked.

Logistics assistant. This category of APIs allows business mini-apps to manage logistics such as the delivery of products. We chose 2 Get APIs (`getAllDelivery` and `getPrinter`) to test if an attacker will have access to the logistics information of a mini-app. For 26,190 (71.9%) mini-apps, calls to `getAllDelivery` API were successful but only 8,622 (23%) mini-apps returned the mini-app logistics delivery details as a result of this API. We conducted further tests using the `getPrinter` API, which we rely on for the openIDs to test customer message APIs. 312 (<1%) mini-apps returned data for the `getPrinter` API, which in turn is useful in calling the `updatePrinter` API, and contains the sensitive user openIDs (required for other attacks e.g., malicious redirects). For the remaining 10,235 (28%) mini-apps, both APIs returned an error stating that the API is unauthorized (i.e., the logistics assistant is not configured, or the mini-app is for personal use as opposed to business).

Updatable messages. We test this category of APIs to see if an attacker can update the messages that are already posted to users. We call the `createActivityId` API to generate a unique activity ID which can be further used to call the `setUpdatableMessage` API. Although the `setUpdatableMessage` API is always callable, we find that only 17 mini-apps use this feature.

Operations and management. We pick two Get APIs from this category and test by calling them to verify if an attacker is able to access the mini-app's server domain configuration and mini-app's feedback from the users. 18,475 (50%) mini-apps returned customer feedback as a result of the `getFeedback` API, which also returns the user's openID against each feedback. The openID returned by this API can further be exploited in the customer service message API to send phishing messages and malicious redirects.

5.2.3 Testing dangerous server-side APIs with our own mini-app. To confirm the callability of all the server-side APIs, including the Modify APIs (which we could not call in live mini-apps of other developers), we developed our own simple WeChat mini-app and published it on the WeChat mini-app platform. With this published mini-app, we enabled all the available features provided for individual developers, and we make all the Get and Modify API calls that apply to personal mini-apps. We could call all the Get APIs successfully after generating the access token. We employed the response returned by the Get APIs to call the Modify APIs. We enabled the plug-in feature for the mini-app, added a few relevant plug-ins from the developer portal and used them in our mini-app, which we later deleted using the server-side API, breaking the entire mini-app functionality. We further accessed all the cloud database related information, and updated/deleted all our stored (test) information. As a last step, we also made continuous arbitrary calls to the server-side APIs, exhausting the API calling limit to the point that the mini-app cannot make any further calls to the exhausted server-side APIs. Our proof of concept attacks confirm the exploitability of mini-apps with known app secrets. Since Baidu does not allow individual developer mini-apps (only enterprises), we were unable to confirm the callability of Modify APIs in Baidu.

5.3 Temporal Comparison

We perform two additional measurements to address another aspect of our first objective regarding the extent of the insecure development practices over time.

Prevalence over time. To learn the trend of the prevalence of the insecure development practices, we also conducted the same measurement on the more recent `DATASET3` in February 2023, and we found the app secrets in 2,894 (28.9%) mini-apps, out of which 2,572 (25.7%) mini-apps have valid secrets in them. Our additional analysis did not change the outcomes of the evaluation that we performed with the older dataset; see Table 1 for the comparison between `DATASET1` and `DATASET3`, and Table 7 in Appendix for breakdown on individual server-side API calls and their impacts.

Same-mini-app comparison. We randomly selected 100 WeChat mini-apps with hard-coded app secrets detected from the previous analysis (`DATASET1`, crawled in 2021), and downloaded their current version (on January 24, 2023 and again on March 6, 2023) through an updated reverse-engineering approach with Frida [49] (due to the technical evolution of WeChat, hooking a different library and function), and rerun our analysis for comparison.

This experiment is motivated by the fact that hard-coding app secrets (as well as not configuring IP whitelisting in WeChat) has been discouraged or even prohibited (but not enforced), only recently more strict checks were introduced [4]. So, we would like to see if there was any change to the mini-apps vulnerable to those unauthorized server-side API calls. WeChat has recently enforced the restriction on launching a mini-app when it has hard-coded app secret in it (exact timeline unknown, but this was observed coincidentally after our report was filed). We verified this by attempting to submit our own mini-app with the hard-coded app secret in plain-text. Our mini-app got rejected during the audit phase stating the obvious reason of having app secret in the code. However, with our temporal comparison, it is evident that the mini-apps that are already on the platform with the app secrets are still not patched, thus remain vulnerable to attacks.

Out of the total 100 mini-apps investigated that were identified with valid app secrets in our initial analysis, 83 still have the same app secrets hard-coded into their source code in the latest versions. Four have eliminated the app secret from the source code, yet the app secrets that were previously identified in these mini-apps are still valid (access token generated). Three mini-apps have changed their app secrets to other values, which are also hard-coded in the current versions. Among the remaining, 9 mini-apps which had valid secrets before are currently invalid and their current source code does not contain any app secrets. One mini-app with hard-coded app secret has IP whitelisting configured (previously absent).

These results for the 100 mini-apps remained the same both in our tests in January 2023 and in March 2023, showing no very significant changes in mini-apps with hard-coded app secrets that were published before the current (no hard-coded app secret) enforcement by WeChat. In the end, 90/100 mini-apps remain vulnerable.

We further tested all the previously identified WeChat mini-apps with app secrets (36,425 mini-apps) if those app secrets are still valid by trying to generate the access tokens again in March 2023. We find that out of the total tested, for 36,293 mini-apps (99.6%) the secrets remain valid. For the ones that are no longer valid, the `getAccessToken` API threw an error stating that the app secret is invalid, except for 2 mini-app which has IP whitelisting enabled and configured. However, in our recent analysis of the same dataset in July 2023, we observed a significant decrease in the percentage

of valid secrets to 16.5% (18,332 mini-apps) as a result of WeChat's restrictions on the use of app secrets. Out of these, 3,414 mini-apps leak the app secrets of other mini-apps (i.e., own app secret leakage by 14,918 mini-apps).

5.4 Implementation and Efficiency

We use Python to implement the automated analysis with 4,643 lines of code. The framework consists of three main components in common for WeChat and Baidu: (1) static searching of hard-coded app secrets and validating them, (2) calling the server-side Get APIs, and (3) evaluating the Modify APIs. We also implement the cloud base APIs testing for WeChat mini-apps. Based on our observations, we created two separate regular expression patterns for WeChat and Baidu mini-apps for finding app secrets. We run our analysis on an Ubuntu desktop (Intel i7-10700, 2.90GHz, 16GB RAM). For WeChat mini-apps, we first decrypted and unpacked them, which on average took 3.98 seconds/mini-app. The average lines of JavaScript code per mini-app (both WeChat and Baidu) are approx. 10,000 and the average time taken for finding an app secret is 3.58 seconds/mini-app, and the actual evaluation of one mini-app is approx. 6 minutes against 26 WeChat server-side APIs and approx. 4 minutes against 10 Baidu server-side APIs. To fully utilize our CPU, the analysis was carried through 7 parallel threads, and it took about 3 days to analyze all the WeChat mini-apps, and about 5 hours for Baidu mini-apps.

In terms of effectiveness of our app secret search via regular expressions, successfully generating the access tokens ensures the validity of the identified app secrets, avoiding false positives. Although as we identified in 9,244 WeChat mini-apps and in 21 Baidu mini-apps with multiple hard-coded app secrets, only one of those secrets turns out to be valid after the token generation attempt. To identify false negatives, we randomly chose 200 WeChat mini-apps and 50 Baidu mini-apps where no hard-coded app secrets were identified by our regular expression, and checked them manually. Out of the 250 mini-apps, no hard-coded app secret was found.

6 SECURITY CONSEQUENCES

Based on the analysis results presented in Sec. 5.2, in this section, we address our third objective to understand the security consequences of the unauthorized server-side API calls caused by hard-coded app secrets and absence of IP whitelisting.

6.1 Consequences from Server-side APIs

After analyzing individual server-side APIs for their callability by an attacker based on per-mini-app semantics/states, we come up with six impact categories by reviewing several sources (e.g., [31, 34]) adapted to the mini-app paradigm; see Table 2 for a complete list of the APIs tested and matched with the impact categories, and their corresponding CVSS severity scores, as we determined using the CVSS calculator (Appendix A.2). We provide further details on these categories below; see Table 4 in Appendix for the statistics of mini-apps against each consequence.

Authentication bypass and entity impersonation. Pertaining to the type of mini-app to super-app authentication, we have studied the problem of unauthorized calls to server-side APIs, and how the super-app server authenticates individual mini-apps. This is very different from the other types of authentication where a human user is involved to provide knowledge/possession/biometrics at the time of authentication. For mini-apps to get authenticated with the super-app server, which in some sense is on behalf of the mini-app developer, the developer needs to store a form of secret securely in advance to be used at runtime. The WeChat way is to recommend the use of a developer server to call their server-side APIs where the app secret can be stored, assuming the developer server is secure (which is relatively true). However, in practice, some developers make direct server-side API calls within the mini-app, which is not blocked by WeChat or Baidu (blocked in the WeChat developer portal, but can still be unchecked in the devtool for development and testing), and hence creating the need for hard-coded secrets in the mini-app package. IP whitelisting in WeChat was expected to add further restrictions on which server IPs can be used to make calls involving app secrets; however, this again required developers' understanding and engagement, which does not happen readily in practice. In consequence, an attacker having extracted the app secret from a package will be able to generate a valid access token from any system and effectively bypass mini-app to super-app authentication impersonating the mini-app developer. Super-app servers assume that any request that contains a valid app secret or access token is a legitimate request and respond with the requested data, and thus allow the attacker to interact with the super-app servers. What makes it worse is that whoever calling these APIs do not even need to possess a super-app account, due to no binding to the super-app (e.g., WeChat, Baidu) environment.

Reading mini-app data. 36,425 WeChat mini-apps from `DATASET1` and 112 Baidu mini-apps from `DATASET2` are subject to data exposure, through unauthorized server-side API calls (when the corresponding feature is enabled by the developer); see Table 2. Examples of sensitive mini-app data that can be exposed include but are not limited to mini-app data analytics, email addresses, security questions and answers, order IDs, tracking numbers and transaction information. This undermines data confidentiality of the super-app ecosystem, both for the millions of users of affected mini-apps and their developers. We refrain from measuring the extent of such leakage due to obvious ethical issues.

Mini-app data tampering. Through unauthorized server-side API calls, in addition to losing confidentiality, mini-app data can also be tampered with. For example, when a WeChat mini-app has the plug-in feature enabled and the mini-app's category matches with the plug-in's category, the `managePlugin` API can send arbitrary requests to the legitimate plug-in apps, and the same API can remove any plug-in from the live mini-app, thus breaking the mini-app's functionality (details in Sec. 5.2.2). Unauthorized access to a mini-app's cloud database can lead to more catastrophic consequences as an attacker can read/modify the entire database, including deleting it. This combined with the plug-in deletion can bring down the mini-app completely, thus affecting the availability of the mini-app. Unauthorized calls to the previously mentioned `deletePOI` and `setShowStatus` can obviously manipulate configured points of interest of a business mini-app.

In Baidu, with the APIs such as `submitResource`, and `submitSitemap`, an attacker can upload arbitrary file resources to the mini-app. This unauthorized access also enables the attacker

to create and submit fake coupons for the corresponding vulnerable mini-apps, thus adding arbitrary and fake data to the server.

Resource exhaustion attacks. As all the WeChat and Baidu mini-app server-side APIs have a quota limit (how many times an API can be called), the attacker can call these APIs repetitively until the limit is exhausted and are no longer available for the legitimate in-mini-app usage. Therefore, we can consider all the mini-apps for which an access token can be successfully generated to be vulnerable to resource exhaustion attacks. What is worse, in WeChat, by using the `getApiQuota` API, an attacker can obtain the quota for each API further facilitating the attacks, e.g., by making just enough number of calls to exhaust the limit; and the `clearQuota` API may also be exhausted for the legitimate user which can only be called 10 times per month. All these harm the availability of mini-apps in the super-app ecosystem.

Sending arbitrary messages and malicious redirects. From Customer Service Messages in Sec. 5.2.2, it is clear that an attacker can send arbitrary customer service messages to the affected WeChat mini-app users. This message can be a text, an image, an external link or a link to open another mini-app. Attackers can obtain the openIDs of an affected mini-app with the help of other server-side APIs like `getFeedback` and `getPrinter` (also cloud functions or database, in some cases), and use those openIDs to send arbitrary messages. Further, attackers can also modify the news-feed of the affected WeChat mini-apps as discussed in Sec. 5.2.2 (under "Updatable Messages"), including past messages from a user's feed. This can lead to fraud and phishing attacks, and malicious redirects. In Baidu, unlike WeChat, we did not find the leakage of openIDs, and hence, sending malicious messages to users seems infeasible.

6.2 Consequences from Cloud Functions

As WeChat cloud functions provide the developers with an online space to host business logic, they will not be part of the mini-app package. In the package, we only see where a cloud function call is made, the cloud function names, and the parameters passed to that function. These cloud functions can also be triggered by the developer's server using the server-side API `invokeCloudFunction`. As the API requires only the cloud function name, the parameters to that function and the mini-app's access token, an attacker can easily use the server-side API to invoke the cloud function by using the extracted app secret. Making such unauthorized calls can further harm business logic in the cloud, impacting all the mini-app's users.

As an example, we observed that an e-commerce mini-app with hard-coded app secret has multiple cloud functions such as `confirmCustomerOrder`, `editCustomerOrder`, `changeDeliveryAddress`, `changePaymentStatus`, and `cancelOrders`, with order ID and openID as the parameters to these functions. It appears to be possible to obtain the order ID and openID by calling the `getCurrentOrderList` cloud function, which will return a list of all orders, containing the order ID, openID, order date, total cost, and delivery address. As a result, an attacker can simply use the `curl` command on the terminal to call `invokeCloudFunction` with the identified cloud function name `getCurrentOrderList`, and obtain all the currently placed orders' order IDs and openIDs. By employing this information, the attacker can invoke other cloud functions to modify the order address to their own, cancel orders arbitrarily, edit current orders, change the payment status, etc.

Another important aspect of the cloud base is that it can be shared between other services of a WeChat mini-app developer, such as other mini-apps, and WeChat official accounts that can provide a hub for branding products and gather followers. If one of their mini-apps has its secret hard-coded in the mini-app code, using that secret, an attacker can access the common cloud base (e.g., cloud functions, database, storage) and attack or even worse, take down all the relying services of that cloud base.

7 DISCUSSION

In here, we discuss the takeaways and reflections on our measurement and analysis of the mini-app to super-app authentication problem due to developer secret leakage, as well as our limitations.

Root cause analysis. After examining the code of the large number of mini-apps and corresponding documentations, we try to further understand why such app secret leakage problems happen.

Binding through super-app. When making calls to super-app server-side APIs, the subject being authenticated is just *code* (at least in WeChat, TikTok and Baidu) on behalf of the mini-app developer, or in certain terminologies, the third-party user or the merchant. Without another aiding entity, the app secrets must be stored with the code. This is in contrast to the `getAuthCode` API of Alipay mini-apps [6], which prompts the user through the super-app to consent and returns an authorization code (equivalent to a dynamic app secret). This code can then be used with `applyToken` to get an access token to call server-side APIs. Access tokens generated this way will be per user session, "endorsed" by the super-app (as the super-app server would not assign a dynamic app secret to random requesting code, unlike the static app secret which can be pre-assigned).[1] Therefore, we can see that involving the super-app to locally authenticate the mini-app code and generate dynamic app secrets can avoid having to hard-code static app secrets, which is not the case for super-apps like WeChat and Baidu.

Intended use of server-side APIs. Despite having to hard-code app secrets as discussed above, the location matters. Hard-coding app secrets in the mini-app is considered an insecure practice according to WeChat, and the intended way is to manage the entire business logic on the developer server where the app secret is stored and used to make all the super-app server-side API calls. However, our measurement clearly showed that shifting this to mini-apps (potentially falling in the wrong hands) can lead to significant security consequences. Therefore, our observation is that mini-app developers involved insecure development practices and the super-app platform also failed to prevent them in the first place.

Comparing WeChat and Baidu with other super-app platforms. Other miniapp platforms like QQ,[2] Duoyin (Chinese TikTok), Toutiao,[3] and DingTalk[4] also offer mini-app features with

[1] The same authorization also exists in WeChat/Baidu but not for this purpose. Only when the API involves accessing super-app-stored user information, a dynamic app secret is used along with the regular app secret (using `code2Session` API for WeChat and `getsessionkey` [9] for Baidu), which still does not avoid hard-coding app secrets.

[2] QQ mini-apps - https://q.qq.com/wiki/develop/miniprogram/frame/

[3] Duoyin and Toutiao from ByteDance - https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/introduction/overview/

[4] https://open.dingtalk.com/document/orgapp/introduction-to-dingtalk-mini-programs

their own server-side APIs, and similar to WeChat, they require the usage of an access token for these APIs' invocation. QQ, being under Tencent's ownership, shares substantial similarities with WeChat in terms of API functionalities [3]. Duoyin and Toutiao, developed by ByteDance, follow a similar pattern, requiring access tokens obtained through the `getAccessToken` API with parameters like app ID, app secret, and grant type set to `"client_credentials"` [17]. DingTalk, developed by Alibaba, provides different mini-app platforms but retains a common access token acquisition process through the `gettoken` API, utilizing parameters like corpID and corpsecret [1]. Our approach for WeChat and Baidu may also apply to the above three platforms, and lead to similar attacks if the front-end code of mini-apps from these platforms discloses app secrets, despite the documentations stating otherwise [1, 3, 17].

In contrast, Paytm,[5] a prominent digital payments app in India, utilizes a different approach for access token retrieval. The `getAccessToken` API requires the client ID and client secret in the request header, along with other parameters in the request body, such as the scope, grant type, and authorization string obtained through the `paytmFetchAuthCode` JSAPI [2]. Similarly, Alipay[6] mini-apps have their own set of server-side APIs, where the access token is obtained using the `applyToken` API with specific parameters like the mini-app's ID, authClientID, grant type set to `"AUTHORIZATION_CODE"`, and the corresponding authCode acquired through the `my.getAuthCode` JSAPI [6]. It is worth noting that the acquisition of authorization codes, linked to individual users, poses a challenge for the attackers. Requiring explicit authorization makes it largely impractical to obtain users' authorization codes at a large-scale. While attackers may resort to creating malicious mini-apps to deceive users and obtain their authorization codes, replicating this process for other mini-apps is not feasible. Consequently, access tokens obtained through this grant type offer better security against the attacks considered in our study.

Mini-app to developer server authentication. Although our work is mostly centered on mini-app to super-app authentication using the app secrets, we also manually test the authentication of mini-app to its developer server. Through experimenting with our own mini-app, we confirm that the developer server does not enforce restrictions for incoming requests from mini-apps. We also confirm that it is possible for a mini-app to communicate with other mini-apps' developer servers. Thus, keeping the secrets in the developer server just shifts the authentication problem if it is not handled by the developers properly.

Ways forward. We enumerate a few potential directions below.

Replacing the developer server with cloud features. As one important advantage of mini-apps, developers can be saved from the need to deploy their own servers. They can make use of WeChat cloud base and Tencent cloud hosting to host their mini-app's backend, in which no explicit app secret or access token is required—an implicit grant is implemented. In addition, all necessary server-side APIs can be invoked from cloud functions to lessen the reliance on access tokens. However, this is secure only if the app secret has not been hard-coded, otherwise without IP whitelisting, cloud functions can be called without authorization as seen in Sec. 5.2.2. In certain cases, a developer server is not avoidable, e.g., the business may involve a large amount of data/code, not manageable or cost-effective for the cloud base; in such cases, the developers must avoid hard-coding app secrets and enable IP whitelisting.

Mandating IP whitelisting. When a developer server is necessary and can be used to make super-app server-side API calls, IP whitelisting can be enabled to only allow making calls from the IP address of the configured developer server. Despite its technical feasibility, forcing a fixed set of IP addresses may not work for developers who do not own the infrastructure or have servers of a varying nature. But this might be mandated to popular, security-critical mini-apps.

Disallowing app secret hard-coding. A straightforward way is to restrict it from the source by WeChat/Baidu. As of this writing, WeChat already prevents mini-apps from being released if hard-coded secrets are detected and the latest version of devtools [55] supports app secret detection as part of code quality analysis. However, this is not retrospective, leaving still 32.6% live mini-apps with the app secrets hard-coded. We strongly suggest that such hard-coding prohibition should be implemented for all mini-apps. Note that, we even observed from our manual code review, several mini-apps hard-coded valid app secrets for no apparent reason (i.e., no use of developer/WeChat server-side APIs), and simply add app secrets in the `globalData` object.

Disallowing server-side API invocation from mini-apps. This is already in use through server domain name restrictions [57], which requires any domain name contacted by a mini-app must be configured in the portal. By disallowing *"api.weixin.qq.com"* (WeChat) and *"openapi.baidu.com"* (Baidu) to be configured, direct server-side API calls can be prevented. It is possible that the mini-apps we have seen with these direct server-side API calls have been developed before this restriction was enforced.

Switching to super-app bound dynamic secrets. As mentioned earlier, Paytm and Alipay's use of user-bound (as opposed to mini-app-bound in WeChat, Baidu) authorization tokens, generated and managed by the super-app, replaces fixed app secrets. While binding to a user involves the user to click, this may be necessary since otherwise the initial trust remains a question, e.g., without prompting the user explicitly, ensuring the request's authenticity is difficult.

Limitations. The mini-apps obtained from the WeChat platform whether collected manually or through the use of crawlers, might typically undergo obfuscation, as described in Sec. 4. The unpacker utilized in our framework, which is widely recognized within the community, is considered the most popular tool for unpacking mini-apps. However, while utilizing this tool, approximately 4% of WeChat mini-apps were not unpacked completely, leading to the generation of obfuscated files. This affects the analysis, as our framework performs static analysis on the mini-app code to detect the hard-coded app secrets. If app secrets are present within the obfuscated code, our framework may inadvertently overlook them during the analysis. Secondly, as part of our analysis, we installed the WeChat client application on a rooted Android device to access the mini-app packages. Our recent experiment showed that any WeChat account created on a rooted device would be blocked from further use on that device.

[5]Paytm - https://business.paytm.com/docs/miniapps/overview
[6]Alipay - https://miniprogram.alipay.com/docs/

8 DISCLOSURE & ETHICAL CONSIDERATIONS

We have taken careful consideration to contemplate the ethical implications when designing our analysis framework. We had to generate the access tokens for the corresponding 53,377 mini-apps comprising both WeChat and Baidu, to verify if the identified secrets can lead to security and privacy exposures; note that secrets can be changed by developers, and enabling IP whitelisting can make leaked secrets unexploitable. We refrained from calling any Modify API to avoid modification or deletion of mini-app data; we verified such APIs only on our test mini-app we created for research. To understand and measure the real and immediate threat to mini-apps with valid access tokens, we call only those Get APIs whose results can be used to call the Modify APIs in order to verify if a mini-app's data can be modified. We also did not arbitrarily call any server-side API to avoid resource exhaustion. According to the regulations from our university's Research Ethics Unit, we took multiple precautions to prevent the exposure of secrets from our collection database. We did not store any unnecessary data returned from the calls and erased the minimally captured data appropriately after the analysis. We also have disclosed our findings to Tencent and Baidu, including the list of all the server-side APIs that can be abused when the app secret is known. Tencent notified us, stating, "For increments, we will have a review mechanism, and if hard-coded is detected, it will not be approved for release. *For inventory, we will notify developers to fix it, but some developers don't fix it, such is the case in your report.*" Although we successfully reported to Baidu using their portal, it was more complicated than Tencent. The email to their security team (security@baidu.com) was declined because of authentication requirement, and the report required a Chinese phone number to be bound to the reporting account.

9 RELATED WORK

Zhang et al. [70] implemented the first large-scale WeChat mini-app crawler and performed an empirical study on the crawled mini-apps. Also, Hao et al. [23] studied the key features, system architecture, and development prospects of WeChat mini-apps. Below we summarize the studies more relevant to our work.

Security analysis on mini-apps. Zhang et al. [68] analyzed the mini-apps permission model of 9 super-apps, and found six vulnerabilities with at least one security issue in each super-app; they also presented three proof of concept attacks that can reveal user location, contacts, and clipboard content to unauthorized mini-apps.. Lu et al. [28] studied security vulnerabilities in 11 super-apps based on the resource management between the super-app and the mini-app. Further, Zhang et al. [69] identified the novel identity confusion based on the app ID, domain name, and capability in 47 high-profile super apps based on the identity check adopted by the super-apps. They demonstrated several attacks based on this vulnerability, such as installing malware on victim's phone, stealing victim's financial accounts, and bypassing security patches. Furthermore, the National Computer Network Emergency Response Technical Team tested around 50 personal banking mini-apps from WeChat and reported that more than 60% of the mini-apps did not encrypt the user information both in the device and while it was transmitted [39, 44]. Yang et al. [67] implemented CMRFScanner to identify the cross mini-app request forgery (CMRF) attacks, a novel attack leading to several security consequences, e.g., privileged data access, information leakage, and shopping for free. Wang et al. [48] developed a consistency analysis framework to identify the inconsistencies between privacy policies and the data practice in the mini-apps. They crawled 10,000 mini-apps from WeChat and extracted 2,998 mini-apps in which they found 2,680 mini-apps did not meet the policy requirements.

Unlike the prior studies on the security of mini-apps, our study focuses on the mini-app's *developer server to super-app* authentication problem, mainly affecting mini-app data that can be user-specific or shared among all users of a mini-app, instead of local resources on a user's phone. By measuring the extent of insecure development practices (defeating such authentication), we analyze how the resulting unauthorized server-side API calls can cause severe security issues, including bringing down the mini-apps and their services. In a concurrent study, Zhang et al. [71] identified 40,880 mini-apps (approximately 1.18% of the total 3,450,586) that leaked their own app secrets. However, their study focused on identifying vulnerable mini-apps that leaked their own secrets. We considered both this self-leakage and the leakage of other mini-apps' secrets, resulting in a significantly higher percentage (32.8%) of app secret leakage (albeit mostly self-leakage, see the last paragraph in Sec. 5.3).

Identifying hard-coded secrets. Sinha et al. [41] provided practical solutions to detect, prevent and fix API key leaks in the source code repositories (GitHub). Meli et al. [29] studied the large-scale secrets leakage with GitHub Search API and BigQuery snapshot, for a period of six months, especially targeting 11 different platforms. CredMinder [16] is aimed at finding credentials that are leaked in Android apps, by using code analysis instead of string matching, in order to identify credentials even when they are obfuscated. Wen et al. [63] developed iCredFinder to fix the gap of credential leak detection in iOS apps. Saha et al. [38] developed a generalized machine learning-based framework with regular expressions to identify the secrets in source code, and analyzed 24 different types of secrets with precision and recall rate of 59% and 97% respectively. Several other studies focus on OAuth and SSO-related vulnerabilities [24, 46, 47]; e.g., MoSSOT [40] detects app secret and access token leakage from the network traffic between the relying party and provider apps, which also includes WeChat and its relying third-party apps (but not the mini-apps).

10 CONCLUSION

We have presented the app secret leakage issues in popular mini-app platforms, which is the result of non-compliance of mini-apps with the super-app's security guidelines. In order to identify this non-compliance in a large number of mini-apps, we developed a tool to detect the hard-coded app secrets in the mini-app's source code and accessibility of the server APIs using the identified secret. We analyzed 110,993 WeChat mini-apps and 10,000 Baidu mini-apps, out of which 36,425 WeChat mini-apps and 112 Baidu mini-apps are found to have valid app secrets hard-coded. We also discussed how these hard-coded secrets can lead to the misuse of server-side APIs by defeating the authentication mechanisms implemented by the super-apps. Additionally, we used the measurement results to identify potential attack vectors on the identified vulnerable mini-apps that could lead to severe security consequences at the mini-app level.

REFERENCES

[1] DingTalk mini-app API documentation. Available at: https://open.dingtalk.com/document/orgapp/how-to-call-apis.

[2] Paytm mini-app API documentation. Available at: https://business.paytm.com/docs/api/miniapps/login-flow/getaccesstoken.

[3] QQ mini-app API Documentation. Available at: https://q.qq.com/wiki/develop/miniprogram/server/open_port/port_use.html.

[4] WeChat devtool stable version update log. Available at: https://developers.weixin.qq.com/miniprogram/dev/devtools/stable.html.

[5] Adchina. 2022. The power of the Baidu super-app. Available at: https://www.adchina.io/what-is-baidu/.

[6] Alipay. Get access token API. Available at: https://miniprogram.alipay.com/docs/miniprogram/mpdev/v2_applytoken.

[7] Alipay. 2022. Mini-app framework demystified. Available at: https://juejin.cn/post/7137478354042617869.

[8] Baidu. Get access Token API. Available at: https://smartprogram.baidu.com/docs/develop/serverapi/serverapilist/.

[9] Baidu. Get session key API. Available at: https://smartprogram.baidu.com/docs/develop/api/open/getSessionKey/.

[10] Baidu. Mini-app directory structure. Available at: https://smartprogram.baidu.com/docs/develop/framework/app_service/.

[11] Farzana Ahamed Bhuiyan and Akond Rahman. 2020. Characterizing co-located insecure coding patterns in infrastructure as code scripts. In *IEEE/ACM Conference on Automated Software Engineering Workshops (ASE'20)*. Melbourne, Australia.

[12] ByteDance. Duoyin. https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/introduction/overview/.

[13] ByteDance. Tiktok - Overseas version of Duoyin. https://www.tiktok.com/.

[14] Ao Cheng, Gang Ren, Taeho Hong, Kichan Nam, and Chulmo Koo. 2019. An exploratory analysis of travel-related WeChat mini program usage: affordance theory perspective. In *Information and Communication Technologies in Tourism (ENTER'21)*. Cham.

[15] Chinese article. 2022. Extracting WeChat mini-apps under Windows. Online blog article (in Chinese). Available at: https://zone.huoxian.cn/d/883-pcfirda.

[16] Shuaike Dong, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, Xiaofeng Wang, and Kehuan Zhang. 2018. Understanding Android obfuscation techniques: A large-scale investigation in the wild. In *Security and Privacy in Communication Networks (SecureComm'18)*. Cham.

[17] Duoyin. Get access Token API. Available at: https://microapp.bytedance.com/docs/zh-CN/mini-app/develop/server/interface-request-credential/get-access-token/.

[18] Duoyin. Safety guidelines. Available at: https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/develop/guide/anquankaifa/.

[19] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. 2020. An empirical evaluation of GDPR compliance violations in Android mHealth apps. In *IEEE 31st International Symposium on Software Reliability Engineering (ISSRE'20)*. Coimbra, Portugal.

[20] GitHub. WeChat mini-apps unpacker. Available at: https://github.com/Ryan-Miao/wxappUnpacker.

[21] GitHub. Wxapkg decryptor. Available at: https://github.com/BlackTrace/pc_wxapkg_decrypt.

[22] Mingjia Guo, Ru-De Liu, Yi Ding, Biying Hu, Rui Zhen, Ying Liu, and Ronghuan Jiang. 2018. How are extraversion, exhibitionism, and gender associated with posting selfies on WeChat friends' circle in Chinese teenagers? *Personality and Individual Differences* 127 (June 2018), 114–116.

[23] Lei Hao, Fucheng Wan, Ning Ma, and Yicheng Wang. 2018. Analysis of the development of WeChat mini program. *Journal of Physics: Conference Series* 1087, 6 (September 2018), 062040.

[24] Pili Hu, Ronghai Yang, Yue Li, and Wing Cheong Lau. 2014. Application impersonation: problems of OAuth and API design in online social networks. In *Proceedings of the Second ACM Conference on Online Social Networks (COSN'14)*. Dublin, Ireland.

[25] Che Hui Lien and Yang Cao. 2014. Examining WeChat users' motivations, trust, attitudes, and positive word-of-mouth: Evidence from China. *Computers in human behavior* 41 (December 2014), 104–111.

[26] Yubei Lin, Jingyan Qiu, and Pingping Chen. 2020. Exploration and practice on intelligent teaching patterns based on WeChat mini program. In *Proceedings of the 9th International Conference on Educational and Information Technology (ICEIT'20)*. Oxford, United Kingdom.

[27] Yanyan Liu, Danyu Li, Haishan Ruan, Yun Hu, and Nanping Shen. 2022. Development and usability test of a symptom management WeChat mini program for parents of children with cancer. *Asia-Pacific Journal of Oncology Nursing* 9, 12 (December 2022), 100166.

[28] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying resource management risks in emerging mobile app-in-app ecosystems. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'20)*. Virtual Event, USA.

[29] Michael Meli, Matthew R McNiece, and Bradley Reaves. 2019. How bad can it git? Characterizing secret leakage in public GitHub repositories. In *Network and Distributed Systems Security Symposium (NDSS'19)*. San Diego, CA, USA.

[30] Microsoft. 2023. Detect secrets - credentials scanning tool. Available at: https://microsoft.github.io/code-with-engineering-playbook/continuous-integration/dev-sec-ops/secret-management/recipes/detect-secrets/.

[31] Mitre. CWE top 25. Available at: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.

[32] NPM. wx-server-sdk - Cloud call npm Package. Available at: https://www.npmjs.com/package/wx-server-sdk.

[33] NVD. CVSS calculator. Available at: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[34] OWASP. OWASP top 10 API. Available at: https://owasp.org/www-project-api-security/.

[35] Andrea Possemato and Yanick Fratantonio. 2020. Towards HTTPS everywhere on Android: We are not there yet. In *29th USENIX Security Symposium (USENIX Security 20) (USENIX'20)*. Boston, MA, USA.

[36] Postman. Postman API platform. Available at: https://www.postman.com/.

[37] Property Guru for Business. 2023. The power of the WeChat super-app. Available at: https://www.propertyguruforbusiness.com/publications/the-power-of-the-wechat-super-app.

[38] Aakanksha Saha, Tamara Denning, Vivek Srikumar, and Sneha Kumar Kasera. 2020. Secrets in source code: Reducing false positives using machine learning. In *Conference on Communication Systems & Networks (COMSNETS'20)*. Bengaluru, India.

[39] Scmp.com. 2021. WeChat mini programs for banking pose 'significant' risks of personal data leakage. Available at: https://www.scmp.com/tech/tech-trends/article/3142239/wechat-mini-programs-banking-pose-significant-risks-personal-data.

[40] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau. 2019. MoSSOT: An automated blackbox tester for single sign-on vulnerabilities in mobile applications. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (Asia CCS'19)*. Auckland, New Zealand.

[41] Vibha Singhal Sinha, Diptikalyan Saha, Pankaj Dhoolia, Rohan Padhye, and Senthil Mani. 2015. Detecting and mitigating secret-key leaks in source code repositories. In *IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR'15)*. Florence, Italy.

[42] Sixthtone. 2020. China's 'mini-apps' have big privacy issues, report says. Available at: https://www.sixthtone.com/news/1006196.

[43] Tencent. Tencent cloud API overview. Available at: https://cloud.tencent.com/document/api/876/34809.

[44] Time Business News. 2021. WeChat mini-apps Risk Data Leaks. Available at: https://timebusinessnews.com/wechat-mini-apps-risk-data-leaks/.

[45] U.S. department of health and human services. 2018. The Belmont report - Ethical principles and guidelines for the protection of human subjects of research. Available at: https://www.hhs.gov/ohrp/regulations-and-policy/belmont-report/read-the-belmont-report/index.html.

[46] Hui Wang, Yuanyuan Zhang, Juanru Li, and Dawu Gu. 2016. The achilles heel of OAuth: a multi-platform study of OAuth-based authentication. In *Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC'16)*. Los Angeles, California, USA.

[47] Hui Wang, Yuanyuan Zhang, Juanru Li, Hui Liu, Wenbo Yang, Bodong Li, and Dawu Gu. 2015. Vulnerability assessment of OAuth implementations in Android applications. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC'15)*. Los Angeles, CA, USA.

[48] Yin Wang, Ming Fan, Junfeng Liu, Junjie Tao, Wuxia Jin, Qi Xiong, Yuhao Liu, Qinghua Zheng, and Ting Liu. 2023. Do as you say: Consistency detection of data practice in program code and privacy policy in mini-app. Available at: https://arxiv.org/pdf/2302.13860.pdf.

[49] Web archive. 2022. Extracting WeChat mini-apps using frida. Online blog article (in Chinese). Available at: https://web.archive.org/web/20221215183356/https://www.ljczero.top/article/2022/9/5/144.html.

[50] WeChat. Cloud base. Available at: https://developers.weixin.qq.com/miniprogram/dev/wxcloud/basis/capabilities.html.

[51] WeChat. Cloud initialization. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/wxcloud/guide/init.html.

[52] WeChat. code2Session API. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/api-backend/open-api/login/auth.code2Session.html.

[53] WeChat. Error codes developer error codes. Available at: https://developers.weixin.qq.com/doc/oplatform/en/Return_codes/Return_code_descriptions.html.

[54] WeChat. get access token API. Available at: https://developers.weixin.qq.com/miniprogram/dev/OpenApiDoc/mp-access-token/getAccessToken.html.

[55] WeChat. IDE devtool. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/devtools/download.html.

[56] WeChat. Mini-app directory structure. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/framework/structure.html.

[57] WeChat. Mini-app server domain name information. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/framework/ability/network.html.
[58] WeChat. Safety guidelines by WeChat. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/framework/security.html#Code-Management-and-Leaks.
[59] WeChat. Server-side API classification. Available at: https://developers.weixin.qq.com/miniprogram/en/dev/api-backend/.
[60] WeChat. Server-side API classification v2. Available at: https://developers.weixin.qq.com/miniprogram/dev/OpenApiDoc/.
[61] WeChat. Tencent cloud hosting. Available at: https://developers.weixin.qq.com/minigame/dev/wxcloudrun/src/practice/call.html.
[62] WeChat. 2023. WeChat. Available at: https://www.wechat.com/.
[63] Haohuang Wen, Juanru Li, Yuanyuan Zhang, and Dawu Gu. 2018. An empirical study of SDK credential misuse in iOS apps. In *25th Asia-Pacific Software Engineering Conference (APSEC) (APSEC'18)*. Nara, Japan.
[64] Wikipedia. Baidu. Available at: https://en.wikipedia.org/wiki/Baidu.
[65] Wikipedia. ICP license. Available at: https://en.wikipedia.org/wiki/ICP_license.
[66] Wenbo Yang, Juanru Li, Yuanyuan Zhang, and Dawu Gu. 2019. Security analysis of third-party in-app payment in mobile applications. *Journal of Information Security and Applications* 48 (October 2019), 102358.
[67] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2022. Cross miniapp request forgery: Root causes, attacks, and vulnerability detection. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'22)*. Los Angeles, CA, USA.
[68] Jianyi Zhang, Leixin Yang, Yuyang Han, Zhi Sun, and Zixiao Xiang. 2022. A small leak will sink many ships: Vulnerabilities related to mini programs permissions. In *Syposium on Security, Trust, & Privacy in Computing (COMPSAC'23)*. Torino, Italy.
[69] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. 2022. Identity confusion in webview-based mobile app-in-app ecosystems. In *31st USENIX Security Symposium (USENIX'22)*. Boston, MA.
[70] Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin. 2021. A measurement study of Wechat mini-apps. *ACM SIGMETRICS Performance Evaluation Review* 5, 2 (June 2021), 1–25.
[71] 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 ACM SIGSAC Conference on Computer and Communications Security (CCS'23)*. Copenhagen, Denmark.
[72] Jinyang Zheng, Zhengling Qi, Yifan Dou, and Yong Tan. 2019. How mega is the mega? Exploring the spillover effects of WeChat using graphical model. *Information Systems Research* 30, 4 (December 2019), 1343–1362.

A APPENDIX A

A.1 WeChat Login

`wx.login` is one of the several APIs provided by WeChat for the developers to interact with the WeChat's native app functions and services (also termed as "JSAPI"). Whenever the `wx.login` interface is called in a miniapp, the interface will return a random temporary WeChat login credential (termed as "JS Code") which is valid for only 5 minutes. The JS Code is for onetime use only. This JS Code is then sent to the developer's server using the interface wx.request, from where the WeChat's Code2Session API [52] is called. The WeChat's back-end server returns the session key and Open ID of the miniapp user to the developer server. Figure 3 illustrates the recommended flow of the WeChat Code2Session user authentication API. For security reasons, this session key should not be returned to the miniapp as recommended by WeChat [52]. The session key will expire only when the miniapp user does not use the corresponding miniapp for a long time. This code to session login interface is to authenticate the miniapp users to the WeChat, and a custom login status is determined to the miniapp.

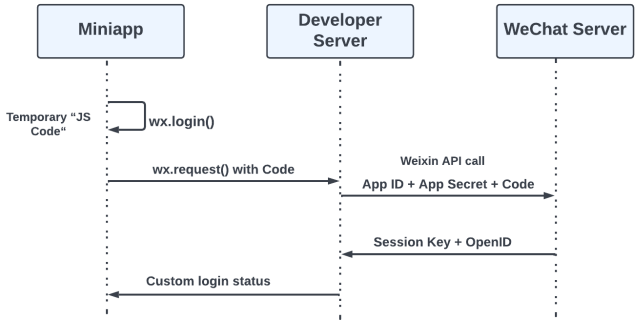

Figure 3: WeChat Code2Session - Login API

A.2 CVSS Scores

We make use of the CVSS (Common Vulnerability Scoring System) score calculator from NVD [33] to calculate the base metrics (confidentiality, integrity and availability), and estimate the security consequence of each super-app server-side API call. We set our attack vector as *network* and attack complexity as *low*, as calling these APIs does not require any special privileges and involves no user interaction. The scope of attack does not change in most cases, except for the WeChat cloud base related attacks, where a vulnerable mini-app could affect other non-vulnerable mini-apps or WeChat official accounts; see Sec. 6.2. The CVSS scores are summarized in Table 2 (and Table 7 in Appendix). It is to be noted that by the impact of medium and low, it defines the impact of the attack on the mini-app and the user. This does not change the attacker capabilities and hence, for executing all these super-app server-side APIs, the attacker requires only a network connection.

A.3 Unauthorized Invocation of Baidu Server-side APIs

Message templates. We test this category of APIs to verify if the message templates of the mini-apps can be accessed by an attacker. We see from our analysis that the `getTemplateLibraryList` API returns valid values for all 112 (100%) mini-apps, and hence it is possible to call the `addTemplate` API for these 112 mini-apps, as the output of the former API can be used as an input for the latter. Using our framework, we call `getTemplateList` API for all 112 mini-apps, which returned valid values for 74 (66%) mini-apps, and thus it is possible to call `deleteTemplate` for these 74 mini-apps.

Traffic distribution resources. Using this category of APIs, a mini-app developer can distribute the resources for a mini-app across different paths. We test this list of APIs to check if an attacker is capable of submitting resources such as image files to the Baidu server. We did not actually make calls using these APIs (`submitResource`, `submitSitemap`, `interfaceSubmission`, `submitsku`), but verified only the callability. These calls require several parameters, which are all user-controlled values, and thus the APIs can be called by the attacker for all the 112 (100%) mini-apps with valid app secrets.

Coupons. We test the APIs under this category to verify if an attacker can create and manage Baidu coupons (provided by Baidu

to the mini-app developers, who then manage and distribute the coupons to the mini-app users). We do not call any APIs under this category to avoid adding anything to the server arbitrarily, and thus only check the callability of these APIs (`createCoupon`, `submitCoupon`, and `ManageCoupon`). For all 112 (100%) mini-apps with valid app secrets, these APIs are callable, hence enabling an attacker to manipulate the coupons.

```
onDateChange: function(e) {
  var t = this;
  wx.request({
      url: "https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=" + this.data.appid + "&secret=" + this.data.secret,
      method: "GET",
      success: function(e) {
          console.log("成功", e), t.setData({
              accessToken: e.data.access_token
          });
      },
      fail: function(e) {
          console.log("失败 ", e);
      }
  }), this.setData({
      selectDate: e.detail
  });
},
```

Figure 4: Example code snippet of a WeChat mini-app using server-side API calls in the code with hard-coded secret

```
getPhoneNumber: function(a) {
    if (a.detail.iv || a.detail.encryptedData) {
        var e = this;
        wx.login({
            success: function(t) {
                t.code ? wx.request({
                    url: getApp().globalData.domain + "apicns/index.php/Index/GetOpenid",
                    method: "POST",
                    data: {
                        code: t.code,
                        appid: getApp().globalData.appid,
                        secret: getApp().globalData.secret
                    },
                    header: {
                        "content-type": "application/x-www-form-urlencoded"
                    },
                    success: function(t) {
                        wx.request({
                            url: getApp().globalData.domain + "apicns/demo.php",
                            method: "POST",
                            data: {
                                session_key: t.data.session_key,
                                iv: a.detail.iv,
                                encryptedData: a.detail.encryptedData,
                                appid: getApp().globalData.appid,
                                secret: getApp().globalData.secret
                            },
                            header: {
                                "content-type": "application/x-www-form-urlencoded"
                            },
                            dataType: "json",
```

Figure 5: An example of a WeChat mini-app calling a developer server API for WeChat login with a hard-coded secret that is passed from the mini-app.

API Categories	# miniapps
Mini-app User Login	1723 (42%)
Access Token	554 (13.5%)
Customer Service	429 (10.4%)
Dynamic Messages	319 (7.7%)
Generate QR code	285 (6.95%)
Message Templates	279 (6.8%)
Image Processing	176 (4.3%)
Security Check	134 (3%)
Live Broadcast	81 (1.97%)
Logistics	77 (1.87%)
Data Analytics	41 (1%)

Table 3: WeChat mini-apps with the direct invocation of server-side APIs

Consequences	DATASET1	DATASET3	DATASET2
# read mini-app data	36,425	2,572	112
# send msgs.	18,241	199	-
# data tampering	36,425	2,572	112
# malicious redirects	23,162	573	-
# resource exhaustion	36,425	2,572	112

Table 4: Relationship between number of WeChat (DATASET 1 and DATASET 3) and Baidu (DATASET 2) mini-apps against the identified security consequences.

API Category	Get API	Modify API
Access Token	getAccessToken	-
Customer Service Message	-	uploadTempMedia customerServiceMessage.send
Updatable Message	createActivityId	setUpdatableMsg
Miniapp Plug-in		managePlugin managePluginApplication
Miniapps Nearby	getNearbyPoiList	deleteNearbyPoi setShowStatus
Logistics Assistant	getPrinter getAllDelivery	updatePrinter
openAPI Management	getApiQuota	clearQuotaByAppSecret clearQuota
Operations and Maintenance	getFeedback getDomainInfo	
Cloud Base	databaseCollectionGet getQcloudToken	invokeCloudFunctions databaseCollectionAdd databaseCollectionDelete databaseAdd databaseDelete databaseUpdate databaseQuery

Table 5: List of get and modify WeChat server-side APIs evaluated in our measurement study chosen based on the availability of request parameters.

API Category	Get API	Modify API
Access Token	getAccessToken	-
Message Templates	getTemplateList	addTemplate deleteMessageTemplate
Traffic Distribution Resources	-	submitResource submitSitemap interfaceSubmission submitsku
Coupons	-	createCoupon submitcoupon ManageCoupon

Table 6: List of get and modify Baidu server-side APIs evaluated in our measurement study chosen based on the availability of request parameters.

WeChat server-side APIs	Required parameters	# miniapps	[A]	[B]	[C]	[D]	[E]	Impact
clearQuotaByAppSecret	appID, appSecret	2,572			✓		✓	High
clearQuota	AT, appID	2,572			✓		✓	High
managePlugin	AT, pluginAppID	433	✓		✓		✓	High
deleteNearbyPoi	AT, poiID	303			✓		✓	High
setShowStatus	AT, poiID	303			✓		✓	High
managePluginApplication	AT, appID	179	✓		✓		✓	High
invokeCloudFunctions	AT, CloudFunctionName	65	✓	✓	✓		✓	High
databaseCollectionGet	AT, CloudEnv	21	✓				✓	High
databaseCollectionAdd	AT, CloudEnv, CollectionName	21			✓		✓	High
databaseCollectionDelete	AT, CloudEnv, CollectionName	21			✓		✓	High
databaseAdd	AT, CloudEnv	21			✓		✓	High
databaseDelete	AT, CloudEnv	21			✓		✓	High
databaseUpdate	AT, CloudEnv	21			✓		✓	High
databaseQuery	AT, CloudEnv	21	✓		✓		✓	High
setUpdatableMsg	AT	4		✓	✓	✓	✓	High
uploadTempMedia	AT	2,572			✓		✓	Medium
getApiQuota	AT, cgi_path	2,572	✓				✓	Medium
getDomainInfo	AT	2,101	✓				✓	Medium
getAllDelivery	AT	252	✓				✓	Medium
customerServiceMessage.send	AT, openID	151		✓		✓	✓	Medium
getPrinter	AT	114	✓				✓	Medium
updatePrinter	AT, openID	114			✓		✓	Medium
getFeedback	AT	93	✓				✓	Medium
getQcloudToken	AT	59	✓				✓	Medium
createActivityId	AT	2,572	✓				✓	Low
getNearbyPoiList	AT	303	✓				✓	Low

Table 7: Statistics of unauthorized callable WeChat server-side APIs for `DATASET3`. AT: Access Token; [A]: Read Mini-app Data; [B]: Send Arbitrary Messages; [C]: Data Tampering; [D]: Malicious Redirects; [E]: Resource Exhaustion; ✓ denotes the possibility of the attack using the corresponding -side API.